TornadoNoC: A Lightweight and Scalable On-Chip Network Architecture for the Many-Core Era

JUNGHEE LEE, Georgia Institute of Technology
 CHRYSOSTOMOS NICOPOULOS, University of Cyprus
 HYUNG GYU LEE, Daegu University, South Korea
 JONGMAN KIM, Georgia Institute of Technology

The rapid emergence of Chip Multi-Processors (CMP) as the de facto microprocessor archetype has highlighted the importance of scalable and efficient on-chip networks. Packet-based Networks-on-Chip (NoC) are gradually cementing themselves as the medium of choice for the multi-/many-core systems of the near future, due to their innate scalability. However, the prominence of the debilitating power wall requires the NoC to also be as energy efficient as possible. To achieve these two antipodal requirements—scalability and energy efficiency—we propose TornadoNoC, an interconnect architecture that employs a novel flow control mechanism. To prevent livelocks and deadlocks, a sequence numbering scheme and a dynamic ring inflation technique are proposed, and their correctness formally proven. The primary objective of TornadoNoC is to achieve substantial gains in (a) *scalability* to many-core systems and (b) the *area/power* footprint, as compared to current state-of-the-art router implementations. The new router is demonstrated to provide better scalability to hundreds of cores than an ideal single-cycle wormhole implementation and other scalability-enhanced low-cost routers. Extensive simulations using both synthetic traffic patterns and real applications running in a full-system simulator corroborate the efficacy of the proposed design. Finally, hardware synthesis analysis using commercial 65nm standard-cell libraries indicates that the area and power budgets of the new router are reduced by up to 53% and 58%, respectively, as compared to existing state-of-the-art low-cost routers.

Categories and Subject Descriptors: C.1.2 [Multiple Data Stream Architectures]: Interconnection Architectures

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Network-on-Chip, ring-based interconnection network, low-cost router architecture, deflection-based routing

ACM Reference Format:

Lee, J., Nicopoulos, C., Lee, H. G., and Kim, J. 2013. TornadoNoC: A lightweight and scalable on-chip network architecture for the many-core era. *ACM Trans. Architect. Code Optim.* 10, 4, Article 56 (December 2013), 30 pages.
 DOI: <http://dx.doi.org/10.1145/2555289.2555312>

1. INTRODUCTION

The last decade has witnessed a major paradigm shift in the design of microprocessors. Computer architects have shifted their focus from complex, monolithic uncore CPUs to multicore implementations employing several—but leaner—processing elements.

New article, not an extension of a conference paper.

Authors' addresses: J. Lee and J. Kim, School of Electrical and Computer Engineering, Georgia Institute of Technology, USA; C. Nicopoulos, Department of Electrical and Computer Engineering, University of Cyprus; H. G. Lee, School of Computer and Communication Engineering, Daegu University, South Korea.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481 or permissions@acm.org.

© 2013 ACM 1544-3566/2013/12-ART56 \$15.00

DOI: <http://dx.doi.org/10.1145/2555289.2555312>

Rapid technological advancements have enabled immense transistor integration densities, which enable the integration of a large number of cores. Tilera Corp. has already integrated 100 cores on a single die [TILE-Gx Family 2012], while a research product, Rigel [Kelm et al. 2009], has demonstrated that even 1,000-core hardware accelerators are feasible. On the other hand, the power wall [Esmailzadeh et al. 2011] forces each core to be extremely power efficient. A heterogeneous architecture that employs a small number of powerful cores along with a large number of power-efficient small cores has also appeared as an emerging paradigm [Lee et al. 2012]. It is expected that the number of cores on a single die will continue to increase, but the power efficiency of each core must also be enhanced.

The shift from multi- to many-core microprocessors puts enormous strain on the on-chip communication backbone. The on-chip communication fabric of the near future is required to meet two opposing requirements: effortless *scalability* with the growing number of cores and *power efficiency*.

Packet-based Networks-on-Chip (NoCs) have emerged as the most viable candidates for a scalable on-chip communication infrastructure. However, their overhead in a many-core architecture could be significant. It is reported that the power consumption of the interconnect accounts for up to 40% in Tilera's chips [Fallin et al. 2011]. To reduce the overhead of routers, there has been a substantial amount of research in the reduction, or even elimination, of buffers within the NoC [Fallin et al. 2011; Moscibroda and Mutlu 2009; Fallin et al. 2012; Gómez et al. 2008; Hayenga et al. 2009; Michelogiannakis et al. 2009; Nicopoulos et al. 2006]. As an alternative approach, a low-cost ring-based router [Kim 2009] has also been proposed and demonstrated to be very efficient. Unfortunately, all of the previous attempts at designing low-cost routers have succeeded in only reducing the *cost*, but the resulting architectures are not as *scalable* as conventional buffered NoC routers. It has been shown that the performance of low-cost routers is comparable to conventional routers at low to medium injection rates, but not at high injection rates [Fallin et al. 2011; Moscibroda and Mutlu 2009; Fallin et al. 2012]. Since the network can saturate even at low injection rates in a large network, this implies that the aforementioned solutions' scalability is not up to par with conventional routers.

In an effort to simultaneously tackle both opposing goals (scalability and power efficiency), we propose TornadoNoC,¹ a ring-based interconnect architecture that employs a novel flow control mechanism. For small-scale multicores (up to 64 cores), the primary goal of TornadoNoC is to maintain the same levels of performance—ultra-low latency and high throughput—as current state-of-the-art router implementations, but at a *much lower area/power cost*. For many-core designs (more than 64 cores), TornadoNoC aims to achieve substantially *better scalability* to hundreds of cores while still incurring minimal overhead. Overall, TornadoNoC can be viewed as a feasible way to contain the NoCs' footprint in the many-core realm while still providing exceptional performance.

The main contributions of this article are:

- The introduction of a lightweight and scalable on-chip network** that supports virtual channels with minimal hardware cost. Virtual Channels (VCs) are instrumental in minimizing Head-of-Line (HoL) blocking and avoiding protocol-level deadlocks (e.g., pertaining to the cache coherence protocol). Extensive simulation experiments demonstrate that the proposed ring-based router architecture exhibits real-workload performance (up to 64 cores) that is nearly identical to various state-of-the-art router designs while incurring significantly **lower over-**

¹The name *TornadoNoC* is inspired by the packets rotating around the rings (imitating a tornado), as they are being deflected by the proposed Spillover Flow Control mechanism.

head. In larger-scale many-core systems (with more than 64 cores), the proposed design is demonstrated to sustain markedly **better scalability** to *hundreds of cores*, as compared to existing designs. The alternative routers included in our experiments are an ideal single-cycle full-sized conventional router, a buffered deflection router [Fallin et al. 2012] (whose performance is known to be better than bufferless deflection routers [Moscibroda and Mutlu 2009]), and a ring-based router [Kim 2009] with multiple parallel buffers (to facilitate virtual channel support).

- A novel Spillover Flow Control mechanism.** The proposed deflection-based flow control scheme allows flits to “spill over” into the network when buffers are full, but flits are not allowed to reach their destination through a different path. The deflection routing schemes typically employed by bufferless routers [Fallin et al. 2011; Moscibroda and Mutlu 2009] allow misrouting when multiple flits are headed to the same output port. Instead, we allow deflection only within a ring and exploit this characteristic as a flow control mechanism that contributes to the excellent scalability characteristics of the new router architecture.
- Formally proven livelock and deadlock freedom.** To ensure livelock and deadlock freedom under deflection routing, mechanisms such as Oldest-First [Moscibroda and Mutlu 2009], Golden Packet [Fallin et al. 2011], and Retransmit-Once [Fallin et al. 2011] have been proposed. However, their correctness has not been formally proven. In this article, a *sequence numbering mechanism* and a *dynamic ring inflation technique* are tasked with ensuring livelock and deadlock freedom, and their correctness is formally proven.

The rest of the article is organized as follows: Section 2 discusses related prior work in the domains of low-latency router architectures and ring-based NoC designs. Section 3 describes the microarchitecture of the proposed router, including the new Spillover Flow Control mechanism and the new microarchitectural techniques developed for the ring-based router. The correctness of the proposed design is formally verified in Section 4, while Section 5 describes and analyzes the various simulation experiments. Finally, Section 6 concludes the article.

2. RELATED WORK

2.1. Low-Cost NoC Routers

The architectures of BLESS [Moscibroda and Mutlu 2009], CHIPPER [Fallin et al. 2011], and MinBD [Fallin et al. 2012] revolve around a bufferless router design that employs deflection routing. Instead of storing blocked flits in a buffer, the deflection routing scheme forwards them to an adjacent node, even though the latter may not lie on the shortest path from source to destination. MinBD [Fallin et al. 2012] further enhances performance by attaching a small side buffer with minimal cost. The BPS [Gómez et al. 2008] and SCARAB [Hayenga et al. 2009] designs employ retransmission to realize bufferless routing. When contention occurs, the involved packets are dropped and retransmitted.

There have also been attempts to reduce the buffer size, rather than eliminate it altogether. Elastic buffering [Michelogiannakis et al. 2009] reduces the buffer space by exploiting latches in the interrouter links. ViChar [Nicolopoulos et al. 2006] dynamically adjusts the buffer size so that the available buffer space can be utilized more efficiently.

A recently introduced low-cost ring-based router [Kim 2009] reduces the overall cost of the router by employing simple ring primitives. The routing and switching logic components of the ring-based router are much simpler than in conventional routers.

The design of HiRD is currently a work in progress [Fallin et al. 2011], and, in its current incarnation, it employs a number of hierarchical rings working with deflection. Although the design provides injection and ejection guarantees to avoid deadlocks and

livelocks, it does not support virtual channels, and, hence, it does not address protocol-level deadlocks. This is a serious deficiency in designs aimed to be integrated within a CMP, since the interconnect is expected to support the cache coherence protocol. Without protocol-level deadlock guarantees, the interconnect may cause the whole system to halt. In contrast, the proposed TornadoNoC architecture inherently supports virtual channels and provides techniques for resolving both network- and protocol-level deadlocks, as well as livelocks, and the correctness of all these techniques is formally proven in this work.

In general, low-cost routers succeed in reducing the *cost* while offering competitive performance at low to medium injection rates only. Their *scalability* is typically limited.

2.2. Low-Latency NoC Routers

Scalability can be enhanced by adding more buffer space or by reducing the latency of the routers. Several researchers have targeted the latter.

There have been various approaches of using a preconfiguration network [Kumary et al. 2007; Jerger et al. 2008; Hayenga et al. 2009; Abousamra et al. 2012] to reduce the latency of routers. A setup flit goes through the preconfiguration network in advance of the actual packet in order to form a path within the main data network. Instead of a separate setup flit, a lookahead message can be employed for preconfiguring the next hop [Park et al. 2012]. Prediction within the router is another approach for single-cycle routing [Micheliogiannakis et al. 2007; Matsutani et al. 2009; Kim et al. 2006]. A packet's next destination is *predicted* and the packet is forwarded within a cycle. If the prediction turns out to be incorrect, the packet is killed and inevitably suffers a longer delay. Bufferless routers may also achieve single-cycle latency by preconfiguration [Hayenga et al. 2009] or by look-ahead techniques [Moscibroda and Mutlu 2009].

There have also been approaches to reduce the router pipeline stages by aggressively optimizing the critical path of routers [Mullins et al. 2004; Gratz et al. 2006]. These studies have achieved single-cycle router latencies, but the critical path delay of the router increased. Therefore, the maximum operating clock frequency decreased accordingly. The critical path delay of Mullins et al. [2004] becomes about 2.5 times longer [Mullins et al. 2006], while the clock frequency of Gratz et al. [2006] is reported at 366MHz using 130nm technology, which is relatively low.

Another approach to designing low-latency routers is to employ asynchronous interconnects [Gill et al. 2011; Horak et al. 2011; Kumar 2012; Gebhardt et al. 2011; Jain et al. 2010]. The router is implemented using synchronous logic, whereas the interconnects work asynchronously. This design philosophy is known as Globally Asynchronous Locally Synchronous (GALS). While asynchronous design has benefits over synchronous design in terms of power efficiency, this article focuses on synchronous logic, because synchronous designs are easier to test and compatible with commercial design automation tools.

2.3. Ring-Based Router Designs

The basic building block of a ring-based router is the *ring stop*, as illustrated in Figure 1. This setup has been given several different names in the literature, but, in this article, we refer to it as a *ring stop*.

A ring stop consists of buffers, a multiplexer, and a demultiplexer. When a packet arrives, it is stored in the intermediate buffer if its destination is the current stop. Otherwise, it is stored in the ring buffer and, subsequently, forwarded to the next stop. A packet can be injected into the ring only when the ring buffer is vacant. In other words, a packet in the ring buffer always has higher priority.

The work in this article exploits the fact that a ring-based architecture can achieve one cycle per hop without markedly affecting the critical path delay [Kim 2009]. Fur-

Fig. 1. A high-level illustration of a *ring stop*, the basic building block of ring-based router architectures.

thermore, a ring-based architecture does not incur additional latency for a setup flit (since a setup flit is not required). In general, it offers more sustainable performance—over a fairly wide range of injection rates—than the prediction and retransmission approaches.

A major drawback of ring-based interconnects is their limited scalability with the number of nodes. To combat this weakness, hierarchical rings [Holliday and Stumm 1994], hyperrings [Sibai 2008], torus rings [Chuang and Chao 1994], and hybridized ring/mesh architectures [Bourduas and Zilic 2007a, 2007b] have been proposed. The new architecture introduced in this article can be part of such hierarchical architectures. TornadoNoC can be used as one component in a hierarchical setup in order to offer even better scalability. For example, if a network consists of local networks connected through a global network, TornadoNoC can be employed for either the local, the global, or both networks. TornadoNoC can complement hierarchical approaches when even better scalability is required. If a monolithic (i.e., nonhierarchical) TornadoNoC is directly compared against hierarchical approaches, its benefit is low cost, as opposed to the better scalability of hierarchical approaches. A large system consisting of many processing elements is likely to be communication centric, which may result in excessive power consumption in the communication network. For a large system, the communication network should be not only scalable but also power efficient. Since hierarchical approaches require more resources than a monolithic network to accommodate multiple networks and their interfaces, the overhead is likely to be larger.

The work in Kim [2009] adopts the ring primitive as a basis for a router design for 2D mesh networks in an effort to reduce both area and power consumption. The new architecture proposed in this article adopts a similar topology and employs bidirectional 2D rings to ensure scalability and minimal hop counts among the various CMP nodes.

However, ring-based interconnects cannot completely satisfy all communication demands in future CMPs with tens of cores, because rings are inherently deficient in terms of raw throughput. When the injection rate is very low, the latencies achieved using a ring are significantly lower than a conventional multistage NoC router, but they sharply increase as the injection rate increases, as reported in Kim [2009].

This behavior is attributed to the fact that ring-based routers suffer from HoL blocking. In order to combat this problem, virtual channels are typically employed to alleviate the situation. As an additional benefit, VCs are also useful in preventing protocol-level deadlocks, if the upper-layer protocol (e.g., the cache-coherence protocol) offers no safeguards. One naive way to apply VCs to the ring-based router is to add more intermediate buffers and ring buffers in parallel (see Figure 1). However, this simplistic methodology increases the hardware cost significantly, which cancels out one of the primary merits of a ring-based router. Instead, we employ a novel Spillover Flow Control scheme to minimize the hardware cost required to support VCs. It will be demonstrated in this article that the throughput of the ring-based router is substantially improved by employing the new Spillover Flow Control technique in combination with VCs. Moreover, these improvements are achieved with only minimal hardware cost.

Fig. 2. A high-level block diagram of the proposed ring-based router microarchitecture. Black boxes marked with “In” indicate injection from the Network Interface Controller (NIC) and those with “Out” indicate ejection to the NIC. Note that multiple injection/ejection points are shown for clarity; in reality, *only one* flit can be injected/ejected per clock cycle.

Fig. 3. The topology of the proposed ring-based routers composing *TornadoNoC*. A box denotes a ring-based router and a filled triangle denotes a ring stop. Empty triangles indicate *omitted* ring stops in the routers on the edges of the mesh. The dotted feedback links are parts of the various rings.

3. THE TORNADONOC ARCHITECTURE

3.1. The Microarchitecture of the New Router

Figure 2 presents a high-level overview of the proposed ring-based architecture. The ring-based router employs a deterministic XY routing algorithm, which ensures *network-level* deadlock freedom. The topology of the proposed ring-based routers assumed in this work is shown in Figure 3 and it forms the basis of the *TornadoNoC* architecture. Due to lack of space, Figure 3 shows a 4×4 mesh, but larger meshes are

also employed in our experiments. A box indicates a router and a triangle indicates a ring stop. Without the dotted feedback links (lines), the topology is identical to a 2D mesh. The feedback links are added so as to form a ring for each column and row of the mesh. The dotted feedback links support the deflection-based Spillover Flow Control mechanism, which is explained in Section 3.2. There is one *unidirectional* ring for each column and each row, but each ring has *two stops per node*, so that a flit injected in any node can be forwarded to any one of the two ring directions (up/down, left/right). This feature enables efficient *bidirectionality* (in a physically unidirectional ring) and ensures that a flit can be delivered through the minimum Manhattan distance. Routers on the edges of the mesh do not need two ring stops, only one facing toward the inside (hence the empty triangles in Figure 3). This topology is different from a 2D mesh in the wrap-around links. As compared to a 2D torus, it has *two* ring stops per node, instead of one. In hierarchical rings, only a limited number of ring stops have an interrupting interface (for transfers between rings), but, in this topology, *all* ring stops have an interrupting interface. The work of Kim [2009] also adopts a ring-based router, but its topology is still a 2D mesh.

When a flit is injected from the NIC, it is immediately forwarded to the next node. This forwarding continues until the flit reaches either its destination node or a junction where it needs to transfer to a perpendicular ring (i.e., change travel dimension). If a flit requires a ring transfer, it is ejected from the ring and stored in the intermediate buffers in the Inter-Ring Interface (IRI). It takes a single cycle per hop when a flit travels within a ring, and two cycles when a flit transfers to another ring (to change from the X to the Y dimension). Since the ring-based router adopts fixed Dimension-Order Routing (DOR), ring transfers occur only from horizontal rings to vertical rings.

The proposed ring-based router supports VCs by employing multiple intermediate buffers in the IRI, as illustrated in Figure 2. There are two groups of buffers, one of which is for the east-bound ring stop and the other for the west. Without loss of generality, each group has three buffers to support three VCs. Any number of VCs can be supported, based on the number of buffers in the IRI. Flits ejected from horizontal rings to transfer to vertical rings are stored in different buffers, according to their assigned VC. Once popped from a VC buffer, a flit goes to the north, south, or NIC (ejection), depending on its destination.

In Figure 2, black boxes marked with “In” indicate injection from the NIC and those with “Out” denote ejection to the NIC. There are multiple injection/ejection points in a router, but *the NIC can inject/eject only one flit per cycle*. This restriction may incur additional delay in the NIC, because it needs to decide which ring stop a flit will be injected into or be ejected from. Conservatively, in our experiments later on, we assume that the NIC requires two additional clock cycles for its operation: one for injection and the other for ejection.

The appropriate VC is assigned to a flit upon injection and the assignment is *not* changed until the flit is finally ejected. Therefore, virtual channel allocation is not necessary within the proposed ring-based router. Because of the use of the Spillover Flow Control scheme, HoL blocking can be avoided without changing the assigned VC in-flight.

3.2. The Proposed Spillover Flow Control Mechanism

Existing ring protocols employ a *credit-based* flow control scheme orchestrated by dedicated control logic that sends credit information to the upstream node, so as to throttle the sending of flits when buffer space is not available [Ravindran and Stumm 1997; Kim 2009]. Instead, the proposed ring stop implements flow control by adding minimal logic to the input demultiplexer. The input demultiplexer forwards the incoming flit to the intermediate buffer (see Figure 1) *only* if its destination is the current node *and*

Fig. 4. An illustrative example of the proposed deflection-based Spillover Flow Control mechanism.

the intermediate buffer is “available.” It should be noted that both checks can be *fully parallelized and overlapped* in hardware.

The Spillover Flow Control mechanism works for each flit independently. Thus, each flit should carry its header information, including its destination node ID. This overhead is accounted for in the cost evaluation of Section 5.3. Since the number of bits representing the node ID grows *logarithmically* with the number of nodes, this overhead does not limit the overall scalability of the scheme. Regardless, the use of wormhole flow control could reduce this overhead. For instance, one may employ header and tail flits—as wormhole routers usually do—and allow the decision for Spillover to be made only by the header flit. Subsequent flits would then blindly follow the header flit until the tail flit releases the Spillover condition. Thus, the unit of operating granularity of a wormhole-based TornadoNoC would be the *packet*, instead of the *flit*. The detailed implementation of wormhole flow control is left as future work.

If the intermediate buffer is full, the flit is forwarded (spills over) to the next stop, even if its destination is the current node. Since the topology is a ring, the flit will, eventually, come back to the current node (after it completes one full cycle around the ring). If a source node keeps injecting flits into the ring while the intermediate buffer in the destination node is *not* draining at the same pace, then the ring buffers in the ring(s) where the source and destination nodes are located will become full. Hence, no more flits can be injected, because a new flit can only be injected when there is no flit in the local ring buffer. In fact, this is the technique used to stop a source from sending more flits when the destination node cannot keep up.

Figure 4 shows an example of the operation of the deflection-based Spillover Flow Control mechanism. Suppose that there are four ring stops in a ring. Ring stop 0 (RS0) currently holds flit B, whose destination is RS3. RS1 has flit A, which is going to RS2, but the intermediate buffer of RS2 is full. If a credit-based flow control mechanism is employed, this is a typical HoL blocking situation. Since flit A cannot go to RS2, flit B cannot go to RS1. On the other hand, if the proposed Spillover Flow Control scheme is applied, flit A can still move. Flit A goes to the ring buffer of RS2, instead of the intermediate buffer, because the intermediate buffer is full. In the next cycle, flit A goes to RS3. Therefore, flit A does not block flit B from going to RS3. Since the topology is a ring, flit A eventually comes back to RS2. If its intermediate buffer has become available, flit A can be ejected at RS2.

Credit-based flow control requires the ring buffer depth to be at least two in order to hide the latency of the credit exchange. However, the deflection-based Spillover Flow Control technique needs only one-deep ring buffers, since it is guaranteed that the ring buffer of the next stop is always available. This feature also eliminates the possibility of deadlock, as will be discussed in Section 4.

There still exists, however, a chance of message-dependent protocol-level deadlock. A protocol-level deadlock occurs when a node is waiting for a certain type of message, but its buffer is filled with other types of messages. This serious issue is addressed in this work by employing VCs and an Inflation Mechanism, which will be explained in Section 3.4.

The Spillover Flow Control technique may cause (1) reversing of the order of flits and (2) network livelocks. If a flit cannot be stored in the intermediate buffer of its destination node—because the buffer happens to be full at a certain moment—it is forwarded to the next node until it finally returns to the destination node again. However, if the intermediate buffer becomes available again before the flit wraps around the ring, a following flit with the same destination may very well use the intermediate buffer and be ejected from the local router. If the two flits originate from the same source, their order will then be reversed. Furthermore, if this situation occurs repeatedly, the spilled flit may *never* be able to reach its destination, which gives rise to a livelock situation. To combat these two problematic artifacts, we hereby propose a *sequence numbering mechanism* that addresses both problems (flit ordering and livelocks) at the same time. A **formal proof** of the correctness of this mechanism will be presented in Section 4.

3.3. Using a Sequence Numbering Mechanism to Preserve Flit Order and Prevent Livelocks

This subsection presents a sequence numbering mechanism. Its purpose is to preserve the order of *flits* for each source and destination pair. The order of received flits at a destination node should be the same as the sending order at the source node. As long as the flit-level order is maintained between source-destination pairs, the packet-level order is also maintained. For example, suppose that a source node sends packets A, B, and C—in this order—and packet B is split into flits B0, B1, and B2. If the flit-level order (i.e., A, B0, B1, B2, C) is maintained, then the packet-level order (i.e., A, B, C) is also preserved.

The sequence numbering mechanism requires each ring stop to maintain three fields: *Spillover Flag*, *Head*, and *Tail*. Additionally, two fields should be added to the header of each flit: a *Spillover Bit* and a *Sequence Number*. The *Spillover Flag* and *Spillover Bit* fields are single bit, while the sizes of the remaining three fields (*Head*, *Tail*, and *Sequence Number*) will be analyzed later on in this subsection.

Initially, all the fields are zero. When a flit arrives (whose destination is the local ring stop) but cannot be ejected due to a full intermediate buffer, the flit should be spilled. Upon spillover, the *Spillover Flag* and *Spillover Bit* are turned on. The *Spillover Flag* indicates that there are spilled flits. The *Spillover Bit* in the header of the flit indicates that this flit has been spilled. The *Sequence Number* field is assigned the current *Head* value and then *Head* is increased by one. The *Sequence Number* indicates the order of the spilled flits. While the intermediate buffer is full, all the incoming flits are spilled and assigned a sequence number in increasing order. When the intermediate buffer becomes available, the ring stop starts accepting flits again. While the *Spillover Flag* is on—which implies that there are spilled flits—a node only accepts the flit whose *Sequence Number* value matches the node's *Tail* value. Whenever a flit is accepted, *Tail* is increased by one. When the *Tail* value reaches (equals) the *Head* value, the *Spillover Flag* is cleared to zero. If the *Spillover Flag* is zero, a node does not check the *Sequence Number* field of each flit. While a ring stop is accepting spilled flits, a new flit that has not experienced spillover may arrive. The flit should be spilled, even though the intermediate buffer may be available, in order to preserve the flit order. This mechanism is summarized in pseudo-code form in Algorithm 1.

The proposed sequence numbering mechanism is conservative, since the *Spillover Flag* is maintained for each VC without considering the source node. If we maintain a flag for every source node, the condition would be relaxed and we could achieve even better performance and scalability. However, such an improved scheme would incur hardware overhead proportional to the number of ring stops in a ring. We leave further investigation on the tradeoff between the performance improvement and the hardware overhead as future work.

ALGORITHM 1: The Sequence Numbering Mechanism's pseudo-code

```

if Spillover Flag is OFF & Buffer is NOT available then
  Spill the flit
  Spillover Flag  $\leftarrow$  ON
  Spillover Bit  $\leftarrow$  ON
  Sequence Number  $\leftarrow$  Head
  Head  $\leftarrow$  Head + 1
end if
if Spillover Flag is ON & Spillover Bit is OFF then
  Spill the flit
  Spillover Bit  $\leftarrow$  ON
  Sequence Number  $\leftarrow$  Head
  Head  $\leftarrow$  Head + 1
end if
if Spillover Flag is ON & Spillover Bit is ON & Buffer is NOT available then
  Spill the flit
end if
if Spillover Flag is ON & Spillover Bit is ON & Buffer is available & Sequence Number  $\neq$ 
Tail then
  Spill the flit
end if
if Spillover Flag is ON & Spillover Bit is ON & Buffer is available & Sequence Number =
Tail then
  Accept the flit
  Spillover Bit  $\leftarrow$  OFF
  Tail  $\leftarrow$  Tail + 1
  if Tail = Head then
    Spillover Flag  $\leftarrow$  OFF
  end if
end if

```

The maximum possible value of the *Sequence Number*, *Head*, and *Tail* fields is equal to $2 \times M$ in a horizontal ring and $2 \times V \times M + 2 \times M$ in a vertical ring, where M denotes the number of ring stops in a ring, and V denotes the number of VCs. If any of these fields reaches the maximum value, it wraps around to zero. The *Head* field indicates how many flits have been spilled and *Tail* indicates how many spilled flits have been accepted. Therefore, the difference between *Head* and *Tail* indicates the number of spilled flits remaining in the ring. The maximum number of flits remaining in the ring is M . However, the Inflation Mechanism to be described in Section 3.4 allows one additional flit (waiting to be injected) per ring stop to be given a sequence number. Therefore, the maximum possible number of flits that are assigned a sequence number is $2 \times M$ in a horizontal ring. In vertical rings, a flit can be injected from the intermediate buffers. Since the number of intermediate buffers is $2 \times V$, the maximum number of flits that can be given a sequence number in a vertical ring is $2 \times V \times M + 2 \times M$.

In the case of an 8×8 mesh topology with 3 VCs, the number of additional bits in the flit header is 6 bits in a horizontal ring and 8 bits in a vertical ring. This overhead is due to the *Spillover Bit* and the *Sequence Number*. Clearly, the overhead—in terms of extra bits—is negligible, considering the number of wires dedicated to the payload, which is usually assumed to be 128 bits (or even 256 bits in recent designs [Kumar et al. 2007; Kumary et al. 2007; Hayenga et al. 2009; Das et al. 2009; Fallin et al. 2011]).

Fig. 5. Ring stops with one extra flit slot in the ring buffer to support the inflation mechanism.

3.4. Employing an Inflation Mechanism to Prevent Protocol-Level Deadlocks and Starvation

The Inflation Mechanism is a lightweight augmentation of the ring, which is introduced to address (1) *protocol-level deadlocks* and (2) *starvation*. Figure 5 shows the modified ring stop that supports inflation. Under normal operation, an incoming flit is forwarded to slot 0 and leaves the ring stop in the next cycle. When the ring stop is inflated (upon detection of a possible protocol-level deadlock), incoming flits go to slot 0, then move to slot 1 in the next cycle and, finally, leave the ring stop in the following cycle. Thus, during ring inflation, an extra slot is made available. The inflated ring stop causes incoming flits to take one extra cycle to pass through the ring stop.

If a ring stop is unable to inject a flit for a period longer than a predefined threshold, the ring stop is inflated. The value of the threshold is an implementation decision. In our experiments, we set the threshold to 100 cycles, but we never observed a single case of protocol-level deadlock, or starvation under *real* workloads. This means that the threshold is high enough to avoid false-positive triggers.

Upon inflation, an extra slot is immediately made available. However, a flit is injected only if it is *ejectable*. A flit is considered *ejectable* in a horizontal ring if the intermediate buffer in the IRI of the receiving node is available. In a vertical ring, a flit is *ejectable* if the NIC in the receiving node can process the flit (i.e., the NIC is not blocked while waiting for other flits). To check if the flit is *ejectable*, an inquiry message is sent through the extra slot. The sequence number of the flit is also checked to ensure that sequence numbering (maintained by the sequence numbering mechanism) is not violated. The receiving node responds to the sender through the extra slot. If the inquiring flit is, indeed, *ejectable*, the receiver reserves one flit slot in the buffer for the sender, and the sender injects a flit into the extra slot when the extra slot rotates back to the sender. The extra slot arrives at the destination with the actual flit and comes back to the sender empty, because the extra slot is allocated exclusively to the sender. Once the slot returns to the sender empty, the ring stop is deflated; that is, the extra slot is removed from the ring. If the flit is not *ejectable*, the sender ring stop is deflated temporarily, and the sender tries again after the predefined threshold. However, the flit is still given a sequence number, even if it has not been deemed *ejectable* yet. In the next attempt, the inquiry message to the receiving node contains the sequence number and the inquiring flit is treated just like a spilled flit. This ensures that (1) the flit order is not violated by the Inflation Mechanism and (2) every ring stop is given a fair chance to inject.

The details on how this simple mechanism breaks protocol-level deadlocks and prevents starvation will be discussed in Section 4, which **formally proves** the correctness of this scheme. Moreover, all microarchitectural mechanisms described in this section will be included in the hardware synthesis evaluation of Section 5.3.

Note that although the Inflation Mechanism can prevent starvation by preventing a downstream node from ceaselessly injecting packets, it cannot guarantee *fairness*. The Inflation Mechanism can be augmented with additional logic/safeguards to address fairness issues. For instance, one possible way to address fairness is by allowing the

threshold of the timeout to be adjusted according to a router's spatial location. A lower threshold could be assigned to center nodes, so that they can trigger the Inflation Mechanism more often than others. Of course, the determination of appropriate threshold values and their relation to a node's location in the network (within the context of fairness) would need to be investigated in detail. The augmentations to the Inflation Mechanism to ensure fairness are left as future work.

3.5. Scalability of the Proposed Router

The highly scalable trait of the proposed router is attributed to four aspects:

Spillover: Conceptually, the ring buffers in a ring can be viewed as an extension of the intermediate buffer for the spilled flits. Since the number of ring buffers increases with the number of nodes in the network, increasing the number of nodes has the same effect as increasing the buffer size.

The existing deflection routing schemes [Moscibroda and Mutlu 2009; Fallin et al. 2011] could also potentially benefit from this attribute. However, their deflection routing policy forwards misrouted flits anywhere in the network. In contrast, the deflection-based Spillover Flow Control mechanism in the proposed router keeps misrouted flits within the ring(s). Therefore, the worst-case latency of the spilled flits is limited, even in large networks, which contributes to significantly better scalability than existing router architectures employing deflection routing.

Adaptivity: The proposed router is more resilient to nonuniform traffic—where traffic is skewed toward a certain node, or VC—than conventional credit-based routers. When credit-based flow control is employed, the blocked nodes hinder neighboring nodes from sending flits. Credit-based routers exhibit the best performance when the traffic is uniformly distributed. In contrast, hot-spot nodes, or VCs, in the proposed router do not block other nodes from sending flits.

Contention Probability: Head-of-Line (HoL) blocking is incurred by blocked flits. A flit may still be blocked—even if its destination buffer is available—when contention occurs between multiple flits going to the same destination. One way to reduce the contention probability is to split the crossbar [Kim et al. 2005, 2006]. By splitting the crossbar into row and column modules, the RoCo router [Kim et al. 2006] achieves higher throughput than conventional routers. The proposed ring-based router benefits from the same effect, because the rings are separated into horizontal and vertical dimensions.

HoL Blocking Avoidance: The ring-based architecture and the Spillover Flow Control of TornadoNoC enable the avoidance of HoL blocking scenarios that cannot be addressed by conventional routers. For example, let us assume that three packets *belonging to the same message class* (i.e., the same VC) arrive from the west and are headed to the south, north, and east, respectively. If the downstream router in the south could not receive any packets (e.g., no credits), the first flit—headed to the south—would block the following two flits. This is a typical HoL blocking scenario in conventional routers. Instead, in TornadoNoC, the first and second flits would be ejected from the ring to change their direction (to the south and north, respectively), as long as the intermediate buffer were available. Therefore, the third flit would move to the east without any blocking. Even if the intermediate buffer were not available, the first two flits would be spilled, and the third one would still be able to move to the east toward its destination.

4. PROOF OF CORRECTNESS

This section formally proves that the proposed techniques in Section 3 are correct in their operation.

4.1. Flit Order Preservation

Since the intermediate buffers in the IRI are FIFOs, the order of flits within the IRI is always maintained. The sequence numbering mechanism works for each VC independently, because only the flit order within each VC must be preserved. Therefore, this subsection will focus on proving the order preservation of flits belonging to the same message class (VC) within a single ring.

Let $A(f)$ denote the time a flit f arrives at its destination node for the first time and $Q(f)$ denote the flit's sequence number. Since a flit may be spilled, it may arrive at its destination node multiple times before being ejected. As such, $A(f)$ denotes the time of the first arrival. The first arrival of an inquiry message through the extra slot is also treated as the first arrival of the inquiring flit pending to be injected in the originating ring stop. Note that the inquiry message traveling through the Inflation Mechanism cannot reach the destination earlier than the flits that left prior to the inquiry. Therefore, if flits are ejected in the order of $A(f)$, then flit order is preserved by the nature of the ring's operation.

LEMMA 1. *If there is no spillover, flits are ejected in the order of $A(f)$.*

PROOF. This is obvious. If there is no spillover, which implies that the buffer was never unavailable, flits are ejected in the incoming order. Since the order cannot be violated in a ring, the ejection order is unchanged. $\square$

THEOREM 1. *If all incoming flits are given a sequence number, such that $Q(f_p) < Q(f_q)$ iff $A(f_p) < A(f_q)$, then flits are ejected in the order of the first arrival time.*

PROOF. If the buffer is not available, all the incoming flits are given a sequence number in an increasing order. The sequence number is given only once, upon first arrival, and it is not changed until the flit is ejected. Therefore, the property that $Q(f_p) < Q(f_q)$ iff $A(f_p) < A(f_q)$ is always preserved. At a certain moment, if the buffer becomes available, the spilled flits are ejected according to their sequence number. Since the sequence number order is the same as the order of the first arrival times, the ejection order is also the same as the order of the first arrival times. $\square$

The spillover flag (see Section 3.3) is turned on and off depending on buffer availability. When the flag is off, the flit order is preserved, as stated in Lemma 1. When the flag is on, all the incoming flits are given a sequence number. Even though the buffer may become available at any time, if there still remain spilled flits rotating around the ring, any new incoming flits are also spilled and given an increasing sequence number. Remember that the spillover flag is cleared only when all spilled flits are drained. Thus, while the flag is on, the order is preserved according to Theorem 1.

4.2. Formal Verification of Deadlock and Livelock Freedom

This subsection provides a formal proof that the proposed router is free from network- and protocol-level deadlocks, livelocks, and starvation.

In general, there are four necessary conditions for a *network-level* deadlock to occur [Coffman et al. 1971]: (1) resources are exclusively claimed, (2) nodes hold already allocated resources while waiting for additional resources, (3) resources cannot be forcefully released, and (4) a cyclic dependency is formed. Conventional wormhole routers actually satisfy the first three conditions, and they usually break the last condition by employing deadlock-free routing algorithms (e.g., deterministic XY routing). Thus, network deadlocks are avoided simply by breaking the last of the four aforementioned conditions. Within a ring, the proposed router satisfies conditions (1), (3), and (4), but it breaks the *second* condition through the use of the Spillover Flow Control mechanism: A ring stop never holds, or waits, for resources. From a higher-level perspective, the

second condition still holds *outside* the ring. When a flit is waiting to be injected, it holds resources while waiting for other resources. At this higher abstraction level, the *last condition* is broken instead, by employing a deterministic XY routing algorithm.

However, when relationships among *protocol-level* messages are also considered, the additional dependencies at the destination node can cause cyclic dependencies among messages [Song and Pinkston 2003]; these are known as *message-dependent, protocol-level deadlocks*. The proposed router breaks the cyclic dependency by using the Inflation Mechanism described in Section 3.4.

As an illustrative example, let us consider a single ring that carries two classes of messages: one class is for *requests* and the other is for *responses*. The *requests* are dependent on the *responses*. The formal definition of a message dependency will be given shortly, but for now, it suffices to say that, in some occasions, the NIC cannot process a *request* until it receives a *response*. While a NIC is blocked waiting for a *response*, if other ring stops keep sending *requests* to the same node, the ring buffers eventually become full with *request* flits. Even if a node wants to inject a *response*, it cannot do so. This is a typical protocol-level deadlock situation. Since *responses* are blocked at the source node by *requests*, and *requests* are blocked by *responses* at the destination node, a cyclic dependency is formed. By using the Inflation Mechanism of Section 3.4, the dependency at the source node is broken. If the source node cannot inject a *response* for a period longer than a threshold, it is inflated. It can thereby send a *response* as long as the *response* flit is *ejectable* at the destination. In this scenario, the *response* flit is certainly ejectable, because the destination NIC is actually waiting for it. Therefore, a *response* can be sent and the protocol-level deadlock is broken. The proof provided to follow is a generalization of this observation.

Let $S(f)$ be the source node of flit f , $T(f)$ be a transit node where the flit transfers to another ring, and $D(f)$ be the destination of flit f . Depending on the locations of $S(f)$ and $D(f)$, $T(f)$ may not exist. In addition, $C(f)$ denotes the message class of flit f .

LEMMA 2. *If both the intermediate buffer of $T(f)$ and the NIC of $D(f)$ are not blocked permanently, f can reach $D(f)$ within a finite amount of time.*

PROOF. If f is rotating in the vertical ring, where $D(f)$ is located, f is ejected within a finite amount of time, because flits are ejected in the order of their first arrival and the first arrival occurs within a finite amount of time, or f can be ejected immediately upon arrival without spillover. If f is in the intermediate buffer of $T(f)$, f can reach $D(f)$ within a finite amount of time by the Inflation Mechanism, because f becomes *ejectable* at $D(f)$ within a finite amount of time, or f can reach $D(f)$ through a regular path. In a similar vein, f rotating in the horizontal ring, or waiting in the ring stop to be injected, can reach $T(f)$ and, subsequently, $D(f)$ within a finite amount of time. $\square$

If a flit can reach its destination within a finite amount of time, the flit is called *deliverable*. Note that Lemma 2 covers the flits pending at the source node to be injected. Therefore, if a flit is *deliverable*, the flit is free from starvation (whereby a flit is permanently unable to be injected), and it is also free from livelock (whereby a flit is permanently unable to be ejected). Assuming buffer space is available in the IRI of $T(f)$ and it is not being blocked by the NIC of $D(f)$, this lemma proves that any given f is deliverable. This assumption is proven to be true for all the flits in the network by the following theorems.

THEOREM 2. *For $\forall f$, where $C(f)$ is a terminating class, f is deliverable.*

PROOF. Let us introduce a relation $m_1 < m_2$, if m_1 depends on m_2 . The formal definition is: " $m_1 < m_2$ iff m_2 can be generated by a node receiving m_1 for some data transaction" [Song and Pinkston 2003]. When we draw a dependency graph among

message classes, it must be a totally, or partially, ordered graph, because there must be no cyclic dependency [Song and Pinkston 2003]. Then, there exists at least one message class that does not depend on any other message classes. Such a class is called a *terminating class* [Song and Pinkston 2003].

If flit f belongs to one of the terminating classes, it is never blocked at the NIC of $D(f)$ by other classes. In other words, f is always *ejectable* at $D(f)$. All the pending flits in the intermediate buffer of $T(f)$ can reach their destination within a finite amount of time, because they are always ejectable at their destination. Since the buffer stores only flits of the same class, the buffer of $T(f)$ is not permanently blocked. Therefore, f is *deliverable*, according to Lemma 2. $\square$

If all the flits belonging to a certain class are *deliverable*, the class is called a *deliverable class*. The aforementioned terminating class is a deliverable class.

THEOREM 3. *Given a set of message classes $\mathbb{G}$, where for $\forall C_d \in \mathbb{G}, C(f) < C_d$, if $\forall C_d$ is a deliverable class, f is deliverable.*

PROOF. $C(f)$ may be dependent on multiple message classes. If all the classes that $C(f)$ depends on are deliverable classes, all the flits belonging to those classes are deliverable. Even though the NIC of $D(f)$ may be blocked waiting for some flits at a certain moment in time, all of those flits can reach $D(f)$ within a finite amount of time. Therefore, the NIC of $D(f)$ can process f within a finite amount of time and, thus, f is deliverable. $\square$

The relation $<$ is transitive, which means if $m_1 < m_2$ and $m_2 < m_3$, then, $m_1 < m_3$. Since the dependency graph is totally, or partially, ordered, any given class ultimately depends on a set of terminating classes. The set $\mathbb{G}$ in Theorem 3 can always be expressed by a set of terminating classes. Therefore, all the message classes are deliverable classes, which means that all the flits are deliverable and, thus, free from deadlocks, livelocks, and starvation.

5. EXPERIMENTAL EVALUATION

5.1. Simulation Framework

Our evaluation approach is double-faceted, utilizing (1) synthetic traffic patterns and (2) real application workloads running in an execution-driven, full-system simulation environment. We employ Wind River's Simics simulator [Wind River Systems 2012], extended with the Wisconsin Multifacet GEMS simulator [Martin et al. 2005] and GARNET [Agarwal et al. 2009], a cycle-accurate NoC simulator. Without loss of generality, all simulations assume *deterministic XY routing*.

The full-system simulation setup running real application workloads is initially used to evaluate near-future multicore systems containing 64 processing cores. A 64-core tiled CMP system (in an 8×8 mesh) is simulated in order to assess the impact of the proposed router on overall system performance. The simulation parameters are given in Table I. The executed applications are part of the PARSEC benchmark suite [Bienia 2011]. PARSEC is a benchmark suite that contains multithreaded workloads from various emerging applications. All applications use 128 threads.

Since one of the fundamental objectives of this work is to demonstrate the scalability attributes of TornadoNoC, experiments with many-core systems (containing up to 1,024 cores) are also conducted. However, existing full-system, execution-driven simulators become prohibitively slow as the number of simulated processing cores increases beyond 100. This crippling limitation renders full-system simulations of many-core systems practically impossible to perform. Hence, our scalability evaluation to hundreds of cores focuses only on the on-chip network, but it employs realistic synthetic

Table I. Simulated System Parameters

System parameters	
Processors	×86 Pentium 4
L1 cache	32KB, 4-way, 64B line
L2 cache (shared)	16MB, 16-way, 128B line
Operating system	Linux Fedora
Main memory	2GB SDRAM
Network parameters	
Topology	8×8 mesh
Flit width	128 bits
Virtual networks	3
Buffer size	4 flits per VC
Intermediate buffer	4 flits per VC

traffic patterns that are derived from the application behavior observed in full-system simulation environments of smaller-scale systems. The details of the construction of these realistic synthetic traffic traces will be explained in the following subsection. For synthetic simulations, GARNET is utilized in a “network-only” mode, with Simics and GEMS detached.

The GARNET simulator cycle accurately models the microarchitecture of the routers. All designs under investigation in this article were implemented within GARNET. The MOESI-CMP-directory is used as a cache coherence protocol. It requires at least three *virtual networks* to prevent protocol-level deadlocks. The term *virtual network* refers to a group of VCs (or even a single VC) dedicated to a particular message class of the cache coherence protocol.

The proposed ring-based router (TornadoNoC) has one VC per virtual network (i.e., three VCs in total). The depth of its ring buffer is 1 and that of the intermediate buffers is 5. Note that the ring buffer of each node includes the extra slot shown in Figure 5 to support the Inflation Mechanism. However, under normal operation, the extra slot is not used. In all simulations, the Inflation Mechanism threshold value was set to 100 cycles. It is worth noting that under *real* application workloads, there was never a case of protocol-level deadlock, or starvation; that is, *the Inflation Mechanism was never triggered*.

Moreover, we conservatively assume that two additional cycles are required by the NIC of the ring-based routers compared to the NIC of the other routers under test. This additional delay is due to the multiple points of injection/ejection in each router, as explained in Section 3.1. Specifically, one extra cycle is needed when a packet is injected, because the NIC needs to decide which ring the packet should be injected into. The other extra cycle is needed when a packet is ejected, because the NIC needs to decide among four ring stops at any given time.

The proposed router design (TornadoNoC) is compared to four other architectures:

Full-Size Ideal Single-Cycle Router (SINGLE): This router is representative of all the previous work on low-latency designs mentioned in Section 2.2, which achieve single-cycle router latencies. It may be considered as a design with preconfiguration [Kumary et al. 2007; Jerger et al. 2008; Hayenga et al. 2009; Abousamra et al. 2012] *without any preconfiguration delay*, or as a prediction router [Michelogiannakis et al. 2007; Matsutani et al. 2009; Kim et al. 2006] *without any mispredictions*, or as a highly optimized router [Mullins et al. 2004; Gratz et al. 2006] *without increasing the critical path delay*. However, the SINGLE router architecture still suffers from contention/conflicts and queuing delay.

This router assumes abundant resources. It has 4 VCs per message class (i.e., 12 VCs in total) and each VC buffer is capable of storing 5 flits. The number of flits of the largest packet is 5 in our experiments. We do not consider this router configuration to be practical, but we include it as a reference point. It will be demonstrated shortly that the TornadoNoC router offers better scalability, even when compared to this impractical (unrealistic) router.

Ring-Based Router with Credit-Based Flow Control (RING) [Kim 2009]: This ring-based router is derived from Kim [2009] and it employs a credit-based flow control mechanism. One simple way to enhance the scalability of the ring-based router is to add more buffers in parallel to support VCs. Hence, in order to support VCs, we include one ring buffer per VC in the ring stops and one intermediate buffer per VC in the IRI. In our experiments, the depth of each ring buffer is 2 and the depth of each intermediate buffer is 4. The router has one VC per virtual network (i.e., 3 VCs in total). The depth of the intermediate buffer is 4, as opposed to 5, because this router already has the unfair advantage of including *three parallel* VC buffers (each of depth 2) in the ring stops (to support VCs). Instead, the proposed TornadoNoC router only has one buffer in the ring stops, while it still supports VCs.

Buffered Deflection Router (BUF-MinBD): This deflection router is derived from MinBD [Fallin et al. 2012]. One way to improve the scalability of a bufferless deflection router [Moscibroda and Mutlu 2009] is to add buffers. The tradeoff between the overhead of additional buffers and the improved performance is discussed in Moscibroda and Mutlu [2009]. We will demonstrate that the TornadoNoC router offers better scalability with less overhead than this router. The number of employed VCs is one per virtual network (i.e., 3 VCs in total) and the depth of each buffer is 4 flits. Additionally, the BUF-MinBD architecture employs a side buffer, which, essentially, provides each VC with an additional buffer slot. Hence, each VC has the equivalent buffer space of 5 flits (four regular slots plus one side buffer slot), which sets up a fair comparison—in terms of buffer space—with the other designs.

Conventional Three-Stage Router (CANONICAL): This is the *practical* incarnation of the aforementioned SINGLE router design. Rather than enjoying the unfeasibly large buffer space of SINGLE, the CANONICAL router only has three 5-flit-deep VCs in order to be comparable to the other router designs explored in this work. Being a three-stage router, the CANONICAL design has a latency of four cycles per hop (three cycles for the intrarouter pipeline stages, plus one cycle for interrouter link traversal). This router architecture is a practical reference point for the *hardware cost* evaluation of Section 5.3. It is not used for performance evaluation.

The rest of this section is organized as follows. The simulation experiments of Section 5.2.1 will help demonstrate that the proposed ring-based router architecture exhibits real-workload performance (up to 64 cores) that is nearly identical to various state-of-the-art router designs. Section 5.2.2 evaluates the scalability of TornadoNoC. In larger-scale many-core systems (with more than 64 cores), the proposed design is demonstrated to sustain markedly better scalability to *hundreds of cores*, as compared to existing designs. The significant gain comes from the fact that the TornadoNoC design offers such highly competitive performance while incurring *significantly lower* area/power overhead, as will be shown in Section 5.3.

5.2. Performance Evaluation

5.2.1. Evaluation Using Real Applications. The performance of the various routers is evaluated under real application workloads. As previously mentioned, we employ benchmarks from the PARSEC benchmark suite [Bienia 2011] running on the GEMS [Martin et al. 2005] simulator (integrated with Simics and GARNET in a full-system simulation environment). Figure 6 summarizes the evaluation results.

Fig. 6. Performance results using an execution-driven, full-system simulator running real applications from the PARSEC benchmark suite [Bienia 2011] on a 64-core CMP. The results are normalized to the SINGLE router.

Figure 6(a) compares the average *network* latency. We can see that the latency of the TornadoNoC router is very competitive when compared to the other scalability-enhanced low-cost routers. Specifically, the latency of TornadoNoC is higher than BUF-MinBD by 1.97%, on average. Compared with RING, the latency is reduced by 0.86%, on average. Obviously, these numbers are so close that they fall within simulation noise—the latency performance is practically indistinguishable between these three architectures. Note that, in our experiments, we conservatively assume that *two extra clock cycles* are required by the NIC in the TornadoNoC and RING routers. Compared with SINGLE, the latency of TornadoNoC is higher by 7.86%, on average. While this may seem considerable, it is important to remember that the SINGLE design is simply used as an idealistic reference point in these simulations, since, in practice, it is infeasible to implement (see Section 5.1).

Network latency alone is not enough to yield insight as to overall *system* performance, since the interconnect is only one component within the system. Thus, it is crucial to also observe the execution times of the running applications. Figure 6(b) compares the total execution times of the various benchmark applications. The execution times reported here are those of the so-called Regions Of Interest (ROI), as identified in the PARSEC benchmarks. The ROI of each benchmark starts right after the initialization of the input data and ends when the computation is complete. This implies that the data cache of the system is warmed up, because the input data has already been initialized at the moment the ROI begins. The instruction cache is not warmed up, because the computation does not begin until after the ROI begins. As can be seen in Figure 6(b), TornadoNoC achieves almost identical total execution time as the other state-of-the-art designs. The execution time of TornadoNoC is longer than SINGLE and BUF-MinBD by 2.51% and 0.03%, respectively. Compared with RING, the execution time is reduced by 2.17%. Once again, such small differences in the results indicate that all designs are

Table II. Distribution of Network Nodes with Respect to the Percentage of the Total Network Traffic Received
 For example, one node receives 15% to 20% of all generated traffic in the network when the *blackscholes* application is running, while six nodes receive 0.1% to 5% of the total traffic. *Note: Each row of the table adds up to 64 cores, which is the size of the CMP.*

Benchmark	Percentage of total generated network traffic received											Traffic pattern
	0 −0.1	0.1 −5	5 −10	10 −15	15 −20	20 −25	25 −30	30 −35	35 −40	40 −45	45 −50	
<i>blackscholes</i>	50	6	4	2	1		1					C
<i>ferret</i>	57	3	1	1				1			1	A
<i>freqmine</i>	60		2						1		1	A
<i>swaptions</i>	59		2	1			1				1	A
<i>fluidanimate</i>	53	4	3	1	3							B
<i>vips</i>	51	6	3	1	1	2						B
<i>x264</i>	54	5		2	2				1			C

extremely close in terms of performance and the absolute values reported here could be considered to fall within simulation noise. Hence, TornadoNoC offers essentially the same real-world performance as current state-of-the-art high-performance designs, and it closely approaches the performance of the reference (yet impractical) SINGLE router architecture.

However, the following section (Section 5.3) will demonstrate that TornadoNoC incurs substantially lower area/power cost than all the other designs under evaluation, thus providing a significantly more efficient design alternative. TornadoNoC's *performance-cost efficiency* is far superior, as illustrated in Figure 14, the details of which will be explained in Section 5.3.

5.2.2. Evaluation of Scalability to Hundreds of Cores. To evaluate scalability to the many-core domain, we increase the number of nodes in the network beyond 64. Due to the aforementioned, prohibitively long full-system simulation times for such large-scale CMPs, we resort to the use of synthetic traffic patterns. However, we aim to make the evaluation as authentic as possible by generating network traffic traces that are realistic and derived from detailed profiling of *real* multithreaded application behavior.

Using the full-system simulation setup employed in the previous subsection, we perform a detailed behavioral analysis of the PARSEC benchmarks, in terms of the network traffic they generate in the CMP.

One important attribute is how traffic is distributed to the various nodes of the CMP. Table II shows the distribution of the 64 network nodes with respect to the percentage of the total network traffic received. For example, when *freqmine* is running, one node receives 45% to 50% of all generated traffic in the network, while 60 (of the 64) nodes receive only 0% to 0.1% of the total traffic.

It is obvious that most of the network traffic is headed to a small number of nodes, which creates *hot spots*. This is a very important conclusion, since it appears that real application workloads do not generate uniform network traffic but, instead, tend to generate irregular, hot-spot-like traffic. The highly irregular traffic patterns produced by real applications have also been observed in Soteriou et al. [2006] and Bahn and Bagherzadeh [2008]. In addition, spatial traffic locality—which is often observed in large networks [Greenfield et al. 2007]—is accounted for in the simulations performed in this article. Specifically, the traffic sent to hot-spot nodes is more likely to originate from nearby nodes than distant nodes, thus ensuring the presence of adequate spatial traffic locality.

A critical question raised at this point is how the observed profiling behavior of a 64-core system would scale to many-core systems with hundreds of cores. In other words,

Fig. 7. Cumulative distribution functions of the network nodes with respect to the percentage of the total network traffic received, under the various PARSEC benchmark applications. The results for three different system sizes are shown (64, 32, and 16 processing cores). The more rapid increase in the values of the CDF curves as we increase the system size from 16 to 64 cores indicates increasing skewness (hot-spot-like nature) in the traffic; that is, the traffic is more heavily skewed toward a smaller number of nodes.

is it reasonable to use synthetic traffic patterns modeled using applications running on a 64-core system to evaluate a 1,024-core system? To investigate how traffic skewness (i.e., the hot-spot severity) varies across system sizes, we gather the same profiling information as depicted in Table II for systems with 32 and 16 cores (8×4 and 4×4 meshes, respectively). The goal of this investigation is to show how the traffic behavior of multithreaded applications changes in multicore systems of increasing sizes, up to 64 cores (the practical maximum size that can be simulated in a full-system evaluation framework). Figure 7 shows the results of this exploration through the use of the Cumulative Distribution Function (CDF) of the network nodes with respect to the percentage of the total network traffic received. Hence, Figure 7(a) is a graphical representation of the 64-core profile of Table II. Similarly, Figures 7(b) and (c) show the corresponding CDFs for 32 and 16 nodes, respectively. One may observe that, regardless of the network size, the traffic is heavily skewed toward a small number of nodes, and, as the number of cores increases from 16 to 64, the skewness increases. The increasing skewness is revealed by the more rapid increase in the values of the CDF curves as we increase the system size. Thus, the hot-spot-like nature of the network traffic observed in smaller systems tends to become more severe as the system size increases.

This observation justifies the use of skewed (hot-spot) synthetic traffic patterns to evaluate the scalability of various router architectures. From the profiling information of Table II, we identify three distinct hot-spot traffic patterns to be used in our subsequent simulation experiments:

- Pattern A:** 10% of nodes receive 100 times more network traffic than the others. This pattern is representative of the behavior observed in *ferret*, *freqmine*, and *swaptions*. In this pattern, a very small number of nodes receive most of the generated traffic.
- Pattern B:** 20% of nodes receive 10 times more network traffic than the others. This pattern is representative of the behavior observed in *fluidanimate* and *vips*. Compared to Pattern A, this pattern is relatively less skewed.
- Pattern C:** 20% of nodes receive 50 times more network traffic than the others. This pattern is representative of the behavior observed in *blacksholes* and *x264*. This pattern lies in-between Patterns A and B in terms of traffic skewness.

Another important attribute pertaining to on-chip network behavior is the utilization of the virtual networks. It turns out that the traffic flowing in the virtual networks of the NoC is not evenly balanced, as demonstrated in Table III. This table shows the percentage of generated packets injected into each of the three virtual networks

Table III. Percentage of Generated Packets Injected into Each of the Three Virtual Networks Required by the MOESI-CMP-Directory Cache Coherence Protocol for Correct Operation
The results for three different system sizes—in terms of the number of identical processing cores—are shown: 64 cores (8×8 mesh), 32 cores (8×4 mesh), and 16 cores (4×4 mesh).

Benchmark	64 cores			32 cores			16 cores		
	Virtual network			Virtual network			Virtual network		
	1	2	3	1	2	3	1	2	3
blackscholes	76.00	23.66	0.34	73.82	26.00	0.19	76.26	23.16	0.59
ferret	77.85	21.34	0.81	74.22	25.41	0.37	77.44	21.40	1.16
fraqmine	76.71	22.75	0.54	74.65	24.89	0.46	77.07	22.07	0.86
swaptions	76.54	22.72	0.74	74.56	25.03	0.41	76.89	22.05	1.07
fluidanimate	77.48	21.58	0.94	74.03	25.69	0.29	77.65	21.22	1.13
vips	76.94	22.52	0.54	74.19	25.44	0.37	77.17	21.89	0.95
x264	77.72	21.23	1.04	74.04	25.61	0.36	77.37	21.51	1.13
Average	77.03	22.26	0.71	74.21	25.44	0.35	77.12	21.90	0.98

required by the MOESI-CMP-directory cache coherence protocol for correct operation. The results for three different system sizes—in terms of the number of identical processing cores—are shown: 64 cores (8×8 mesh), 32 cores (8×4 mesh), and 16 cores (4×4 mesh). In a 64-core system, 77.03%—on average—of the total number of generated packets are injected into the first virtual network, 22.26% into the second, and 0.71% into the last one. As shown in Table III, the virtual network traffic behavior of the PARSEC benchmark applications remains fairly constant across the system size spectrum: The percentage utilizations are very similar for 16, 32, and 64 cores. This indicates that multithreaded applications (as represented by the PARSEC benchmark suite) exhibit similar network behavior—in terms of virtual network utilization—as the number of nodes in the system increases.

Based on these observations, our experiments using synthetic traffic follow the injection pattern identified in Table III: 77%, 22%, and 1% of the generated packets are injected into the three virtual networks, respectively. Note that the packets injected in the three virtual networks are not of the same size. All packets traversing Virtual Networks 2 and 3 are exclusively control packets, which have a size of one flit (128 bits in our experiments), while both control and data packets may traverse Virtual Network 1. The size of data packets is 5 flits. Overall, 72.35% (on average) of all packets injected in the entire network are single-flit control packets, while the remaining 27.65% are 5-flit data packets, as per the PARSEC benchmarks' behavior.

The salient traffic characteristics identified here correspond to a directory-based cache coherence protocol. If the cache coherence protocol type is changed, then these observations will change. However, directory-based protocols (directory-based MOESI in our experiments) currently constitute the most popular protocol used in multicore CMPs, and they have been demonstrated to be scalable to larger-scale CMPs [Jerger et al. 2008; Agarwal et al. 1988]. Thus, we use these traffic patterns to evaluate the scalability of the proposed router.

Having established the behavioral attributes of the traffic patterns to be used, we hereby proceed with the scalability evaluation of the router architectures compared in this work. The scalability analysis results are summarized in Figures 8, 9, and 10.

Figure 8 shows the results under synthetic traffic Pattern A, whereby 10% of the CMP nodes receive 100 times more network traffic than the others. When the number of nodes is 64, as shown in Figure 8(a), the throughput of TornadoNoC is between SINGLE and RING. However, as the number of cores increases, one can observe that the throughput of TornadoNoC decreases more moderately than others. As a result, the throughput of TornadoNoC eventually becomes 4% higher, on average, than the RING

Fig. 8. Scalability analysis using synthetic traffic Pattern A, whereby 10% of the nodes receive 100 times more network traffic than the others. Under this traffic type, very few nodes receive most of the generated traffic, as observed in *ferret*, *fregmine*, and *swaptions*.

Fig. 9. Scalability analysis using synthetic traffic Pattern B, whereby 20% of the nodes receive 10 times more network traffic than the others. This traffic pattern is less skewed than Pattern A and it is representative of the behavior of *fluidanimate* and *vips*.

Fig. 10. Scalability analysis using synthetic traffic Pattern C, whereby 20% of the nodes receive 50 times more network traffic than the others. In terms of traffic skewness, this pattern lies in between Patterns A and B, and it is representative of the behavior of *blackscholes* and *x264*.

router when the number of nodes is 1,024. This trend indicates superior scalability attributes for TornadoNoC. Even though the performance of RING up to 1,024 cores is commendable, it should not be viewed in isolation; when considering the area/power cost as well, then TornadoNoC's *performance-cost efficiency* is substantially higher, as will be shown in the following section.

Figure 9 shows the results under synthetic traffic Pattern B, whereby 20% of the system's nodes receive 10 times more network traffic than the others. Clearly, when the number of nodes is 256 and 1,024, TornadoNoC exhibits the best performance among the compared routers. TornadoNoC's throughput is 5.56% and 8.11% higher than RING at 256 and 1,024 nodes, respectively. More importantly, the trend is the

Fig. 11. Breakdown of average network latency into *queuing* and *hop delay* components, in order to investigate the impact of increasing the hop count in TornadoNoC (due to spillover) on overall network performance.

same: the throughput of TornadoNoC decreases more moderately with increasing number of nodes. Extrapolating this trend, we can expect that TornadoNoC will markedly outperform other routers when the number of nodes exceeds 1,024.

Finally, Figure 10 shows the results under synthetic traffic Pattern C, whereby 20% of the system's nodes receive 50 times more network traffic than the others. Since this pattern falls in between Patterns A and B in terms of traffic skewness, the observed behavior is also somewhere between Figures 8 and 9. Once again, TornadoNoC's throughput decreases at a more moderate pace than the other designs and eventually yields the best performance (6.25% higher than RING) when the number of cores is 1,024.

Most importantly, when performance is viewed in conjunction with the area/power overhead (this is obligatory in the many-core domain, where hundreds of routers are integrated on a single die), TornadoNoC yields a significantly more efficient design, as will be analyzed in Section 5.3.

Since the deflection-based Spillover Flow Control mechanism may result in more network hops (due to spillover) before a packet reaches its destination, it is worth investigating the impact of increasing the hop count in TornadoNoC on overall network performance. Conceptually, the increased hop delay is analogous to increased queuing delay in routers employing credit-based flow control. While hot-spot traffic incurs additional hop delay under Spillover Flow Control, it also incurs longer queuing delays in credit-based flow control, because hot-spot nodes are likely to block subsequent flits. TornadoNoC outperforms credit-based routers, because the increased hop delay is lower than the increased queuing delay.

Figure 11 shows a breakdown of the latencies of SINGLE, RING, and TornadoNoC routers in a network with 1,024 nodes under traffic Pattern C. Figure 11(c) also shows the spillover probability. The probability is computed as the number of spillovers that a flit experiences over the number of ejection trials. The delay is decomposed into queuing and hop delay components. The figure confirms that the hop delay in TornadoNoC increases with the injection rate, but the queuing delay of the SINGLE router increases more rapidly. Before the network saturates (at injection rate 0.016), the increased hop delay is 12.24%—compared with the zero-load latency at injection rate 0.0005. The spillover probability at this rate is 2.5%. Although the spillover probability increases sharply with the injection rate, the probability remains under 5%, even after the saturation point. At the same saturation injection rate (0.016), the average latency of the SINGLE router is more than 500 cycles (not visible in Figure 11(a)), even though its hop delay is lower than that of TornadoNoC. When comparing TornadoNoC to the RING router, the trend is the same: the queuing delay of the RING router increases much more rapidly than the hop delay in TornadoNoC.

Fig. 12. The critical path delay and the hardware area overhead of the proposed router architecture, as extracted from HDL hardware synthesis using 65nm commercial TSMC libraries in the Synopsys Design Compiler.

The results of this subsection may initially look somewhat counterintuitive, as they go against the popular belief that ring-based routers do not scale well. However, the important distinction of the ring-based designs explored in this work is the employment of multiple rings forming a mesh-like or hierarchical topology. This modification yields drastic improvements in the scalability of ring-based routers, as also demonstrated in Fallin et al. [2011], Holliday and Stumm [1994], and Sibai [2008].

Note that, to the best of our knowledge, there have not been any ring-based routers proposed that can support VCs. Without VCs, a router suffers from the HoL blocking problem, which limits its scalability (in addition to the inability to handle protocol-level deadlocks). Remember that the original design of the RING router [Kim 2009] does not support VCs. In our experiments, we enabled VC support in the RING router by employing multiple buffers in the ring stops and the IRI, which boosted the router's scalability. TornadoNoC supports VCs by default through the placement of multiple buffers in the IRI and the use of the Spillover Flow Control mechanism.

Additionally, as explained in Section 3.5, the splitting of the horizontal and vertical rings also contributes to the scalability of TornadoNoC and RING by reducing the contention probability. Compared to RING, TornadoNoC offers better scalability because of its Spillover Flow Control. This flow control technique offers better adaptivity to nonuniform traffic patterns (as observed in real applications).

5.3. Hardware Evaluation

The proposed architecture was *fully* implemented in Verilog Hardware Description Language (HDL) code and synthesized in 65nm commercial TSMC standard-cell libraries using the Synopsys Design Compiler. All the techniques described in Section 3 (the deflection-based Spillover Flow Control mechanism, the sequence numbering mechanism, and the Inflation Mechanism) were included in the implementation. For comparison purposes, we set the same timing constraints to all routers and compare their area and power consumption. The clock frequency is assumed to be 1GHz and the interrouter delay of wires is 0.5ns. Note that all the techniques of Section 3 lie *off* the router's critical path. The critical path delays of the various routers under investigation were extracted from hardware synthesis and are shown in Figure 12(a). The critical path of the proposed TornadoNoC router is shorter than the RING router by 30.43%, but longer than the CANONICAL router by 16%. The RING router has a longer critical path, because it has multiple ring buffers in parallel.

The measured critical path delay was fed back to the real and synthetic workload simulations in order to assess performance assuming that each of the router designs under investigation is clocked to its maximum clock frequency. The new results are shown in Figure 13. The latency is now reported in nanoseconds in order to capture

Fig. 13. Performance results assuming each of the router designs under investigation is clocked to its maximum clock frequency, based on the critical path delay results shown in Figure 12(a). The critical path delay numbers were extracted from the Synopsys Design Compiler.

the different clock periods as extracted from the synthesis results. Since BUF-MinBD has the shortest critical path delay, its performance under PARSEC benchmarks in a 64-core setting is the best among all compared routers, as shown in Figures 13(a) and 13(b). This is because the zero-load latency of BUF-MinBD is the shortest, and the injection rate of PARSEC benchmarks is very low. However, as shown in Figures 13(c), 13(d), and 13(e), the scalability of BUF-MinBD is not comparable to other routers. In fact, it is quite poor. When the number of cores is 1,024, BUF-MinBD saturates much earlier than other routers. Hence, even when considering the maximum achievable clock frequency in each design, TornadoNoC exhibits the best scalability.

Figure 12(b) shows the estimation of the area cost of all synthesized designs, normalized to the SINGLE router. As a practical reference point, we add the area overhead of the canonical three-stage router (CANONICAL) described in Section 5.1. The SINGLE design incurs a significant overhead, while others incur less overhead than the CANONICAL router. This is the reason we consider the SINGLE router as impractical (unrealistic). Nevertheless, the previous subsection demonstrated that the TornadoNoC router offers better scalability than SINGLE, despite incurring much lower overhead. Compared with the other scalability-enhanced low-cost routers, the TornadoNoC router reduces the area overhead by 20.54% and 52.95% when compared against RING and BUF-MinBD, respectively.

As a result, the TornadoNoC router exhibits substantial improvements in terms of *performance-cost efficiency*. Figure 14 shows the *latency-area* product (a composite metric). The system simulated here is a 64-core CMP (8×8 mesh), using the exact same simulation setup and parameters as in Section 5.2.1 (summarized in Table I). This experiment is conducted in our full-system, execution-driven simulator running the PARSEC application benchmarks. The latency results from the simulator are combined with the area results described in this subsection in order to produce the composite *latency-area* metric. Note that the y-axis of Figure 14 is “broken” (i.e., it has a discontinuity) so as to accommodate the higher bars of SINGLE and CANONICAL. The reason for CANONICAL’s large latency-area numbers is the four-cycles-per-hop delay, as

Fig. 14. Latency-area comparison results using an execution-driven, full-system simulator running real applications from the PARSEC benchmark suite [Bienia 2011] on a 64-core CMP. The latency-area product is a composite metric that combines performance and area cost in order to assess a design's *performance-cost efficiency*. The results are normalized to the SINGLE router. Note that the y-axis is "broken" (i.e., it has a discontinuity) so as to accommodate the higher bars of SINGLE and CANONICAL. The reason for CANONICAL's large latency-area numbers is the four-cycles-per-hop delay, as opposed to the single-cycle-per-hop delays of the other routers.

Fig. 15. Power consumption comparison. The per-hop power consumption of each router is extracted from the hardware synthesis results, while the power consumption of links is derived from DSENT [Sun et al. 2012]. These per-hop values are then integrated into the full-system simulation framework. The results are normalized to the power consumption of SINGLE.

opposed to the single-cycle-per-hop delays of the other routers. As shown in Figure 14, the latency-area product of TornadoNoC is reduced by 52.02%, on average, as compared with BUF-MinBD. Compared with RING, TornadoNoC reduces the latency-area product by 21.22%, on average.

It is true that the deflection-based Spillover Flow Control mechanism may result in higher power consumption than conventional deterministic routing, because the former allows spillover, which translates to more network hops before a packet reaches its destination. For the PARSEC benchmarks, we observed that the average number of hops per packet when using the TornadoNoC router design increases by 2.13% as compared to the SINGLE router. However, since the area cost of the TornadoNoC router is substantially lower than other routers, its power consumption is still lower than the others, despite the increase in the number of hops.

In order to compare the power consumption, the per-hop power consumption of each router is extracted from the synthesis results, while the power consumption of the interrouter links is calculated in DSENT [Sun et al. 2012]. These per-hop power values are then integrated into the full-system simulation framework. Figure 15 summarizes

Fig. 16. *Energy-delay* comparison results using an execution-driven, full-system simulator running real applications from the PARSEC benchmark suite [Bienia 2011] on a 64-core CMP. The energy-delay product is a composite metric that combines performance and power consumption. The results are normalized to the SINGLE router.

Fig. 17. Power consumption sensitivity analysis. This figure depicts the power consumption values of each router when varying the switching activity of the input signals.

the results. The overall power consumption of the TornadoNoC router is lower than the RING and BUF-MinBD routers by 20.59% and 57.57%, respectively.

Having established the power efficiency of TornadoNoC, it is interesting to compare the various routers using another pertinent composite metric: the *energy-delay* product. Figure 16 shows the results, assuming the same simulation setup as described earlier in this subsection for the *latency-area* comparisons. The latency results from the simulator are combined with the energy results presented in this subsection in order to produce the composite *energy-delay* metric. The energy-delay product is computed as the *power consumption* $\times$ *execution time*². As shown in Figure 16, the energy-delay product of TornadoNoC is reduced by 56.49%, on average, as compared to BUF-MinBD. Compared to RING, TornadoNoC reduces the energy-delay product by 19.61%, on average. These results support our claim that the proposed TornadoNoC router is the most efficient design among the compared routers.

Finally, Figure 17 presents the results of a sensitivity analysis on the power consumption of each router when varying the switching activity of the input signals. Clearly, the TornadoNoC router consumes the least power among the compared routers across the entire spectrum of switching activity factors. This analysis indicates that the TornadoNoC design is power efficient irrespective of the network traffic levels.

In summary, at the low injection rates of the near-future chips with up to a few hundred cores, TornadoNoC will exhibit comparable performance to RING, but at a significantly lower hardware cost. We believe that a 20.54% reduction in the area cost and 20.59% reduction in the power consumption, as compared to the RING router, are

substantial improvements. When the number of cores increases to more than 1,000 (long-term future), the network will saturate at very low injection rates. In this case, TornadoNoC offers better scalability than RING, as demonstrated in this article.

6. CONCLUSION

Dwindling technology feature sizes have helped materialize the billion-transistor microprocessor. Unprecedented integration densities have enabled the transition to the CMP paradigm. As the number of processing cores increases to the many-core realm, the on-chip communication infrastructure becomes mission critical. Hence, scalable packet-based NoCs are widely considered as the most promising candidates to handle the burgeoning traffic demands of future CMPs. At the same time, the many-core architectural paradigm also requires the communication infrastructure to be as energy efficient as possible to mitigate the effects of the power wall.

In this article, we tackle both design objectives of NoC *scalability* and *energy efficiency* by proposing *TornadoNoC*, a novel ring-based on-chip router that employs a Spillover Flow Control mechanism. A sequence numbering scheme and a dynamic ring inflation technique are also proposed to ensure livelock and deadlock freedom, and their correctness is formally proven. Extensive simulations using both synthetic traffic patterns and real application workloads running in a full-system simulation framework validate our assertion that the proposed router offers better scalability than existing designs while minimizing the hardware cost (i.e., area and power) overhead.

These results demonstrate the promise of ring-based NoCs to serve as a high-performance, yet frugal (in terms of both area and power consumption), communication backbone to future CMPs. While researchers have primarily focused on conventional, crossbar-based router architectures, this work highlights the potential of ring networks as viable alternatives.

ACKNOWLEDGMENTS

This research was partly supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the R&D program supervised by the KCA (Korea Communications Agency). KCA-2013-11921-03001.

REFERENCES

- ABOUSAMRA, A., MELHEM, R., AND JONES, A. 2012. Deja vu switching for multiplane NoCs. In *Proceedings of the 6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12)*. 11–18.
- AGARWAL, A., SIMONI, R., HENNESSY, J., AND HOROWITZ, M. 1988. An evaluation of directory schemes for cache coherence. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*. 280–289.
- AGARWAL, N., KRISHNA, T., PEH, L.-S., AND JHA, N. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*.
- BAHN, J. H. AND BAGHERZADEH, N. 2008. A generic traffic model for on-chip interconnection networks. In *Proceedings of the 1st International Workshop on Network on Chip Architectures*.
- BIENIA, C. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. thesis, Princeton University.
- BOURDUAS, S. AND ZILIC, Z. 2007a. A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing. In *Proceedings of the 1st International Symposium on Networks-on-Chip*. 195–204.
- BOURDUAS, S. AND ZILIC, Z. 2007b. Latency reduction of global traffic in wormhole-routed meshes using hierarchical rings for global routing. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'07)*. 1. 302–307.
- CHUANG, J.-H. AND CHAO, W.-C. 1994. Torus with slotted rings architecture for a cache-coherent multiprocessor. In *Proceedings of the 1994 International Conference on Parallel and Distributed Systems*. 76–81.
- COFFMAN, E. G., ELPHICK, M., AND SHOSHANI, A. 1971. System deadlocks. *ACM Comput. Surv.* 3, 2, 67–78.

- DAS, R., EACHEMPATI, S., MISHRA, A., NARAYANAN, V., AND DAS, C. 2009. Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. In *Proceedings of the 15th International Symposium on High Performance Computer Architecture*. 175–186.
- ESMAEILZADEH, H., BLEME, E., AMANT, R., SANKARALINGAM, K., AND BURGER, D. 2011. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th International Symposium on Computer Architecture*. 365–376.
- FALLIN, C., CRAIK, C., AND MUTLU, O. 2011. CHIPPER: A low-complexity bufferless deflection router. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture*.
- FALLIN, C., NAZARIO, G., YU, X., CHANG, K., AUSAVARUNGNIRUN, R., AND MUTLU, O. 2012. Minbd: Minimally-buffered deflection routing for energy-efficient interconnect. In *Proceedings of the 6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12)*. 1–10.
- FALLIN, C., YU, X., NAZARIO, G., AND MUTLU, O. 2011. A high-performance hierarchical ring on-chip interconnect with low-cost routers. Technical Report, Computer Architecture Lab (CALCM), Carnegie Mellon University.
- GEBHARDT, D., YOU, J., AND STEVENS, K. 2011. Design of an energy-efficient asynchronous NoC and its optimization tools for heterogeneous SOCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 9, 1387–1399.
- GILL, G., ATTARDE, S., LACOURBA, G., AND NOWICK, S. 2011. A low-latency adaptive asynchronous interconnection network using bi-modal router nodes. In *Proceedings of the 5th IEEE/ACM International Symposium on Networks on Chip (NoCS'11)*. 193–200.
- GÓMEZ, C., GÓMEZ, M. E., LÓPEZ, P., AND DUATO, J. 2008. Reducing packet dropping in a bufferless NoC. In *Proceedings of the 14th International Euro-Par Conference on Parallel Processing*. Springer-Verlag, 899–909.
- GRATZ, P., KIM, C., McDONALD, R., KECKLER, S., AND BURGER, D. 2006. Implementation and evaluation of on-chip network architectures. In *Proceedings of International Conference on Computer Design*.
- GREENFIELD, D., BANERJEE, A., LEE, J.-G., AND MOORE, S. 2007. Implications of rent's rule for NoC design and its fault-tolerance. In *Proceedings of the 1st International Symposium on Networks-on-Chip, 2007. (NOCS'07)*. 283–294.
- HAYENGA, M., JERGER, N. E., AND LIPASTI, M. 2009. SCARAB: A single cycle adaptive routing and bufferless network. In *Proceedings of the 42nd International Symposium on Microarchitecture*.
- HOLLIDAY, M. AND STUMM, M. 1994. Performance evaluation of hierarchical ring-based shared memory multiprocessors. *IEEE Transactions on Computers* 43, 52–67.
- HORAK, M., NOWICK, S., CARLBERG, M., AND VISHKIN, U. 2011. A low-overhead asynchronous interconnection network for GALS chip multiprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 30, 4, 494–507.
- JAIN, T. N. K., GRATZ, P. V., SPRINTSON, A., AND CHOI, G. 2010. Asynchronous bypass channels: Improving performance for multi-synchronous NoCs. In *Proceedings of the 2010 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS'10)*. IEEE Computer Society, Washington, DC, 51–58.
- JERGER, N. D. E., PEH, L.-S., AND LIPASTI, M. H. 2008. Circuit-switched coherence. In *Proceedings of the 2nd ACM/IEEE International Symposium on Networks-on-Chip*. 193–202.
- KELM, J. H., JOHNSON, D. R., JOHNSON, M. R., CRAGO, N. C., TUOHY, W., MAHESRI, A., LUMETTA, S. S., FRANK, M. I., AND PATEL, S. J. 2009. Rigel: An architecture and scalable programming interface for a 1000-core accelerator. In *Proceedings of the 36th International Symposium on Computer Architecture*. 140–151.
- KIM, J. 2009. Low-cost router microarchitecture for on-chip networks. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 255–266.
- KIM, J., NICOPOULOS, C., AND PARK, D. 2006. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. *SIGARCH Computer Architecture News* 34, 2, 4–15.
- KIM, J., PARK, D., THEOCHARIDES, T., VIJAYKRISHNAN, N., AND DAS, C. R. 2005. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings of the 42nd Design Automation Conference (DAC'05)*. 559–564.
- KUMAR, A., PEH, L.-S., KUNDU, P., AND JHA, N. K. 2007. Express virtual channels: Towards the ideal interconnection fabric. In *Proceedings of the 34th International Symposium on Computer Architecture*.
- KUMAR, R. 2012. Wavesync: A low-latency source synchronous bypass network-on-chip architecture. In *Proceedings of the 2012 IEEE 30th International Conference on Computer Design (ICCD'12)*. IEEE Computer Society, Washington, DC, 241–248.
- KUMARY, A., KUNDUZ, P., SINGHX, A., PEHY, L.-S., AND JHAY, N. 2007. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *Proceedings of the 25th International Conference on Computer Design*. 63–70.

- LEE, J., LEE, H. G., HA, S., KIM, J., AND NICOPOULOS, C. 2012. A programmable processing array architecture supporting dynamic task scheduling and module-level prefetching. In *Proceedings of the 9th conference on Computing Frontiers*. 153–162.
- MARTIN, M. M. K., SORIN, D. J., BECKMANN, B. M., MARTY, M. R., XU, M., ALAMELDEEN, A. R., MOORE, K. E., HILL, M. D., AND WOOD, D. A. 2005. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Computer Architecture News* 33, 2005.
- MATSUTANI, H., KOIBUCHI, M., AMANO, H., AND YOSHINAGA, T. 2009. Prediction router: Yet another low latency on-chip router architecture. In *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture*. 367–378.
- MICHELOGIANNAKIS, G., BALFOUR, J., AND DALLY, W. 2009. Elastic-buffer flow control for on-chip networks. In *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture*. 151–162.
- MICHELOGIANNAKIS, G., PNEVMATIKATOS, D., AND KATEVENIS, M. 2007. Approaching ideal NoC latency with pre-configured routes. In *Proceedings of the 1st International Symposium on Networks-on-Chip*.
- MOSCIBRODA, T. AND MUTLU, O. 2009. A case for bufferless routing in on-chip networks. *ACM SIGARCH Computer Architecture News* 37, 3, 196–207.
- MULLINS, R., WEST, A., AND MOORE, S. 2004. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*. 188.
- MULLINS, R., WEST, A., AND MOORE, S. 2006. The design and implementation of a low-latency on-chip network. In *Proceedings of Asia and South Pacific Conference on Design Automation*.
- NICOPOULOS, C., PARK, D., KIM, J., VIJAYKRISHNAN, N., YOUSIF, M., AND DAS, C. 2006. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *Proceedings of the 39th International Symposium on Microarchitecture*. 333–346.
- PARK, S., KRISHNA, T., CHEN, C.-H., DAYA, B., CHANDRAKASAN, A., AND PEH, L.-S. 2012. Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI. In *Proceedings of the 49th Annual Design Automation Conference*. ACM, New York, NY, 398–405.
- RAVINDRAN, G. AND STUMM, M. 1997. A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks. In *Proceedings of 3rd International Symposium on High-Performance Computer Architecture*. 58–69.
- SIBAI, F. 2008. Adapting the hyper-ring interconnect for many-core processors. In *International Symposium on Parallel and Distributed Processing with Applications*. 649–654.
- SONG, Y. H. AND PINKSTON, T. 2003. A progressive approach to handling message-dependent deadlock in parallel computer systems. *IEEE Transactions on Parallel and Distributed Systems* 14, 3, 259–275.
- SOTERIOU, V., WANG, H., AND PEH, L.-S. 2006. A statistical traffic model for on-chip interconnection networks. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006*. 104–116.
- SUN, C., CHEN, C.-H., KURIAN, G., WEI, L., MILLER, J., AGARWAL, A., PEH, L.-S., AND STOJANOVIC, V. 2012. DSENT—a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the 6th International Symposium on Networks on Chip*. 201–210.
- TILE-Gx FAMILY. 2012. Homepage. Retrieved from <http://www.tilera.com/>.
- WIND RIVER SYSTEMS. 2012. Homepage. Retrieved from <http://www.windriver.com/>.

Received April 2013; revised October 2013; accepted November 2013