SWIFTNoC: A Reconfigurable Silicon-Photonic Network with Multicast-Enabled Channel Sharing for Multicore Architectures

SAI VINEEL REDDY CHITTAMURU, Colorado State University
SRINIVAS DESAI, Broadcom Ltd.
SUDEEP PASRICHA, Colorado State University

On-chip communication is widely considered to be one of the major performance bottlenecks in contemporary chip multiprocessors (CMPs). With recent advances in silicon nanophotonics, photonics-based network-on-chip (NoC) architectures are being considered as a viable solution to support communication in future CMPs as they can enable higher bandwidth and lower power dissipation compared to traditional electrical NoCs. In this article, we present *SwiftNoC*, a novel reconfigurable silicon-photonic NoC architecture that features improved multicast-enabled channel sharing, as well as dynamic re-prioritization and exchange of bandwidth between clusters of cores running multiple applications, to increase channel utilization and system performance. Experimental results show that *SwiftNoC* improves throughput by up to 25.4× while reducing latency by up to 72.4% and energy-per-bit by up to 95% over state-of-the-art solutions.

CCS Concepts: • **Networks → Network on chip**; • **Computer systems organization → Multicore architectures**; • **Hardware → Photonic and optical interconnect**; **Emerging optical and photonic technologies**

Additional Key Words and Phrases: Network-on-chip (NoC), photonic channel sharing, arbitration

ACM Reference Format:
Sai Vineel Reddy Chittamuru, Srinivas Desai, and Sudeep Pasricha. 2017. SWIFTNoC: A reconfigurable silicon-photonic network with multicast-enabled channel sharing for multicore architectures. J. Emerg. Technol. Comput. Syst. 13, 4, Article 58 (June 2017), 27 pages.
DOI: http://dx.doi.org/10.1145/3060517

1. INTRODUCTION

Advances in technology scaling over the past several decades have enabled the integration of billions of transistors on a single die. Such a massive number of transistors has allowed multiple processing cores and more memory to be integrated on a chip to meet the rapidly growing performance demands of modern applications. With continued technology scaling, designers have observed high wire delays, high energy consumption, and low reliability in traditional electrical shared buses designed in ultra-deep submicron processes [Pasricha et al. 2008b]. Emerging multi-core architectures with high core counts have thus adopted more scalable packet switched electrical network-on-chip (NoC) fabrics [Dally et al. 2001] to support on-chip transfers. But as core

This research is supported by grants from SRC, NSF (CCF-1252500, CCF-1302693), and AFOSR (FA9550-13-1-0110).
Authors' addresses: S. V. R. Chittamuru (corresponding author) and S. Pasricha, Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA, 80523-1373; S. Desai, Broadcom Limited, 4420 Arrowswest drive, Colorado Springs, USA, 80907; email: {sai.chittamuru, sudeep}@colostate.edu, srinivas.desai@broadcom.com.

DOI: http://dx.doi.org/10.1145/3060517

counts steadily increase, such electrical NoC communication fabrics [Kalray 2015; Intel 2016] are beginning to suffer from cripplingly high power dissipation and severely reduced performance [Zhou et al. 2013]. Moreover, the susceptibility of metallic interconnects to crosstalk and electromagnetic interference has also increased with technology scaling, which has further reduced the performance and reliability of electrical NoCs [Pasricha et al. 2008b]. Thus, there is a crucial need to investigate newer and more viable alternatives to metallic interconnects for NoCs.

Recent advances in the area of silicon nanophotonics have enabled the integration of photonic devices with complementary metal-oxide semiconductor (CMOS) circuits. The resulting on-chip photonic interconnects have demonstrated several prolific advantages over their metallic counterparts. Photonic interconnects enable near-light-speed transfers as they employ photons for data communication that are 10× faster compared to electrons in metallic (copper) interconnects [Miller 2009]. Photonic links can achieve distance-independent bit-rates compared to distance dependent (crosstalk-limited) lower bit-rates in electrical wires. The photonic links are also able to achieve high bandwidth density that is 5× higher compared to electrical wires by employing dense wavelength division multiplexing (DWDM) [Bahirat and Pasricha 2011]. In DWDM-based photonic communication, multiple wavelengths of light can be used to simultaneously transfer multiple streams of data in a single photonic waveguide. Additionally, photonic links have lower power dissipation (about 7.9fJ/bit) by dissipating energy only at the endpoints of the communication channel [Miller 2009] with low crosstalk [Zhou et al. 2013]. Thus, silicon nanophotonics is being considered as an exciting new option for integration in future NoCs. Several photonic devices such as microring resonators, waveguides, and photodetectors have already been fabricated and demonstrated at the chip level [Xu et al. 2007; Koester et al. 2007]. These devices have been used as a foundation for several photonic NoC architectures [Vantrease et al. 2008; Bahirat and Pasricha 2009; Pan et al. 2009].

A few prior works have emphasized the importance of network resource contention in photonic NoC channels and proposed arbitration techniques to resolve this contention [Vantrease et al. 2008; Pan et al. 2010]. However, a limitation of these approaches is that they do not fully exploit available network bandwidth and typically only target single parallel application workloads when designing and optimizing the proposed techniques. In emerging multicore systems where multiple applications execute simultaneously on unique subsets of cores, there is significantly greater variation in temporal and spatial characteristics of network injected traffic. For example, cores running memory-intensive tasks can require more network bandwidth than cores running compute-intensive tasks [Pimpalkhute et al. 2014].

To overcome all of these shortcomings, we propose a novel photonic NoC architecture called *SwiftNoC* that utilizes multicast enabled multiple-write-multiple-read (MWMR) photonic waveguides in a crossbar topology and supports dynamic performance adaptation to aggressively utilize network bandwidth and meet diverse application demands. The *SwiftNoC* architecture improves data transfer rate in its MWMR waveguides through a better and faster arbitration mechanism. We compare *SwiftNoC* against alternative architectures with the best-known arbitration mechanisms from prior work, for synthetic and multi-threaded PARSEC [Bienia et al. 2008] workloads on chip multiprocessors (CMP) platform sizes ranging from 64 cores to 256 cores.

The novel contributions of this work can be summarized as follows:

—A flexible on-chip photonic network architecture (*SwiftNoC*) that facilitates selective and reconfigurable prioritization of applications based on their time-varying performance goals;
—Multicast enabled MWMR waveguide in *SwiftNoC* that facilitates energy efficient multicast of messages;

—Improved distributed concurrent token stream arbitration that provides multiple simultaneous tokens and increases channel utilization;
—A dynamic bandwidth transfer technique with low overhead to transfer unused bandwidth among clusters of cores;
—A mechanism to monitor traffic being injected into the network by different co-running applications to facilitate dynamic arbitration wavelength injection rate modulation.

2. RELATED WORK

As per projections from the International Technology Roadmap for Semiconductors (ITRS) [WorkGroup ITRS 2013], in the near future, delay and power consumption of copper-based electrical interconnects will become a serious bottleneck for chip design. These projections motivate the exploration of new technologies to enable viable fabrication of CMPs in future process technologies. On-chip networks in the TILE-Gx72 Processor with 72 cores [Mellonox 2015] and Intel 80-core Terascale processor [Vangal et al. 2008] consume approximately 20–30% of the total chip power. This trend is expected to continue as more wire density becomes available in future process technologies [WorkGroup ITRS 2013]. Thus, the expected on-chip network power will continue to rise as we scale to several hundreds of cores on a single integrated chip (IC).

To overcome this challenge, several novel interconnect technologies are beginning to be explored, including carbon nanotubes (CNTs) [Kreupl et al. 2004; Srivastava et al. 2005; Pasricha et al. 2008a; Vivo et al. 2010] and wireless interconnects [Zhao et al. 2008; Deb et al. 2012; Wettin et al. 2013; Matsutani et al. 2014; Pande et al. 2014] for on-chip communication. However, CNT fabrication is not yet mature and has serious practical concerns to overcome. Multiband RF transmission lines and wireless interconnects (RF-Is) require high operating frequencies in the range of hundreds of GHz to THz. Complex RF-I Frequency Division Multiple Access, transmission lines, or on-chip antennas also entail high area, power, verification, and implementation costs. Nonetheless, these technologies are quite promising and may become more viable in the near future.

Silicon photonic on-chip interconnects are yet another promising alternative for chip-level communication [Bahirat and Pasricha 2012]. A considerable amount of work has focused on the design of photonic NoCs in recent years. The concept of photonic interconnects for on-chip communication was first discussed by Goodman et al. [1984]. Inter-chip photonic interconnects were explored in several works [Chiarulli et al. 1994; Ha et al. 1997; Carrera et al. 1998; Kodi et al. 2004; Kochar et al. 2007; Tan et al. 2008; Batten et al. 2008]. Other efforts have focused on on-chip photonic interconnects with either high-radix low-diameter photonic on-chip crossbar architectures that provide non-blocking connectivity (e.g., [Vantrease et al. 2008; Joshi et al. 2009; Pan et al. 2010; Psota et al. 2010]) or low-radix high-diameter NoCs [Petracca et al. 2008; Cianchettiet et al. 2009; Morris et al. 2010; Kirman et al. 2010; Wu et al. 2010; Kao et al. 2011]. A categorization of various photonic crossbars to meet the design requirements of different CMPs is presented in Li et al. [2014]. Further, there was a recent effort [Xue et al. 2015] that combines reconfigurable adaptive routing and network coding to improve power and performance in electrical NoCs that can also be extended to PNoCs. In addition to these, cross-layer solutions [Li et al. 2015; Chittamuru et al. 2016; Dang et al. 2017] were presented towards the design of thermally resilient PNoCs with enhancements at the circuit, architecture, and operating system (OS) levels. Moreover, few more cross-layer solutions [Chittamuru et al. 2016a; Thakkar et al. 2016a] were presented to mitigate crosstalk noise in PNoCs with enhancements at the device and circuit levels.

Prior work has shown that photonic crossbars are extremely promising on-chip communication architectures to meet future on-chip bandwidths demands, but they can

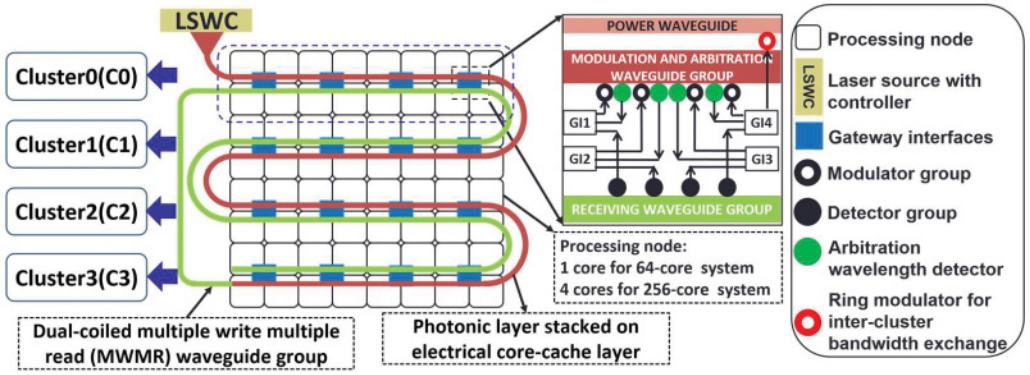

Fig. 1. Layout of MWMR crossbar used in UltraNoC and *SwiftNoC* along with the arrangement of cores and their respective gateway interfaces.

suffer from (i) large power dissipation and (ii) high contention overheads for shared resources, especially when using inefficient token-based arbitration schemes. A few techniques to reduce power overhead of photonic NoCs have been proposed in the literature. For example, an effective policy for runtime management of the laser source is proposed in Chen et al. [2013]. We build on the foundations from their work to manage power dissipation in photonic crossbar NoCs in our work.

To reduce contention issues in crossbars, a few improved arbitration techniques have been proposed in Vantrease et al. [2009] and Pan et al. [2010] that use time division multiplexing (TDM), so a single data waveguide can be simultaneously used by more than one node in different time slots. In Flexishare [Pan et al. 2010], a token stream arbitration scheme is proposed. The scheme requires wavelengths corresponding to each data waveguide to be injected serially into different time slots of an arbitration waveguide. A node writes on the data waveguide only when it gets access to the corresponding arbitration wavelength. Subsequently, the node cannot send data again until its arbitration wavelength is injected into the arbitration waveguide, which takes N cycles for N data waveguides. The scheme leads to channel under-utilization and performs worse as the number of nodes and waveguides increases. In Vantrease et al. [2009], the token ring arbitration scheme from Corona [Vantrease et al. 2008] was improved with the token channel and token-slot arbitration techniques for multiple-write-single-read (MWSR) crossbars. Token-slot arbitration uses TDM and improves on token channel arbitration by dividing the arbitration waveguide into fixed-size, back-to-back slots, with destination nodes circulating tokens in one-to-one correspondence to slots. A limitation of this approach is that a fixed time gap is required between two arbitration slots to set up data for transmission, which reduces the available time slots to send data. UltraNoC [Chittamuru et al. 2015] improves on these prior works by utilizing a more effective concurrent token stream arbitration strategy, together with support for reconfigurable cluster prioritization and bandwidth re-allocation to improve MWMR photonic channel utilization.

3. ULTRANOC AND SWIFTNOC: PHOTONIC ARCHITECTURE OVERVIEW

3.1. UltraNoC Architecture and Terminology

The baseline UltraNoC architecture [Chittamuru et al. 2015] is designed for a 64-core CMP, as shown in Figure 1. We also extend the baseline architecture to a 256-core CMP for the purposes of scalability analysis. Each core has a private L1 and shared L2 cache. In a 64-core CMP, each group of 8 cores has access to main memory via a dedicated

Table I. Micro-Architectural Parameters

CMP Type	64-Core	256-Core
Number of cores	64	256
Number of clusters	4 (16 cores each)	4 (64 cores each)
Per Core:		
L1 I-Cache size/Associativity	16KB/Direct Mapped Cache	
L1 D-Cache size/Associativity	16KB/Direct Mapped Cache	
L2 Cache size/Associativity	128KB/Direct Mapped Cache	
L2 Coherence	MOESI	
Frequency	2.5GHz	
Issue Policy	In-order	
Memory controllers	8	32
Main memory	8GB; DDR4@30ns	32GB; DDR4@30ns

memory controller, whereas in a 256-core CMP, each group of 16 cores has a dedicated memory controller. We have considered memory interleaving in our architecture and adapted its specific implementation from prior work [Cabarcas et al. 2010]. A node (N) is defined as an entity consisting of 1 and 4 cores for the 64-core and 256-core CMPs, respectively. Every node in UltraNoC is attached to a gateway interface (GI) module that facilitates transfers between the CMOS electrical layer and a photonic layer (with photonic waveguides, modulators, detectors, etc.). The entire chip is divided into four clusters (C_0, C_1, C_2, C_3), as shown in Figure 1, where a cluster contains 16 cores in a 64-core CMP and 64 cores in a 256-core CMP. Beyond 256-core CMPs (i.e., 512 or 1024 cores), we can certainly increase the number of clusters (e.g., to 8) to enable more re-configurability in these architectures. An increase in the number of clusters increases the number of arbitration wavelengths (see Section 3.2) in the waveguide, which ultimately requires an increase in DWDM of the waveguide, incurring more power dissipation. Therefore, a careful analysis is required to remain within power constraints while meeting performance objectives. However, our scope of work is limited to 64-core and 256-core CMPs, for which four clusters provides a good tradeoff between performance and power dissipation in our UltraNoC architecture. More details of the micro-architectural parameters of the cores and main memory are shown in Table I.

A detailed layout of the UltraNoC architecture is shown in Figure 1, where 64 nodes (N_0-N_{63}) are arranged in an 8×8 grid. Communication between cores within a node for the 256-core CMP uses an electrical 5×5 NoC router, where four of its input and output port pairs are connected to four cores and the fifth input/output port pair is connected to a GI module. A round-robin arbitration scheme is used within each node for communication between cores and the GI. For higher concentration degree (more than 4 cores within a concentrator) in a general-purpose CMP platform, using a round-robin strategy is a suitable option to achieve fairness for a diverse distribution and choice of workloads; however, if workloads and task to core mapping information is available, then priority-based arbitration schemes (e.g., [Kim et al. 2005]) may be a better choice.

Inter-node transfers are facilitated by dual-coiled MWMR waveguide groups, where each group has four MWMR waveguides. Each MWMR waveguide group in UltraNoC passes every node twice in the dual-coiled structure to enable a two pass inter-node data communication. A node has the ability to write on the first pass using its ring modulators and read from the waveguide group using its ring detectors in the second pass. As all nodes are capable of modulating (writing) in an MWMR waveguide group during the first pass, there is a need for arbitration (see Section 3.2 for more details) between sending nodes to ensure that the data of different senders do not destructively overlap on the shared waveguide group. Throughout this article, this first-pass portion

of the waveguide group is referred to as the *modulating and arbitration waveguide group*. In the second pass of the MWMR waveguide group, all nodes receive data through their respective ring detectors; hence, this portion of the waveguide group is referred to as the *receiving waveguide group*. As all nodes are capable of receiving (reading) from an MWMR waveguide group during the second pass, there is a need for receiver selection (see Section 3.2 for more details) between receiving nodes to ensure that the designated receiver will receive data from the shared waveguide group. Further, each node in our architecture is capable of sending (in the first pass) and receiving (in the second pass) data from all the multiple MWMR waveguide groups through their separate ring modulator and ring detector banks, respectively, on each individual MWMR waveguide group. Additionally, there is a power waveguide that runs in parallel with the other waveguides and carries arbitration wavelengths. This waveguide facilitates our bandwidth transfer and priority adaptation techniques (see Sections 3.3 and 3.4).

Figure 1 depicts an expanded view of the collection of GIs for four nodes, which shows the modulating and arbitration, receiving, and power waveguide groups, along with their connection to GIs. As explained above, each modulating and arbitration and then receiving waveguide group has four MWMR waveguides. Among these four MWMR waveguides, the first waveguide has 68 DWDM (i.e., 68 wavelengths represented as λ_0 to λ_{67}) and the remaining three waveguides have 64 DWDM each. In the first MWMR waveguide 4 wavelengths (λ_0–λ_3) are used for arbitration, and the remaining 64 wavelengths (λ_4–λ_{67}) are used for data transfer and receiver selection. During the receiver selection process, each of the 64 wavelengths is assigned to a unique receiving node (i.e., λ_4, λ_5, ... , λ_{67} are assigned to N_0, N_1, ... , N_{63} respectively), such that whenever a receiver detects its corresponding wavelength during a clock cycle, it switches its detectors "on" to receive data in the next clock cycle. All other receivers keep their detectors turned off to save power. More details about the usage of these wavelengths in the first MWMR waveguide is presented in the next subsection (Section 3.2). As each waveguide in this MWMR waveguide group uses 64 wavelengths for data transfer, each waveguide group in the UltraNoC architecture facilitates simultaneous transfer of a total of 512 bits of data with data modulation at both clock edges in a clock cycle. Thus, in an MWMR waveguide group, each ring modulator and detector group has 256 ring modulators and 256 ring detectors, respectively, that are accessed at the positive and negative edges of the clock.

For powering the waveguides, we use a broadband off-chip laser source with a laser power controller (LSWC). The LSWC has groups of ring modulators capable of injecting different wavelengths in different clock cycles. As we use 68 DWDM in the first MWMR waveguide and 64 DWDM in the remaining three MWMR waveguides of an MWMR waveguide group, there are 260 ring modulators in each ring modulator group in the LSWC. These ring modulators either allow (in non-resonance mode) or remove (in resonance mode) their corresponding wavelengths from the waveguide group. Therefore, these ring modulators in the LSWC inject either of the four arbitration wavelengths (i.e., λ_0–λ_3) in the arbitration slot, the remaining 64 receiver selection wavelengths (i.e., λ_4–λ_{67}) in the receiver selection slot, and the same 64 receiver selection wavelengths (i.e., λ_4–λ_{67}) in the data slot (*SwiftNoC* uses the same set of wavelengths for the receiver selection process and data transfers). Further on-off switching time of a ring modulator is about 3.1ps [Bahirat and Pasricha 2012], which is less than one clock cycle (i.e., 400ps) at 2.5GHz frequency. The laser controller also has control logic that can alter the rate of injection of arbitration wavelengths into each waveguide group. The LSWC's ring modulators and its control logic are assumed to be fabricated on-chip [Chen et al. 2013]. More details about the LSWC are presented in the following subsections and overhead analysis is given in Section 4.1.

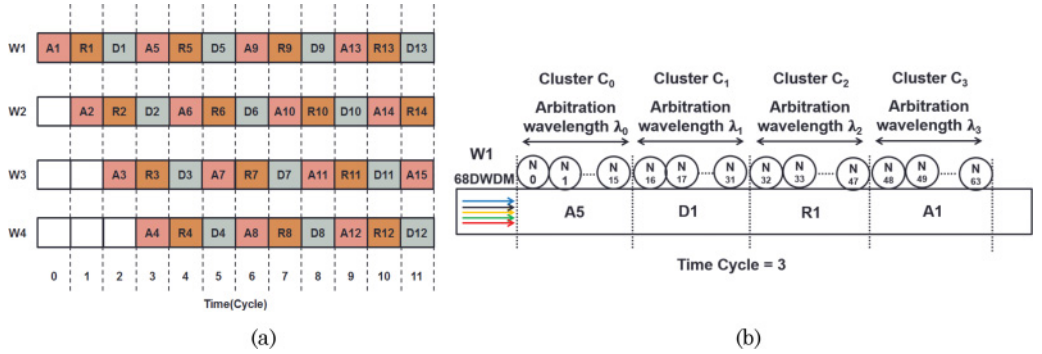

Fig. 2. (a) Timing diagram of arbitration in UltraNoC, which shows distribution of arbitration (Ai), receiver selection (Ri), and data slots (Di) across four MWMR waveguide groups (W1–W4); (b) distribution of different slots within MWMR waveguide group W1 at time cycle 3.

3.2. MWMR Concurrent Token Stream Arbitration and Receiver Selection in UltraNoC

In UltraNoC, all of the cores on a chip are partitioned into four clusters (C_0–C_3), and each cluster is assigned a dedicated arbitration wavelength (λ_0–λ_3). Each MWMR waveguide group is divided into a fixed number of time slots, based on the time taken by light to traverse the waveguide on a die. Based on geometric calculations, each pass of the MWMR waveguide takes four cycles in our architecture at 2.5GHz clock frequency. Thus, we divide each MWMR waveguide group into eight time slots (four time slots for the first pass and four time slots for the second pass). The time slots are further classified into three types: *arbitration slot*, *receiver selection slot*, and *data slot*.

Figure 2(a) shows an example of the distribution of time slots across four MWMR waveguide groups (note: a minimum of eight MWMR waveguide groups are used in our architecture; we only show four in the figure for brevity). As per the explanation provided in the previous subsection, in the arbitration slot, the LSWC injects the arbitration wavelengths of clusters, selectively using a modulator group to dedicate the arbitration slot to a particular cluster. Further, in UltraNoC each receiving node N_i is assigned a receiver selection wavelength λ_{i+4} (see Section 3.1). Thus, after a sending node grabs an arbitration wavelength in the arbitration slot, it gets access to the next receiver selection slot that initially has all the receiver selection wavelengths injected by the LSWC. In this receiver selection slot, the sending node removes all the receiver selection wavelengths except the one corresponding to its receiving node using its modulators bank. Subsequently, in the next data slot, the sending node modulates data on the 64 wavelengths (λ_4–λ_{67}) in each waveguide group assigned for data transfer. In the receiving portion of the MWMR waveguide (second pass of dual-coiled MWMR waveguide) whenever a receiver selection slot reaches a receiving node (N_i), the receiving node only switches on its detector corresponding to its receiver selection wavelength λ_{i+4}. Whenever a receiving node detects its receiver selection wavelength in the receiver selection slot, it switches on its remaining detectors to receive data in the next data slot.

We illustrate this sending-and-receiving process with an example. In Figure 2(b), suppose N_1 in cluster C_0 needs to send data to N_{31} in cluster C_1 that has a corresponding receiver selection wavelength λ_{35}. N_1 first grabs arbitration wavelength λ_0, which is dedicated to cluster C_0, in the arbitration slot. N_1 then modulates in the next receiver selection slot, such that only λ_{35} (the dedicated wavelength for receiver selection of N_{31}) is made available by removing all the wavelengths except λ_{35} (using its ring modulators) in that receiver selection slot. On the receiving end, all the detecting nodes

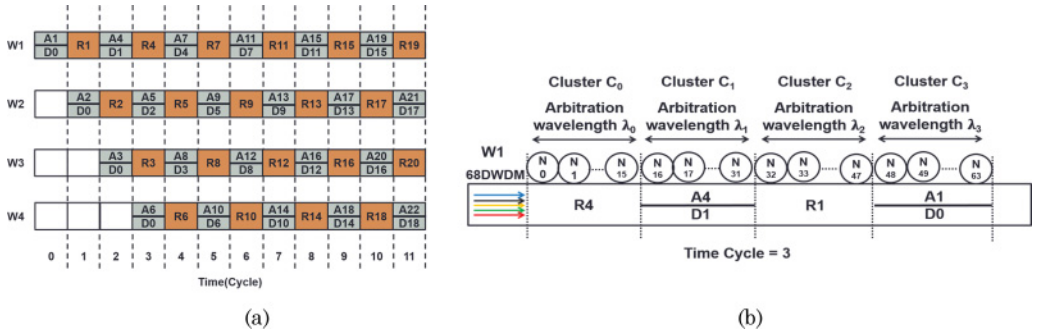

Fig. 3. (a) Timing diagram of arbitration in *SwiftNoC*, which shows distribution of arbitration (Ai), receiver selection (Ri), and data slots (Di) across four MWMR waveguide groups (W1–W4); (b) distribution of different slots within MWMR waveguide group W1 at time cycle 3.

that are in the receiver selection slot switch on their detectors for the corresponding receiver selection wavelengths (e.g., nodes N_{24} to N_{31} switch on detectors with resonance wavelengths λ_{28} to λ_{35}). Thus, at N_{31}, only the detector for wavelength λ_{35} is switched on in the receiver selection slot. Once λ_{35} is detected, N_{31} prepares to receive data in the next data slot by switching on the remaining detectors in that node.

Figure 2(b) shows a snapshot of the position of different slots in the MWMR waveguide group W1 at time cycle 3 for the example in Figure 2(a). As our architecture divides each pass of an MWMR waveguide group into four slots, each slot covers 16 nodes in a particular time instance. The stream of tokens (i.e., stream of arbitration slots with arbitration wavelengths dedicated to a specific cluster) on concurrent slots in waveguide groups allows multiple nodes to inject packets simultaneously on the same MWMR waveguide, resulting in extremely high channel utilization of each MWMR waveguide group. Further in our architecture, multiple nodes can inject packets across different MWMR waveguide groups as well, as each node has a separate modulator and detector bank on each MWMR waveguide group. As each arbitration slot covers 16 nodes on each MWMR waveguide at a time instant, therefore, two or more nodes (up to 16 nodes) from the same cluster can arbitrate for the same arbitration slot on each MWMR group. Ultimately, one of them gets access to the arbitration slot by grabbing the arbitration wavelength. We employ a round-robin arbiter within each cluster to resolve this contention among the 16 nodes within a cluster and avoid starvation.

3.3. Improved MWMR Concurrent Token Stream Arbitration in SwiftNoC

As discussed in the previous subsection, UltraNoC uses separate wavelengths for arbitration (4 wavelengths λ_0–λ_3) and data transfer (64 wavelengths λ_4–λ_{67}). Further from Figure 2, it can be observed that arbitration slots (Ai+1) and data slots (Di) in UltraNoC are adjacent to each other. We propose to overlap arbitration and data slots in our improved *SwiftNoC* architecture. This overlapping mechanism effectively reduces the number of slots for each data transfer from three in UltraNoC to two in the *SwiftNoC* architecture. Figure 3 illustrates the *SwiftNoC* version of the timing diagram for UltraNoC shown in Figure 2. Figure 3(a) shows an example of the distribution of time slots across four MWMR waveguide groups, with overlapped arbitration and data slots. Further, Figure 3(b) shows the position of different slots in the MWMR waveguide group W1 at time cycle 3 for the example in Figure 3(a) with arbitration and data slots overlapped. The *SwiftNoC* architecture improves utilization of MWMR waveguides compared to the MWMR waveguide utilization in UltraNoC,

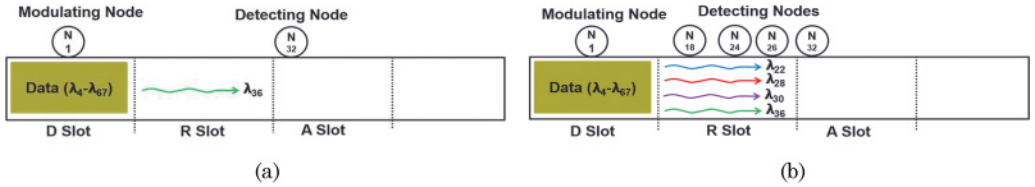

Fig. 4. (a) Transmission of unicast data from Node N_1 to Node N_{32} in *SwiftNoC*, which shows receiver selection wavelength λ_{36} in receiver selection slot (R Slot) of the MWMR waveguide; (b) multicast of data from Node N_1 to Nodes N_{18}, N_{24}, N_{26}, and N_{32} in *SwiftNoC*, which shows respective receiver selection wavelengths λ_{22}, λ_{28}, λ_{30}, and λ_{36} in receiver selection slot (R Slot) of the MWMR waveguide.

which results in an increase in available bandwidth and reduced average packet latency in comparison to the UltraNoC architecture.

3.4. Multicasting of Messages in SwiftNoC

In CMP's with cache coherency support, multicast traffic makes up a significant portion of total network traffic. For example, in the MOESI cache coherence protocol, when a shared block is invalidated, an invalidate message must be multicast to all sharers of that particular shared block. The UltraNoC architecture presented in Sections 3.1 and 3.2 will translate these multicast messages into several unicast messages and send them to their respective destination nodes. These unicast messages cause network congestion and may reduce network performance [Jerger et al. 2008].

In *SwiftNoC*, we avoid such repeated unicast messages by providing multicasting support in its MWMR waveguides. Unlike Corona [Vantrease et al. 2008] and Firefly [Pan et al. 2009] architectures, where all multicast messages are broadcast and transmitted to all nodes in the network, *SwiftNoC* enables multicasting to specific nodes in the network. This is realized as follows: Each sending node in *SwiftNoC*, after removal of the arbitration wavelength from the arbitration slot, releases multiple receiver selection wavelengths corresponding to multiple receiving nodes in the next receiver selection slot (in contrast, in UltraNoC, a sender node, after removal of the arbitration wavelength from the arbitration slot, releases the wavelength of a single receiving node in the next receiver selection slot). In the immediately following data slot, the sending node modulates data that need to be multicast to different receivers. To enable photonic multicast of data in MWMR waveguides, we partially de-tune the ring detectors from their resonating wavelengths [Li et al. 2012], such that a portion of the photonic energy continues on in the MWMR waveguide to be absorbed in subsequent ring detectors. Multicasting thus requires higher laser power compared to unicasting to maintain sufficient photonic signal intensity for detection in the worst case, that is, for the detectors of the last receiving node that receives the multicast data. Laser power injected into the MWMR waveguide for multicasting operation in *SwiftNoC* does not change with the number of nodes that need to receive the multicast message. We designed the laser source for the worst-case power loss, which occurs when all the receiving nodes receive a multicast message from a sending node. We have considered this extra laser power overhead when presenting energy consumption results for the *SwiftNoC* architecture in our results section. In this work, we do not consider optimizing laser power through a laser power management scheme. However, it is possible to integrate previously proposed laser power management schemes [Thakkar et al. 2016; Chen et al. 2013] with our work, as these works are orthogonal to our work.

Figures 4(a) and (b) illustrate the difference between transmission of unicast and multicast messages in our *SwiftNoC* architecture. Suppose N_1 in cluster C_0 needs to multicast data to N_{18}, N_{24}, N_{26}, and N_{32}, whose corresponding receiver selection

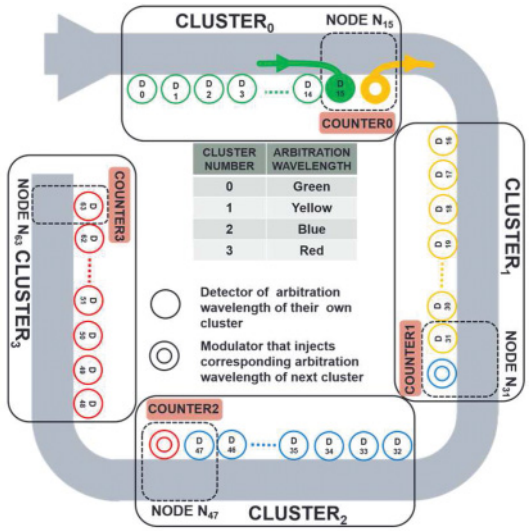

Fig. 5. Bandwidth transfer technique: A cluster can transfer its unused bandwidth to the next cluster by absorbing its own arbitration wavelength and releasing arbitration wavelength of next cluster.

wavelengths are λ_{22}, λ_{28}, λ_{30}, and λ_{36}, respectively. N_1 first grabs arbitration wavelength λ_0, which is dedicated to cluster C_0, in the arbitration slot. N_1 then modulates in the next receiver selection slot, such that only λ_{22}, λ_{28}, λ_{30}, and λ_{36} are made available by removing all the wavelengths except λ_{22}, λ_{28}, λ_{30}, and λ_{36} (using its modulators) in that receiver selection slot. At the receiver end at N_{18}, N_{24}, N_{26}, and N_{32}, the detectors for wavelengths λ_{22}, λ_{28}, λ_{30}, and λ_{36}, respectively, are switched on when these nodes are in the receiver selection slot. At N_{18}, once λ_{22} is detected in the receiver selection slot, the node prepares to receive data in the next data slot by partially de-tuning the ring detectors from its resonating wavelengths in that node. The partial de-tuning of ring detectors of N_{18} will remove a portion of light available in the MWMR waveguide, leaving the remaining portion of light for the other detectors to absorb. Similarly, on detection of λ_{28}, λ_{30}, and λ_{36}, nodes N_{24}, N_{26}, and N_{32}, respectively, prepare to receive data in the next data slot. Our *SwiftNoC* architecture does not differentiate between unicast and multicast transmissions, as it always employs partial detuning to receive both unicast and multicast messages. To further improve channel utilization in the *SwiftNoC* architecture, we adapt the inter-cluster bandwidth transfer mechanism from UltraNoC, as described in in the next subsection.

3.5. Inter-cluster Bandwidth Exchange in SwiftNoC

SwiftNoC supports inter-cluster bandwidth transfers to further improve channel utilization and overall performance. As an example, if cluster C_0 does not need to transfer data, then it transfers its bandwidth to a subsequent cluster C_1. Similarly, any cluster can transfer its unused bandwidth to the subsequent clusters. Figure 5 presents an overview of the bandwidth transfer technique. The last node in each cluster is provisioned with an extra ring modulator that is capable of injecting the arbitration wavelength of the next cluster. Whenever a ring detector of the last node of a cluster detects its own arbitration wavelength, and if this node does not have data to transfer, this indicates a case where the cluster has not used its bandwidth.

Figure 5 illustrates an example of the bandwidth transfer process. Clusters C_0–C_3 are assigned with arbitration wavelengths highlighted with green, yellow, blue, and

red, respectively. The last node in the first three clusters (N_{15}, N_{31}, and N_{47}) is shown with an extra ring modulator that facilitates the injection of the arbitration wavelength of the next cluster. For this example, nodes in C_0 do not need to transfer any data in the current cycle. Then N_{15}, which is the last node in C_0, removes its clusters' arbitration wavelength (green) from the arbitration slot and injects the arbitration wavelength of C_1 (yellow) so nodes in C_1 can use this arbitration slot for sending data in the next available data slot. The bandwidth exchange mechanism performs arbitration wavelength conversion in an arbitration slot in one cycle, thus it has minimal delay/control overhead. The presence of additional microrings for the bandwidth transfer mechanism does lead to more through losses on the MWMR waveguide, which ultimately increases total laser power dissipation. This increase in laser power is included in the laser power dissipation of the 68 DWDM MWMR waveguide ($P_{MWMR\text{-}MCT}$) in Table III. Figure 5 also shows counters at nodes N_{15}, N_{31}, N_{47}, and N_{63} that are used to count the number of arbitration wavelength conversions over a time interval. The next subsection presents more details about the need for these counters.

3.6. Cluster Priority Adaptation with LSWC Reconfiguration

SwiftNoC also supports runtime alteration of allocated bandwidth to each cluster, to closely track changing application bandwidth needs, by altering the number of arbitration slots dedicated to each cluster (i.e., cluster priority). This is essential, because while our bandwidth transfer technique can transfer unused arbitration slots from one cluster to another in the direction of the concurrent arbitration token stream flow (e.g., C_0 to C_1), it lacks the ability to transfer bandwidth in the opposite direction (e.g., C_1 to C_0). To overcome this limitation, we design a cluster priority adaptation mechanism to more comprehensively manage cluster bandwidth allocations over time. This mechanism also helps to minimize laser power by intelligently reducing the total number of injected arbitration slots for runtime scenarios with low-bandwidth (traffic) requirements. The cluster priority adaptation technique consists of three main steps, as discussed below:

Step 1: Determination of wavelength conversion count: Each cluster C_0–C_3 has associated weights W_0–W_3, which determine the proportion of arbitration slots (and, consequently, bandwidth or priority) assigned to the cluster. Initially, these weights can be set to be equal, that is, 0.25 each. At runtime, whenever the last node in a cluster performs a wavelength conversion from its current cluster arbitration wavelength to the next cluster arbitration wavelength, a counter (shown in Figure 5) is incremented. This conversion event represents the case where an unused arbitration slot (bandwidth) is transferred from one cluster to another. Over a time interval T, the recorded wavelength conversion counts $WCC_0 - WCC_3$ from each cluster are then used to determine the unused bandwidth for each cluster.

Step 2: Calculation of excess arbitration slots: The wavelength conversion count values of different clusters show the aggregate number of excess arbitration slots, which includes excess arbitration slots of the present cluster along with the excess arbitration slots of predecessor clusters. The excess arbitration slots for the i^{th} cluster (ES_i) are calculated using Equation (1) by subtracting the cluster wavelength conversion count of the predecessor cluster (WCC_{i-1}) from the wavelength conversion count of the cluster under consideration (WCC_i). ES_i values can also be negative, when a cluster consumes a greater number of arbitration slots (made available by predecessor clusters) than its allocated arbitration slots. Such a cluster has a deficit of arbitration slots,

$$ES_i = \begin{cases} WCC_i, & i = 0 \\ WCC_i - WCC_{i-1}, & i > 0 \end{cases}. \tag{1}$$

Step 3: Setting new weight (priority) for each cluster: Based on the estimation of excesses and deficits in arbitration slots assigned across clusters, this final step attempts to adjust weight values of each cluster to eliminate the excesses and deficits. To determine the new weight of the i^{th} cluster W_i(next) for the upcoming time interval, we must subtract the excess weight EW_i of the cluster from its current weight W_i(current). We can calculate EW_i by dividing the excess arbitration slots of the i^{th} cluster (ES_i), calculated in Equation (1), by the total number of arbitration slots released in the time interval T, which we denote as K. The following equations show these calculations:

$$EW_i = ES_i/K, \tag{2}$$

$$W_i\,(next) = W_i\,(current) - EW_i. \tag{3}$$

Based on the values of the new weights, the LSWC changes the distribution of arbitration wavelengths injected for the next time interval T, such that a cluster with a higher weight will receive more arbitration wavelengths, presenting it with more opportunities to use the waveguides for data transfer. These weight values are communicated to all clusters, so arbiters can adjust their local counters to match the new arbitration slot profile in the waveguides.

4. EXPERIMENTS

4.1. Experimental Setup

To evaluate our proposed *SwiftNoC* architecture, we compared it to a traditional electrical mesh (EMesh) based NoC as well as to four state-of-the-art photonic crossbar NoCs: UltraNoC with concurrent token stream arbitration [Chittamuru et al. 2015], Flexishare with token stream arbitration [Pan et al. 2010], Firefly with reservation-assisted single-write-multiple-reader (R-SWMR) data waveguides [Pan et al. 2009], and Corona with an enhanced token-slot arbitration [Vantrease et al. 2009]. We modeled and simulated the architectures at a cycle-accurate granularity with a SystemC-based NoC simulator for two CMP platform complexities: 64 core and 256 core. We used random synthetic traffic for preliminary analysis of the proposed architectures. Subsequently, we used the PARSEC benchmark suite [Bienia et al. 2008] to create multi-application workloads, with clusters running parallelized versions of different benchmarks from this suite for more detailed comparisons.

Table II shows the PARSEC benchmarks we considered, classified into three categories according to their memory intensities. Compute-intensive benchmarks spend most of the time computing and less time communicating with memory; whereas memory-intensive applications spend a larger portion of their execution time communicating with memory and less time computing within cores. Hybrid intensity benchmarks demonstrate both compute and memory-intensive phases. We created 12 multi-application workloads from these benchmarks. Each workload combines four benchmarks, and the memory intensity of the workloads varies across the spectrum, from compute intensive to memory intensive. As an example, the SC-BT-BS-VI workload combines parallelized implementations of Streamclusters (SC), Bodytrack (BT), Blackscholes (BS), and Vips (VI) and executes them in clusters C_0, C_1, C_2, and C_3, respectively. Each parallelized benchmark is executed on a group of 16 cores and 64 cores in the 64-core and 256-core CMP platforms, respectively. A system-level simulation was performed with the open-source GEM5 [Binkert et al. 2011] architectural simulator with 64 and 256 ARM-based cores running parallelized PARSEC benchmarks to generate traces that were fed into our cycle-accurate NoC simulator. More details about cache sizes, cache associativity, cache coherence, issue policy, memory controllers, and dynamic random access memory (DRAM) sizes that we have considered to generate these traces are presented in Table I. We set a "warm-up" period of 100 million instructions

Table II. Memory Intensity Classification of PARSEC Benchmarks

Application	Representation	Workload Type
Blackscholes	BS	Compute intensive
Bodytrack	BT	Compute intensive
Vips	VI	Compute intensive
Dedup	DU	Compute intensive
Freqmine	FQ	Hybrid
Ferret	FR	Hybrid
Fluidanimate	FA	Hybrid
X264	X264	Hybrid
Streamclusters	SC	Memory intensive
Canneal	CA	Memory intensive
Facesim	FS	Memory intensive
Swaptions	SW	Memory intensive

and then captured traces for the subsequent 1-billion instructions. Then a trace-driven simulation was performed with our cycle-accurate SystemC based NoC simulator.

We targeted 32nm and 22nm process technologies for the 64-core and 256-core CMPs, respectively. Based on the geometric calculation of the waveguides for a 20mm$\times$20mm chip dimension, we estimated the time needed for light to travel from the first to the last node in a single pass of the MWMR waveguide group in *SwiftNoC* as four cycles at 2.5 GHz clock frequency. The same clock and four-cycle round trip time is also applicable to the waveguides in the UltraNoC, Flexishare, Firefly, and Corona photonic crossbar NoCs. Throughout our analysis, we use a flit size of 64 bits for EMesh and a total packet size of 512 bits. Further, we also consider a similar packet size of 512 bits for all photonic NoC architectures. We consider data modulation at both clock edges to enable simultaneous transfer of 512 bits in a single cycle in the *SwiftNoC*, UltraNoC, Flexishare, Firefly, and Corona architectures. We presented architectural information about all the PNoC architectures used in our analysis in Table IV.

The static and dynamic energy consumption of electrical routers is based on results obtained from the open-source DSENT tool [Sun et al. 2012]. Energy consumption of various photonic components for all the photonic NoC architectures are adapted from photonic device characterizations in line with state-of-the-art proposals [Zheng et al. 2011; Grani et al. 2014; Chittamuru et al. 2016b] and shown in Table III. Here $E_{dynamic}$ is the energy/bit for modulators and photodetectors, and $E_{logic-dyn}$ is the energy/bit for the driver circuits of modulators and photodetectors. P_{MWMR}, P_{MWSR}, and P_{SWMR} are the static power consumption of an MWMR, MWSR, and SWMR waveguide group, respectively, which includes the power overhead of ring resonator thermal tuning. The static power consumption of each MWMR waveguide group with multicasting enabled in the *SwiftNoC* architecture is shown as $P_{MWMR\text{-}MCT}$. Further, we have considered a power dissipation overhead of 0.12W and 0.1W in the electrical circuits of the 68 and 64 DWDM MWMR-MCT waveguides, respectively, to realize partial detuning while still maintaining acceptable bit-error-rate as low as 10^{-9}, based on the estimation from the prior work [Li et al. 2012]. We consider a ring heating power of 15μW per ring and detector responsivity of 0.8A/W [Zheng et al. 2011]. To compute laser power consumption, we calculated photonic loss in components, which sets the photonic laser power budget and, correspondingly, the electrical laser power. Last, based on our gate-level analysis, area and power overheads are estimated to be 0.011mm^2 and 0.023W, respectively, for the electrical circuitry (e.g., adders, multipliers, comparators) in the LSWC for our priority adaptation mechanism for *SwiftNoC* at 32nm. We set the reconfiguration delay overhead in *SwiftNoC* to be 20 cycles to account for the time to transfer wavelength

Table III. Energy and Losses for Photonic Devices
[Zheng et al. 2011; Grani et al. 2014]

Energy consumption type	**Energy**
$E_{dynamic}$	0.42pJ/bit
$E_{logic-dyn}$	0.18pJ/bit
Static power per waveguide group	**Power**
P_{MWMR} (with 64 DWDM)	3.73W
$P_{MWMR\text{-}MCT}$ (with 68 DWDM)	5.32W
$P_{MWMR\text{-}MCT}$ (with 64 DWDM)	4.95W
P_{MWSR} (with 64 DWDM)	2.35W
P_{SWMR} (with 64 DWDM)	1.15W
Photonic loss type	**Loss (in dB)**
Microring through	0.02
Waveguide propagation per cm	1
Waveguide coupler/splitter	0.5
Chip coupling	1
Waveguide bending loss	0.005 per 90^0

Table IV. Properties of Various PNoC Architectures

Architecture	**Waveguide Type**	**Arbitration Scheme**	**Multicast Ability**	**Packet Size**
SwiftNoC-8	MWMR	Improved Concurrent Token Stream	Yes	512 bits
SwiftNoC-16	MWMR	Improved Concurrent Token Stream	Yes	512 bits
SwiftNoC-32	MWMR	Improved Concurrent Token Stream	Yes	512 bits
UltraNoC-8	MWMR	Concurrent Token Stream	No	512 bits
UltraNoC-16	MWMR	Concurrent Token Stream	No	512 bits
UltraNoC-32	MWMR	Concurrent Token Stream	No	512 bits
FLEXISHARE	MWMR	2-Pass Token Stream	No	512 bits
FIREFLY	SWMR	-	Yes	512 bits
CORONA	MWSR	Fair Token Slot	No	512 bits

conversion counter values from each cluster to the LSWC, time to determine new priority weights of each cluster, and time to update these values in the arbiters in each cluster.

4.2. Experimental Results

4.2.1. Sensitivity Analysis to Determine Optimal Reconfiguration Window Size. Our first set of experiments presents a sensitivity analysis to explore the optimal dynamic bandwidth and priority reconfiguration time interval window size in *SwiftNoC*. We explore two variants of our architecture: *SwiftNoC-8*, which uses 8 waveguide groups and *SwiftNoC-16*, which uses 16 waveguide groups.

Figure 6(a) shows the energy-delay-product (EDP) for three multi-application PARSEC workloads in *SwiftNoC-8* and *SwiftNoC-16*, with window lengths varying from 100 to 10,000 cycles. In this analysis, to compute EDP we have considered energy consumption of PNoC only (core + cache energy consumption is not considered in our analysis). The three workloads were chosen to possess high, medium, and low aggregate memory intensity to explore the impact of varying memory intensities on window size. At a particular window size, this figure shows higher EDP for memory-intensive workloads compared to compute-intensive workloads, as memory-intensive workloads route more packets in SwiftNoC, which increases their dynamic energy consumption

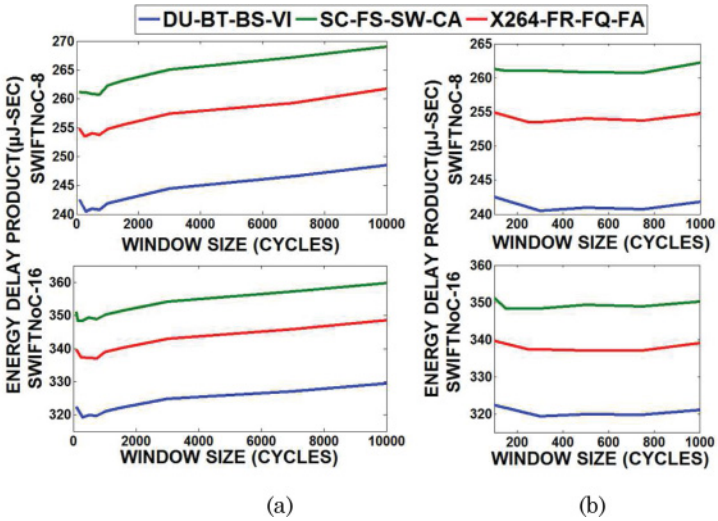

Fig. 6. Energy-delay-product (EDP) comparison for SwiftNoC-8 and SwiftNoC-16 in a 64-core CMP with time interval window sizes (a) 100–10,000 cycles (b) 100–1,000 cycles (zoomed version of Figure 6(a)).

and average packet latency (due to increased network congestion), thereby increasing overall EDP. Also, for both memory- and compute-intensive workloads, a large window size should intuitively result in lower reconfiguration overhead but will also result in less reactivity to changing application traffic demands, which ultimately increases average packet latency and EDP as well; while a small window size will result in higher reconfiguration overhead with higher energy consumption in the reconfiguration hardware and increased EDP but better adaptivity to changing application traffic. A careful observation of the plot in Figure 6 shows that for compute-intensive workloads (i.e., DU-BT-BS-VI) EDP is lower for a larger window size, whereas for memory-intensive workloads (i.e., SC-FS-SW-CA) EDP is lower at smaller window sizes. However, there is an overlap region from 300 cycles to 750 cycles that can be observed from Figure 6(b) (which is the zoomed version of Figure 6(a) between window sizes 100–1000 cycles) where EDP is low for both memory- and compute-intensive workloads. Additionally, results for average throughput and latency also indicate worsening performance beyond 750–1,000 cycles. Thus, we set reconfiguration time interval window size in *SwiftNoC* to 300 cycles to balance reconfiguration overhead and performance. Our analysis of the reconfiguration time interval window size for UltraNoC also indicates an optimal EDP at around 300 cycles, and thus we also set a 300-cycle reconfiguration time interval window size for UltraNoC.

4.2.2. Results of 64-Core System for Synthetic Traffic. Our second set of experiments targets a 64-core CMP platform with a synthetic benchmark that utilizes a uniform random traffic pattern. In uniform random traffic, cores arbitrarily generate packets to random destination cores in the CMP. Cache coherency or multicast traffic is not considered in this analysis with random traffic. We compare network throughput, average packet latency, and EDP of *SwiftNoC* with the electrical mesh (EMesh), UltraNoC with concurrent token arbitration [Chittamuru et al. 2015], Flexishare with token stream arbitration [Pan et al. 2010], Firefly with reservation-assisted single write multiple reader (R-SWMR) data waveguides [Pan et al. 2009], and Corona with token-slot arbitration [Vantrease et al. 2009]. Later in this section, we also present comparison results of *SwiftNoC* architecture (*SwiftNoC-MCT*) for various percentages of multicast traffic to the total traffic of the network.

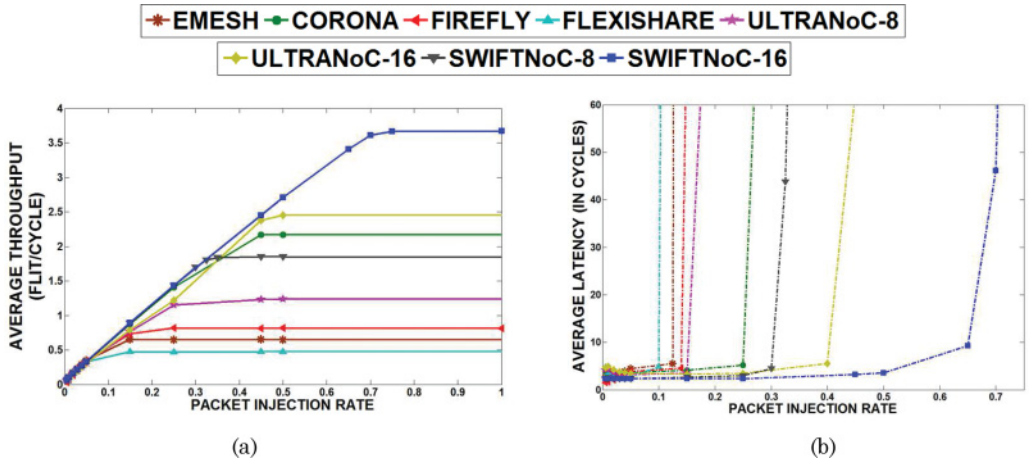

Fig. 7. (a) Average throughput and (b) average latency comparison of *SwiftNoC-8* and *SwiftNoC-16* with UltraNoC-8, UltraNoC-16, Flexishare, Firefly, Corona, and EMesh architectures for a 64-core CMP. Results are shown for uniform random traffic.

The average throughput for uniform random traffic in the 64-core CMP is shown in Figure 7(a). It can be observed that *SwiftNoC* with 8 MWMR waveguides (*SwiftNoC-8*) has 4.2× and 1.7× higher throughput compared to Flexishare and UltraNoC-8 with the same number of MWMR data waveguides. Even though Flexishare uses MWMR waveguides and TDM as in *SwiftNoC*, there are significant differences between these architectures. In Flexishare, arbitration wavelengths corresponding to each MWMR data waveguide are injected serially into an arbitration waveguide. A node that grabs a token in the arbitration waveguide gets exclusive access to the corresponding MWMR data waveguide, which leads to underutilization of the MWMR waveguide. In contrast, *SwiftNoC-8* uses improved concurrent token stream arbitration with TDM such that each MWMR waveguide with multiple arbitration, receiver selection, and data slots can be accessed concurrently by multiple nodes to facilitate simultaneous transfer of multiple packets to improve MWMR waveguide utilization. Moreover, unlike *SwiftNoC*, Flexishare does not support the priority reconfiguration and bandwidth exchange mechanisms. Further, compared to UltraNoC, *SwiftNoC* uses an improved version of concurrent token stream arbitration that increases the data rate in MWMR waveguides by overlapping arbitration and data slots. This overlapping mechanism effectively reduces the number of slots for each data transfer from 3 in UltraNoC to 2 in the *SwiftNoC* architecture and increases throughput for *SwiftNoC-8* compared to UltraNoC-8. The throughput of *SwiftNoC-8* is 2.8× higher than the throughput of EMesh, as our architecture uses faster silicon photonic waveguides for data communication compared to slower electrical links. *SwiftNoC-8* also has 2.2× higher throughput compared to Firefly. This is because *SwiftNoC-8* for a 64-core CMP is an all-optical MWMR crossbar, which transfers all of its data at near light speed, whereas Firefly is a hybrid photonic network, where a significant portion of data traverses through slower electrical links. The *SwiftNoC* configurations with 16 waveguide groups (*SwiftNoC-16*) with approximately twice the number of microring resonators in *SwiftNoC-8* provides even better throughput than the UltraNoC-16 (with 16 MWMR waveguide groups), Firefly, Flexishare, and EMesh architectures. *SwiftNoC-16* has 1.6×, 8.4×, 4.5×, and 5.6× greater throughput compared to UltraNoC-16, Flexishare, Firefly, and EMesh, respectively, for the 64-core CMP.

Table V. Photonic Hardware Comparison

Architecture	Waveguides	Ring Modulators	Ring Detectors	PNoC Area (in mm^2)
SwiftNoC-8	32	131,640	131,584	24.50
SwiftNoC-16	64	263,280	263,168	49.01
SwiftNoC-32	128	526,560	526,336	98.01
UltraNoC-8	32	131,640	131,584	24.51
UltraNoC-16	64	263,280	263,168	49.01
UltraNoC-32	128	526,560	526,336	98.01
FLEXISHARE	33	131,080	131,648	24.58
FIREFLY	64	4,096	28,672	10.25
CORONA	257	1,032,256	20,416	113.47

Figure 7(a) also shows that Corona has greater throughput compared to *SwiftNoC-8*. This is because Corona has 64 MWSR waveguides and an MWMR arbitration waveguide to facilitate communication between 64 cores, which utilizes approximately four times the number of microring resonators compared to *SwiftNoC-8* (as shown in Table V earlier). However, *SwiftNoC* with 16 waveguide groups (*SwiftNoC-16*) and approximately half the number of microring resonators of Corona has 1.9× better throughput compared to Corona. *SwiftNoC-16* uses MWMR waveguides with improved concurrent token stream arbitration and achieves higher data rates through simultaneous data transfers using TDM, which is not possible with the MWSR waveguides used in Corona. The enhanced token-slot arbitration [Vantrease et al. 2009] in Corona requires a fixed time gap between two arbitration slots to set up data for transmission, which reduces available time slots to send data.

From Figure 7(b) it can be seen that, in terms of average latency, *SwiftNoC-8* and *SwiftNoC-16* have better performance compared to Flexishare and Firefly. Flexishare with its inefficient arbitration scheme has underutilized MWMR waveguides, which increases overall packet latency. In contrast, *SwiftNoC* with an improved concurrent token stream arbitration mechanism, bandwidth transfer mechanism, and cluster priority adaptation mechanism increases MWMR waveguide utilization and reduces wait time for packets, which in turn reduces latency. Further, in the reservation assisted Firefly architecture, a sender needs extra cycles to broadcast reservation flits to all the destination nodes, so destination node can tune in on the corresponding SWMR data waveguides to receive the data in the following cycles. These extra cycles lead to lower data rates in SWMR waveguides of the Firefly architecture and increase its average latency compared to *SwiftNoC*. The improved concurrent token stream arbitration scheme used in *SwiftNoC* enables higher communication parallelism compared to token-based arbitration in Corona, which explains the lower latency in *SwiftNoC* compared to Corona. Last, *SwiftNoC* has lower latency compared to UltraNoC with the same number of waveguides, as the improved concurrent token stream arbitration in *SwiftNoC* increases the data rate in its MWMR waveguides by overlapping arbitration and slots, which helps reduce average latency.

Figure 8 summarizes the EDP of our *SwiftNoC* architectures with UltraNoC-8, UltraNoC-16, Flexishare, Firefly, Corona, and EMesh for the uniform synthetic traffic pattern. These results are generated for a packet injection rate of 0.7, for which the throughputs for all of these compared architectures are saturated. Energy consumption includes static, dynamic, and laser energy for every architecture. From these results, it can be seen that *SwiftNoC-8* has 49%, 57%, 91%, and 88% and *SwiftNoC-16* has 17%, 30%, 85%, and 81% lower EDP compared to Flexishare, Firefly, Corona, and EMesh, respectively. Corona has more EDP compared to *SwiftNoC-8* and *SwiftNoC-16*, as it uses more number of microring resonators as shown in Table V, which in turn leads to more static energy consumption. The lower EDP of *SwiftNoC-8* and *SwiftNoC-16*

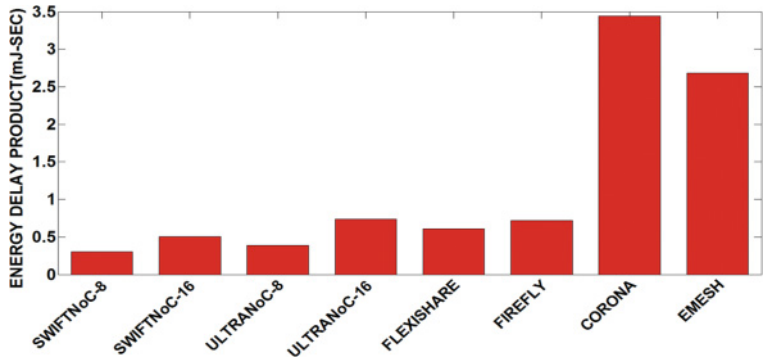

Fig. 8. EDP comparison of *SwiftNoC-8* and *SwiftNoC-16* with UltraNoC-8, UltraNoC-16, Flexishare, Firefly, Corona, and EMesh architectures for a 64-core CMP. Results are shown for uniform random traffic with packet injection rate of 0.7.

compared to Firefly is due to higher energy consumption in the electrical network of the Firefly architecture. Although *SwiftNoC-8* has similar number of microring resonators compared to Flexishare, the improvements in average latency for *SwiftNoC-8* due to improved sharing contribute to its lower EDP. *SwiftNoC-8* also has 21% and 58% lower EDP compared to UltraNoC-8 and UltraNoC-16, respectively, and *SwiftNoC-16* has 31% lower EDP compared to UltraNoC-16. Despite higher dynamic power due to an increase in its data rate (increase in number of data modulation and detection events), *SwiftNoC* has higher EDP savings compared to UltraNoC with similar number of MWMR waveguides because of overlapped arbitration and data slots that reduce average packet latency and EDP. *SwiftNoC-8* also has half the number of microring resonators compared to UltraNoC-16, which in turn results in higher relative static energy consumption and EDP for UltraNoC-16.

As a final set of experiments with synthetic traffic, we present average throughput, average packet latency, and EDP for the *SwiftNoC* architecture for various percentages of multicast traffic to total traffic in the network. The average throughput for uniform random traffic for a 64-core CMP with *SwiftNoC* having 16 MWMR waveguides is shown in Figure 9(a) for 10% (*SwiftNoC-16-MCT-10*), 20% (*SwiftNoC-16-MCT-20*), 30% (*SwiftNoC-16-MCT-30*), 40% (*SwiftNoC-16-MCT-40*), and 50% (*SwiftNoC-16-MCT-50*) of multicast traffic from the total traffic in the network. It can be observed that with an increase in multicast traffic percentage there is a monotonic increase in throughput for *SwiftNoC*. From Figure 9(b), for average latency, *SwiftNoC-16-MCT-50* has lower latency compared to *SwiftNoC-16-MCT-40*, *SwiftNoC-16-MCT-30*, *SwiftNoC-16-MCT-20*, and *SwiftNoC-16-MCT-10*. With the increase in multicast traffic in the network, *SwiftNoC* enables sharing of photonic signals during multicast with partial de-tuning of microring resonators, which allows multiplexing of data streams. This in turn increases throughput and reduces latency with simultaneous delivery of packets to their respective destinations and shows the adaptability of the proposed *SwiftNoC* architecture for higher multicast traffic rates.

Figure 10 shows the EDP comparison among *SwiftNoC-16-MCT-10, SwiftNoC-16-MCT-20, SwiftNoC-16-MCT-30, SwiftNoC-16-MCT-40,* and *SwiftNoC-16-MCT-50* for a 64-core CMP for a uniform synthetic traffic pattern. These results are generated for a packet injection rate of 0.95, where the throughput for all of these compared architectures is saturated. From these results, it can be seen that *SwiftNoC-16-MCT-50* has 50%, 38%, 32%, and 18% lower EDP compared to *SwiftNoC-16-MCT-10, SwiftNoC-16-MCT-20, SwiftNoC-16-MCT-30,* and *SwiftNoC-16-MCT-40*. Although *SwiftNoC-16-MCT-50* has similar number of microring resonators as *SwiftNoC-16-MCT-10,*

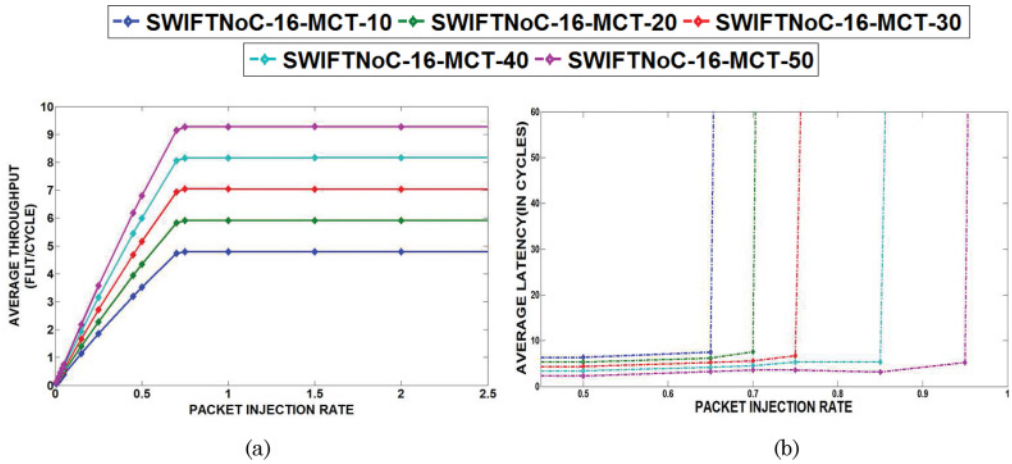

Fig. 9. (a) Average throughput and (b) average latency comparison of *SwiftNoC-16* with random multicast traffic having 10% (*SWIFTNoC-MCT-10*), 20% (*SWIFTNoC-MCT-20*), 30% (*SWIFTNoC-MCT-30*), 40% (*SWIFTNoC-MCT-40*), and 50% (*SWIFTNoC-MCT-50*) of multicast messages for a 64-core CMP.

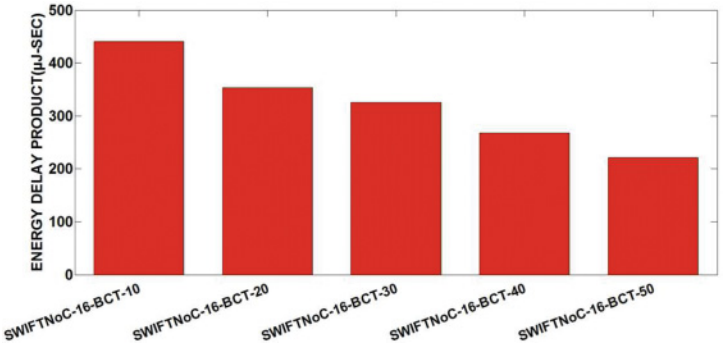

Fig. 10. EDP comparison of *SwiftNoC-16-MCT-10*, *SwiftNoC-16-MCT-20*, *SwiftNoC-16-MCT-30*, *SwiftNoC-16-MCT-40,* and *SwiftNoC-16-MCT-50* for a 64-core CMP. Results are shown for uniform random traffic with different percentages of multicast traffic at packet injection rate of 0.95.

SwiftNoC-16-MCT-20, SwiftNoC-16-MCT-30, and *SwiftNoC-16-MCT-40*, but with higher multicast traffic in the network, *SwiftNoC-16-MCT-50* performs more number of multicasts though partial de-tuning of microring resonators and significantly increases the delivery rate of packets, which reduces overall average latency and decreases EDP.

4.2.3. Experimental Analysis with 64-Core CMP Using PARSEC Benchmark Applications. Our next set of experiments target a 64-core CMP platform and compare network throughput, average packet latency, and energy-per-bit (EPB) of *SwiftNoC* with the electrical mesh (EMesh), UltraNoC, Flexishare, Firefly, and Corona architectures.

Figures 11(a)–(c) show the results of this study, with all results normalized with respect to the EMesh results. From the throughput comparison in Figure 11(a), it can be observed that, not surprisingly, all photonic NoCs provide better throughput than EMesh, due to the presence of higher bandwidth photonic links. Further, *SwiftNoC*, when compared to EMesh, UltraNoC, Flexishare, Firefly, and Corona, has even better throughput improvements for PARSEC benchmark traffic than with synthetic traffic. From Figure 11(a), it can be seen that *SwiftNoC-8* has 7.8× greater throughput compared to EMesh, as well as 9.1× and 2.3× greater throughput compared to

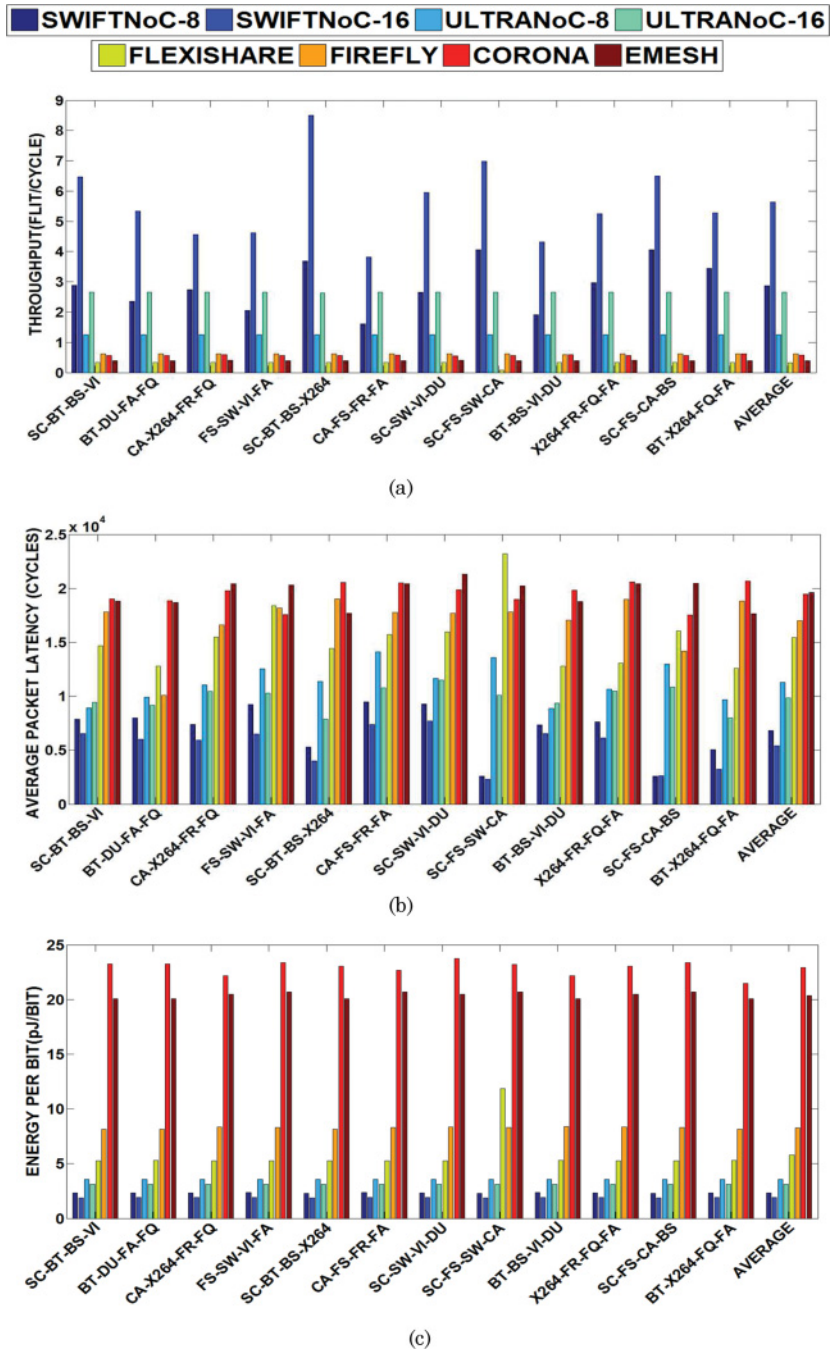

Fig. 11. (a) Average throughput, (b) average packet latency, and (c) average EPB comparison of *SwiftNoC-8* and *SwiftNoC-16* with other architectures for a 64-core CMP. Results are shown for multi-application PARSEC workloads.

Flexishare and UltraNoC-8, with the same number of MWMR waveguides. In Flexishare, as explained in the previous subsection token stream arbitration hinders utilization of MWMR waveguides, whereas in *SwiftNoC*, multiple arbitration, reservation and data slots are available concurrently in an MWMR waveguide, such that each MWMR waveguide can be accessed simultaneously by multiple nodes. The improved concurrent token stream arbitration, which reduces the number of time slots for each data transfer, and multicasting, which enables simultaneous transfer of multiple messages, contribute to increase in the throughput of *SwiftNoC-8* compared to UltraNoC-8. *SwiftNoC-8* also provides 5.1× higher throughput than Corona. This is because of instances in Corona where multiple sender nodes attempt to communicate with a single receiver node (e.g., memory controller). Such instances result in the sender nodes attempting to access the single MWSR waveguide connected to the receiver, creating a significant imbalance among MWSR waveguides, with the other waveguides being underutilized while packets get queued waiting for the waveguide connected to the receiver. *SwiftNoC* avoids such an imbalance with its use of more-efficient MWMR waveguides and improved arbitration. *SwiftNoC-8* also provides 4.6× more throughput than Firefly. *SwiftNoC* for a 64-core CMP is an all optical MWMR crossbar, which transfers data entirely over photonic links and thus has increased throughput compared to Firefly which is a hybrid photonic network, where a significant portion of data traverses through slower electrical links. The bandwidth transfer mechanism and cluster priority adaption mechanism in *SwiftNoC* also increase available bandwidth and contribute to increase in throughput.

SwiftNoC with 16 MWMR waveguide groups (*SwiftNoC-16*) with approximately twice the number of microring resonators of *SwiftNoC-8* provides even better throughput than the other architectures. *SwiftNoC-16* has 2.2×, 17.8×, 9.1×, 9.9×, and 14.2× higher throughput compared to UltraNoC-16, Flexishare, Firefly, Corona, and EMesh, respectively. The improvement is somewhat higher for memory-intensive workloads than for compute-intensive workloads. The large throughput improvement for *SwiftNoC* is a direct consequence of improved concurrent token stream arbitration, multicasting, avoiding unused bandwidth by transferring it to cores that need it the most, and using the bandwidth transfer and priority alteration mechanisms at runtime.

These mechanisms also improve the average packet latency in *SwiftNoC*, as shown in Figure 11(b), by reducing the time spent waiting for access to the photonic waveguides. On average, *SwiftNoC-8* has 39.8%, 55.7%, 59.7%, 65.1%, and 65.3% lower average packet delay over UltraNoC-8, Flexishare, Firefly, Corona, and EMesh, respectively, for the different multi-application workloads. On the other hand, *SwiftNoC-16* has 45.1%, 63.9%, 68.4%, 72.1%, and 72.4% lower average packet delay over UltraNoC-16, Flexishare, Firefly, Corona, and EMesh, respectively. From these results, we can surmise that average latency improvements of *SwiftNoC* over UltraNoC, Flexishare, Firefly, Corona, and EMesh with benchmark traffic and synthetic traffic follow similar trends.

Figure 11(c) shows the EPB comparison between the architectures. It can be observed that, on average, *SwiftNoC-8* has 34%, 25%, 59%, 72%, 90%, and 89% and *SwiftNoC-16* has 47%, 38%, 67%, 77%, 92%, and 91% lower EPB compared to UltraNoC-8, UltraNoC-16, Flexishare, Firefly, Corona, and EMesh, respectively. Most of the energy in the photonic architectures was consumed in the form of static energy. From Table V presented earlier, it can be observed that *SwiftNoC-8* has 75% fewer microring resonators, whereas *SwiftNoC-16* has 50% fewer microring resonators compared to Corona. This allows *SwiftNoC-8* and *SwiftNoC-16* to have lower EPB compared to Corona. Further, *SwiftNoC-8* with a similar number of microring resonators as Flexishare maintains lower EPB with more efficient utilization of its MWMR waveguides through its concurrent token stream arbitration. On the other hand, despite *SwiftNoC-16* using more

hardware than Flexishare, it has lower EPB compared to Flexishare because of more efficient arbitration and multicasting, the bandwidth transfer mechanism, and priority alteration mechanism. Firefly has higher EPB, even though it fewer microring resonators compared to *SwiftNoC-8* and *SwiftNoC-16*. Firefly, being a hybrid network, consumes most of its energy in the electrical network, and this in turn increases overall EPB compared to the *SwiftNoC* architectures. All the variants of our *SwiftNoC* architecture have lower EPB compared to EMesh, as our architectures use energy-efficient photonic links for data transfer instead of power-hungry electrical links. Although *SwiftNoC-8* and *SwiftNoC-16* use power-hungry multicast MWMR waveguides (see Table III), the increase in data rate due to efficient multicasting and improved concurrent token stream arbitration decreases EPB of these architectures compared to different variations of the UltraNoC architecture.

4.2.4. Scalability Analysis with 256-Core CMP Using PARSEC Applications. Our final set of experiments explores the scalability of *SwiftNoC*. We considered a larger 256-core CMP platform by increasing the core concentration in each tile to four to enable higher traffic injection into the network for this scalability study. We evaluated NoC throughput, average packet latency and EPB for all photonic NoCs. In addition to *SwiftNoC-8* and *SwiftNoC-16*, we also considered a *SwiftNoC-32* variant of our architecture with 32 MWMR waveguide groups. Similarly, we also considered an additional UltraNoC-32 variant of the UltraNoC architecture.

Figures 12(a)–(c) show the results of these experiments. From the throughput results in Figure 12(a), it can be seen that, on average, *SwiftNoC-8* has 2.4×, 1.2×, 7.3×, 4.1×, 7.2×, and 4.8×; *SwiftNoC-16* has 5.1×, 2.5×, 16.5×, 9.1×, 16.3×, and 10.8×; and *SwiftNoC-32* has 8×, 3.7×, 25.4×, 14.3×, 25.1×, and 16.6× greater throughput compared to UltraNoC-8, UltraNoC-16, Flexishare, Firefly, Corona, and EMesh. Further, *SwiftNoC-16* and *SwiftNoC-32* have 1.3× and 2.1× higher throughput compared to UltraNoC-32, respectively. The improvements in throughput for *SwiftNoC* over UltraNoC, Firefly, Flexishare, Corona, and EMesh are even better for the 256-core CMP than in the 64-core CMP case. With the increase in core count, the amount of traffic injected into the network increases, and *SwiftNoC*, with its better-utilized MWMR waveguides, effectively handles this traffic, whereas UltraNoC, Firefly, Flexishare, Corona, and EMesh end up moving to the saturation region (throughput will not increase), which explains throughput improvements for *SwiftNoC*.

From Figure 12(b), it can be seen that, on average, *SwiftNoC-8* has 19%, 12%, 7%, 49%, 39%, 46%, and 53%; *SwiftNoC-16* has 27%, 20%, 11%, 54%, 45%, 51%, and 57%; and *SwiftNoC-32* has 39%, 34%, 26%, 62%, 54%, 59%, and 64% lower latency compared to UltraNoC-8, UltraNoC-16, UltraNoC-32, Flexishare, Firefly, Corona, and EMesh architectures, respectively. Last, Figure 12(c) shows that, on average, *SwiftNoC-8* has 39%, 28%, 14%, 63%, 74%, 89%, and 92%; *SwiftNoC-16* has 51%, 42%, 30%, 71%, 79%, 91%, and 94%; and *SwiftNoC-32* has 62%, 55%, 46%, 77%, 84%, 93%, and 95% lower EPB compared to UltraNoC-8, UltraNoC-16, UltraNoC-32, Flexishare, Firefly, Corona, and EMesh architectures, respectively. The average packet latency and EPB improvements in *SwiftNoC* are higher for the 256-core CMP compared to the 64-core CMP. The greater volume of traffic in the 256-core system increases packet wait time in the sending nodes across all the architectures. *SwiftNoC* is able to reduce this wait time with its efficient arbitration, multicasting, bandwidth transfer, and priority alteration mechanisms to achieve better average latency improvements over the UltraNoC, Flexishare, Firefly, Corona, and EMesh architectures. Despite the increase in energy consumption for all the architectures when going from the 64-core CMP to the 256-core CMP, the EPB of the *SwiftNoC* architecture has greater improvements over UltraNoC, Flexishare, Firefly, Corona, and EMesh architectures because of its higher packet delivery rate.

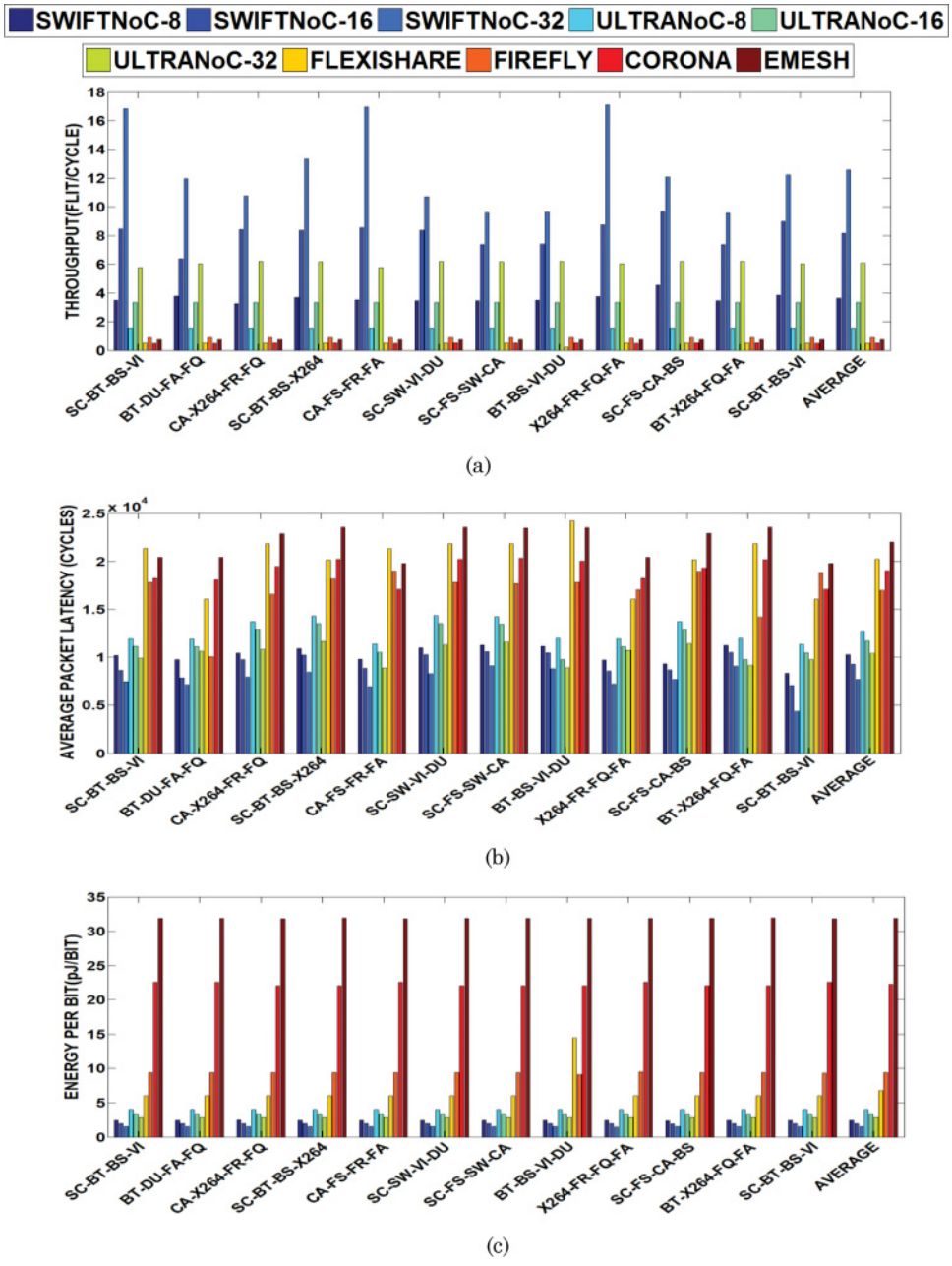

Fig. 12. (a) Average throughput, (b) average packet latency, and (c) average EPB comparison of *SwiftNoC-8, SwiftNoC-16,* and *SwiftNoC-32* with other architectures for a 256-core CMP. Results are shown for multi-application PARSEC workloads.

4.2.5. Summary of Results and Observations. From the results presented in the previous sections, we can summarize that our proposed *SwiftNoC* architecture can achieve better performance with less hardware compared to existing state-of-the-art photonic NoCs for CMP platforms with low as well as high core counts. *SwiftNoC* achieves higher performance and energy efficiency by efficiently utilizing MWMR waveguides with its improved arbitration scheme, bandwidth transfer mechanism, and cluster priority adaptation. *SwiftNoC* also improves on UltraNoC due to its more aggressively concurrent token stream arbitration scheme that increases utilization of MWMR waveguides compared to UltraNoC. The *SwiftNoC* architecture's ability to efficiently multicast messages across different nodes in the network using its multicast-friendly MWMR waveguides also contributes to its higher performance with lower energy consumption.

5. CONCLUSIONS

In this work, we presented the *SwiftNoC* photonic NoC architecture, which is an improved version of the UltraNoC architecture, with more efficient channel sharing among cores with an aggressive concurrent token stream-based arbitration strategy and more efficient multicast support. *SwiftNoC* supports the ability to dynamically transfer bandwidth between clusters of cores and to re-prioritize multiple co-running applications to further improve channel utilization and adapt to time-varying application performance goals. *SwiftNoC* improves throughput by up to 25.4× while reducing latency by up to 72.4% and EPB by up to 95% over state-of-the-art solutions. *SwiftNoC* also scales well with increasing core counts on a chip.

REFERENCES

S. Bahirat and S. Pasricha. 2009. Exploring hybrid photonic networks-on-chip for emerging chip multiprocessors. In *Proceedings of the IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'09)*. 129–136.

S. Bahirat and S. Pasricha. 2011. OPAL: A multi-layer hybrid photonic noc for 3d ics. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'11)*. 345–350.

S. Bahirat and S. Pasricha. 2012. A particle swarm optimization approach for synthesizing application-specific hybrid photonic networks-on-chip. In *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED'12)*.

C. Batten, A. Joshi, J. Orcutt, A. Khilo, and B. Moss. 2008. Building many core processor-to-dram networks with monolithic silicon photonics. In *Proceedings of the 16th Annual Symposium on High Performance Interconnects*. 21–30.

C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC benchmark suit: Characterization and architectural implications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (Pact'08)*. 72–81.

N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. The gem5 simulator. *ACM SIGARCH Comput. Arch. News* 39, 2 (2011), 1–7.

F. Cabarcas, A. Rico, Y. Etsion, and A. Ramirez. 2010. Interleaving granularity on high bandwidth memory architecture for cmps. In *Proceedings of the International Conference on Embedded Computer Systems (SAMOS'10)*.

E. Carrera and R. Bianchini. 1998. OPTNET: A cost effective optical network for multiprocessors. In *Proceedings of the 12th International Conference on Supercomputing (ICS'98)*. ACM Press, New York, NY, 401–408.

C. Chen and A. Joshi. 2013. Runtime management of laser power in silicon-photonic multibus NoC architecture. *IEEE J. Select. Top. Quant. Electron.* 19, 2 (2013), art. 3700713.

D. Chiarulli, S. Levitan, R. Melhem, M. Bidnurkar, R. Ditmore, G. Gravenstreter, Z. Guo, J. Qao, and C. Teza. 1994. Optoelectronic buses for high performance computing. *Proc. IEEE* 82, 11, 1701–1710.

S. V. R. Chittamuru, S. Desai, and S. Pasricha. 2015. A reconfigurable silicon-photonic network with improved channel sharing for multicore architectures. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*. 63–68.

S. V. R. Chittamuru and S. Pasricha. 2016. SPECTRA: A framework for thermal reliability management in silicon-photonic networks-on-chip. In *Proceedings of the International Conference on VLSI Design (VLSID)*.

S. V. R. Chittamuru, I. Thakkar, and S. Pasricha. 2016a. PICO: Mitigating heterodyne crosstalk due to process variations and intermodulation effects in photonic nocs. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*.

S. V. R. Chittamuru, I. Thakkar, and S. Pasricha. 2016b. Process variation aware crosstalk mitigation for dwdm based photonic noc architectures. In *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED)*.

M. J. Cianchettiet, J. C. Kerekes, and D. H. Albonesi. 2009. Phastlane: a rapid transit optical routing network. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09)*. 441–450.

W. J. Dally and B. Towles. 2001. Route packets, not wires. In *Proceedings of the Design Automation conference (DAC'01)*. 684–689.

D. Dang, S. V. R. Chittamuru, R. Mahapatra, and S. Pasricha. 2017. Islands of heaters: A novel thermal management framework for photonic noCs. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*.

S. Deb, K. Chang, A. Ganguly, Y. Xinmin, C. Teuscher, P. Pande, D. Heo, and B. Belzer. 2012. Design of an efficient NoC architecture using millimeter-wave wireless links. In *Quality Electronic Design (ISQED)*. 165–172.

P. Grani and S. Bartolini. 2014. Design options for optical ring interconnect in future client devices. *ACM J. Emerg. Technol. Comput. Syst.* 10, 4 (2014), 30.

J. Goodman, F. Leonberger, K. Sun-Yuan, and R. Athale. 1984. Optical interconnects for vlsi systems. *IEEE Opt. Interconnect. VLSI Syst.* 72, 7, 850–866.

J. Ha and T. Pinkston. 1997. Speed demon: Cache coherence on an optical multichannel interconnect architecture. *J. Parallel Distrib. Comput.* 41, 1 (1997), 78–91.

P. K. Hamedani, N. E. Jerger, and S. Hessabi. 2010. QuT: A low-power optical network-on-chip. In *Proceedings of the IEEE/ACM International Symposium on NoCS*.

R. Ho, K. W. Mai, and M. A. Horowitz. 2001. The future of wires. *Proc. IEEE* 89, 4 (2001), 490–504.

Intel Corporation. 2007. Intel 80-core teraflops research chip. Retrieved from http://www.intel.com/pressroom/kits/teraflops/.

Intel Corporation. 2016. Intel Xeon Phi™ Processor 7290F. Retrieved from http://ark.intel.com/products/95831/Intel-Xeon-Phi-Processor-7290F-16GB-1_50-GHz-72-core.

N. E. Jerger, L. S. Peh, and M. H. Lipasti. 2008. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *ISCA*. 229–240.

A. Joshi, C. Batten, Y. J. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic. 2009. Silicon-photonic clos networks for global on-chip communication. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS'09)*. 124–133.

Y. H. Kao and H. J. Chao. 2011. BLOCON: A bufferless photonic clos network-on-chip architecture. In *Proceedings of the IEEE/ACM International Symposium on NoCS*.

Kalray Inc. 2015. Kalray Bostan MPPA2 256-core processor. Retrieved from http://www.kalrayinc.com/tag/bostan/.

J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das. 2005. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings of the 42nd Annual Conference on Design Automation (DAC'05)*. 559–564.

N. Kirman and J. F. Martinez. 2010. A power-efficient all-optical on-chip interconnect using wavelength-based oblivious routing. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems (ASLOS'10)*. 15–28.

C. Kochar, A. Kodi, and A. Louri. 2007. Nd-rapid: A multidimensional scalable fault-tolerant optoelectronic interconnection for high performance computing systems. *J. Opt. Netw.* 6, 5.

A. Kodi and A. Louri. 2004. RAPID: Reconfigurable and scalable all-photonic interconnect for distributed shared memory multiprocessors. *J. Light-Wave Technol.* 22, 2101–2110.

F. Kreup, A. Graham, M. Liebau, G. Duesberg, R. Seidel, and E. Unger. 2004. Carbon nanotubes for interconnect applications. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'04)*. 683–686.

C. Li, M. Browning, P. V. Gratz, and S. Palermo. 2012. Energy-efficient optical broadcast for nanophotonic networks-on-chip. In *Proceedings of the Optical Interconnects Conference*. 64–65.

H. Li, S. Le Beux, G. Nicolescu, J. Trajkovic, and I. O'Connor. 2014. Optical crossbars on chip, a comparative study based on worst-case propagation losses. *Concurr. Comput.: Pract. Exper.* 26 (15) (2014), 2492–2503.

Z. Li, A. Qouneh, M. Joshi, W. Zhang, X. Fu, and T. Li. 2015. Aurora: A Cross-Layer Solution for Thermally Resilient Photonic Network-on-Chip. *IEEE Trans. VLSI Syst.* 23 (1) (2015), 170–183.

H. Matsutani, M. Koibuchi, I. Fujiwara, T. Kagami, Y. Take, T. Kuroda, P. Bogdan, R. Marculescu, and H. Amano. 2014. Low-latency wireless 3D NoCs via randomized shortcut chips. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE'14)*.

Mellanox Technologies. 2007. TILE-Gx72 Processor. Retrieved from http://www.mellanox.com/page/multi_core_overview.

D. A. B. Miller. 2009. Device requirements for optical interconnects to silicon chips. *Proc. IEEE Spec. Issue Silicon Photon.* 97, 7 (2009), 1166–1185.

R. W. Morris and A. K. Kodi. 2010. Power-efficient and high-performance multilevel hybrid nanophotonic interconnect for multicores. In *Proceedings of theACM/IEEE International Symposium on Networks-on-Chip (NOCS'10)*. 207–214.

Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary. 2009. Firefly: Illuminating future network-on-chip with nanophotonics. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09)*. 429–440.

Y. Pan, J. Kim, and G. Memik. 2010. Flexishare: Channel sharing for an energy efficient nanophotonic crossbar. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA'10)*. 1–12.

P. P. Pande, A. Nojeh, and A. Ivanov. 2014. T1B: Wireless NoC as interconnection backbone for multicore chips: Promises and challenges. In *Proceedings of the System-on-Chip Conference (SOCC). xxxvii – xxxviii.*

S. Pasricha, F. Kurdahi, and N. Dutt. 2008a. System level performance analysis of carbon nanotube global interconnects for emerging chip multiprocessors. In *Proceedings of the IEEE International Symposium on Nanoscale Architectures (NANOARCH'08)*. 1–7.

S. Pasricha and N. Dutt. 2008b. *On-chip Communication Architectures*. Morgan Kauffman.

M. Petracca, B. G. Lee, K. Bergman, and L. P. Carloni. 2008. Design exploration of optical interconnection networks for chip multiprocessors. In *Proceedings of the IEEE Symposium on High Performance Interconnects (HOTI'08*).

T. Pimpalkhute and S. Pasricha. 2014. An application-aware heterogeneous prioritization framework for NoC based chip multiprocessors. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED'14*). 76–83.

J. Psota, J. Miller, G. Kurian, H. Hoffman, N. Beckmann, J. Eastep, and A. Agarwal. 2010. ATAC: Improving performance and programmability with on-chip optical networks. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'10)*. 3325–3328.

N. Srivastava and V. Banerjee. 2005. Performance analysis of carbon nanotube interconnects for vlsi applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'05)*. 383–390.

C. Sun, C. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. Peh, and V. Stojanovic. 2012. DSENT—a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS'12)*. 207–214.

M. Tan, P. Rosenberg, Y. Jong-Souk, M. McLaren, S. Mathai, T. Morris, K. Pei, J. Straznicky, N. Jouppi, and S. Wang. 2008. A high-speed optical multi-drop bus for computer interconnections. In *Proceedings of the 16th IEEE Symposium on High Performance Interconnects*. 3–10.

I. Thakkar, S. V. R. Chittamuru, S. Pasricha. 2016. Run-Time Laser Power Management in Photonic NoCs with On-Chip Semiconductor Optical Amplifiers. In *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*.

I. Thakkar, S. V. R. Chittamuru, S. Pasricha. 2016a. Mitigation of Homodyne Crosstalk Noise in Silicon Photonic NoC Architectures with Tunable Decoupling. In *Proceedings of the IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.

S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. 2008. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE J. Solid-State Circ.* 43, 1 (2008).

D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. Beausoleil, and J. Ahn. 2008. Corona: System implications of emerging nanophotonic technology. In *Proceedings of the International Symposium on Computer Architecture (ISCA'08)*. 153–164.

D. Vantrease, N. Binkert, R. Schreiber, and M. H. Lipasti. 2009. Light speed arbitration and flow control for nanophotonic interconnects. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. 304–315.

B. De Vivo, P. Lamberti, G. Spinelli, and V. Tucci. 2010. Reliable bounds for the propagation delay in VLSI nano interconnects based on Multi Wall Carbon Nano Tubes. In *Signal Propagation on Interconnects* (SPI). 149–152.

P. Wettin, P. P. Pande, H. Deukhyoun, B. Belzer, S. Deb, and A. Ganguly. 2013. Design space exploration for reliable mm-wave wireless NoC architectures. In *Application-Specific Systems, Architectures and Processors (ASAP)*. 79–82.

WorkGroup ITRS. 2013. International technology roadmap for semiconductors, 2013 ed. ITRS Technology Working Groups.

X. Wu, Y. Ye, W. Zhang, W. Liu, M. Nikdast, X. Wang, and J. Xu. 2010. UNION: A unified inter/intra-chip optical network for chip multiprocessors. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'10)*.

Q. Xu, S. Manipatruni, B. Schmid, J. Shakya, and M. Lipson. 2007. 12.5gbit/s carrier-injection-based silicon micro-ring silicon modulators. In *Proceedings of the Conference on Lasers and Electro-Optics (CLEO'07)*. 1–2.

Y. Xue and P. Bogdan. 2015. User Cooperation Network Coding Approach for NoC Performance Improvement. In *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS'15)*.

D. Zhao and Y. Wang. 2008. SD-MAC: Design and synthesis of a hardware-efficient collision-free qos-aware mac protocol for wireless network-on-chip. *IEEE Trans. Comput.* 57, 9 (2008), 1230–1245.

L. Zhou and A. K. Kodi. 2016. PROBE: Prediction-based optical bandwidth scaling for energy-efficient NoCs. In *Proceedings of the IEEE/ACM International Symposium on Networks-on-Chip (NOCS'13)*.

X. Zheng, D. Patil, J. Lexau, F. Liu, G. Li, H. Thacker, Y. Luo, I. Shubin, J. Li, J. Yao, P. Dong, D. Feng, M. Asghari, T. Pinguet, A. Mekis, P. Amberg, M. Dayringer, J. Gainsley, H. F. Moghadam, E. Alon, K. Raj, R. Ho, J. E. Cunningham, and A. V. Krishnamoorthy. 2011. Ultra-efficient 10Gb/s hybrid integrated silicon photonic transmitter and receiver. *Opt. Expr.* 19, 6 (2011), 5172–5186.

Received May 2016; revised February 2017; accepted February 2017