SHARP: Shared Heterogeneous Architecture with Reconfigurable Photonic Network-on-Chip

SCOTT VANWINKLE and AVINASH KARANTH KODI, Ohio University

As the relentless quest for higher throughput and lower energy cost continues in heterogenous multicores, there is a strong demand for energy-efficient and high-performance Network-on-Chip (NoC) architectures. Heterogeneous architectures that can simultaneously utilize both the serialized nature of the CPU as well as the thread level parallelism of the GPU are gaining traction in the industry. A critical issue with heterogeneous architectures is finding an optimal way to utilize the shared resources such as the last level cache and NoC without hindering the performance of either the CPU or the GPU core. Photonic interconnects are a disruptive technology solution that has the potential to increase the bandwidth, reduce latency, and improve energy-efficiency over traditional metallic interconnects.

In this article, we propose a CPU-GPU heterogeneous architecture called *Shared Heterogeneous Architecture with Reconfigurable Photonic Network-on-Chip (SHARP)* that clusters CPU and GPU cores around the same router and dynamically allocates bandwidth between the CPU and GPU cores based on application demands. The SHARP architecture is designed as a Single-Writer Multiple-Reader (SWMR) crossbar with reservation-assist to connect CPU/GPU cores that dynamically reallocates bandwidth using buffer utilization information at runtime. As network traffic exhibits temporal and spatial fluctuations due to application behavior, SHARP can dynamically reallocate bandwidth and thereby adapt to application demands. SHARP demonstrates 34% performance (throughput) improvement over a baseline electrical CMESH while consuming 25% less energy per bit. Simulation results have also shown 6.9% to 14.9% performance improvement over other flavors of the proposed SHARP architecture without dynamic bandwidth allocation.

CCS Concepts: • **Hardware → Network on chip**;

Additional Key Words and Phrases: Network-on-chips, photonic interconnects, power-efficient, performance

ACM Reference format:
Scott VanWinkle and Avinash Karanth Kodi. 2018. SHARP: Shared Heterogeneous Architecture with Reconfigurable Photonic Network-on-Chip. *J. Emerg. Technol. Comput. Syst.* 14, 2, Article 25 (July 2018), 22 pages.
https://doi.org/10.1145/3185383

1 INTRODUCTION

As technology continues to scale down toward sub-nanometer processes, hundreds and even thousands of cores are anticipated to be integrated on a single chip in the future. The cores that can be integrated include both general purpose cores such as central processing units (CPUs)

This research was supported by NSF Awards No. CCF-1054339 (CAREER), No. CCF-1318981, No. CCF-1420718, No. CCF-1513606, and No. CCF-1703013.
Authors' addresses: S. VanWinkle, Ohio University, 305 Stocker Center, Athens, OH, 45701; email: sv247901@ohio.edu; A. K. Kodi, Ohio University, 322D Stocker Center, Athens, OH, 45701; email: kodi@ohio.edu.

https://doi.org/10.1145/3185383

and specialized cores such as graphical processing units (GPUs). Recent work has shown that GPUs can outperform CPUs for similar tasks or applications due to thread level parallelism (see Appendix A) [26]. Research into CPU-GPU architectures has grown extensively in the past few years in an effort to take advantage of heterogeneity as evidenced by Intel's Broadwell [11] and Skylake [10], NVIDIA's Tegra X1 [24], and AMD's Carrizo [2], to name a few. CPU and GPU cores can share multiple on-chip resources, such as the network bandwidth, last level cache (LLC), memory controllers, and main memory. Few of the critical challenges in heterogeneous architectures are power minimization, fairness in resource allocation, control complexity, and improving the overall system performance.

One of the critical infrastructures in heterogeneous platforms is the interconnection network that glues all the cores and the memory hierarchy. While technology scaling is enabling the integration of hundreds of heterogeneous cores, communication demands by these cores that run diverse applications has also increased exponentially. This has resulted in increased power consumption for data communication at the on-chip level. For example, the network consumed 28% of the tile's power in the 80-core TeraFlops prototype machine from Intel [33]. More power-efficient design such as Intel's 48-core Single-Chip Cloud (SCC) dedicated 10% of the chip power to the network where the cores were general purpose CPUs [27]. Therefore, as technology scaling has lowered the energy/bit for data computation, the energy/bit for data communication has increased, and future on-chip communication will become a major impediment to improving multicore performance [6].

Recent work, such as Reference [7], has demonstrated that using disruptive interconnect technology rather than traditional electrical solutions can be beneficial to multicore architectures. One potential technology is photonic interconnects that can leverage Dense Wavelength Division Multiplexing (DWDM), power-efficient modulators/demodulators, and distance-independent transceivers to significantly improve the energy-efficiency, network throughput, and area efficiency of on-chip communication fabric [3]. Prior photonic interconnect research has proposed monolithic, split, and hybrid crossbars for CPU-only architecture [22, 25, 34]. Few prior works such as References [29, 32, 37] have even explored heterogeneous architectures using photonic interconnects.

In this article, we propose Shared Heterogeneous Architecture with Reconfigurable Photonic Network-on-Chip (SHARP), a novel 3D-NoC architecture for CPU-GPU heterogeneous multicores. Our goal is to improve performance (throughput and latency) and energy-efficiency of SHARP by dynamically allocating bandwidth between the CPU and GPU cores. The uniqueness of our proposed approach is that this dynamic allocation is done locally within the router without requiring global coordination across multiple routers. This fine-grain bandwidth reconfiguration is achieved by considering a sliding window of buffer utilization for each core type, thereby balancing the network bandwidth with application demands. While the dynamic bandwidth allocation is implemented to improve performance, we safeguard CPU application from being overrun by GPU application by providing priority to CPU traffic. With recent advances in 3D integration enabled by wafer bonding and thru-oxide vias (TOVs), we propose a similar integration for SHARP architecture to enable higher density through TOVs while lowering parasitics [28]. We evaluate the power (using DSENT [30]) and performance on PARSEC [5], SPLASH2 [35], and OpenCL [1] benchmarks using Multi2Sim [31] with a cycle-accurate network simulator. We compare SHARP to three state-of-the-art optical architectures and a baseline electrical CMESH architecture. We further evaluate the dynamic allocation mechanism by gauging the network response under increased network loads and bandwidth constrained environments. In this article, we make the following important contributions:

- We propose a heterogeneous 3D-NoC architecture that effectively combines both CPUs and GPUs in a single cluster. The proposed single-cluster design efficiently shares network resources to maximize performance.
- We maximize network throughput with a fine-grain dynamic bandwidth reallocation algorithm that adapts to application demands and regulates the bandwidth usage between CPUs and GPUs. As the proposed algorithm is implemented locally, no global coordination is required with multiple routers.
- We evaluate the performance of the dynamic allocation mechanism under high contention and bandwidth constrained environments. Our results indicate that SHARP shows, on an average, a 6.9% to 14.9% throughput improvement over comparable photonic networks while demonstrating a 39% performance improvement over a baseline CMESH. Additionally, SHARP shows up to a 9.7% energy per bit improvement over other competitive photonic architectures with 64 wavelengths per waveguide, while exhibiting a 25% improvement in energy per bit when compared to a baseline CMESH.

2 RELATED WORK

There has been extensive research on photonic interconnect-based architectures for on-chip communication [4, 8, 16, 25, 34, 36]. Corona [34] is a 256-core CPU with a total of 1,024 threads that uses a Multiple-Write Single-Read (MWSR) photonic crossbar-based interconnect. Write access to a link is granted by a token arbitration scheme [25, 34]. In Firefly [25], token arbitration is avoided by using Single-Write Multiple-Read (SWMR) communication. Furthermore, a Reservation-assisted SWMR (R-SWMR) communication scheme is utilized to avoid the high power consumption of having multiple listeners on the same data channel. Recently, a 128 Computational Unit (CU) GPU architecture was proposed that connects two photonic crossbars where the first crossbar communicates between L1 and L2 using MWSR and the second communicates between L2 and L1 using SWMR to save power [38]. While all nodes listen on the channel, similar to R-SWMR, only the intended destination will read the packet to minimize power consumption [38]. Having the destination embedded in the packet header in a SWMR channel eliminates any latency overhead for token arbitration. Recently, several works have contributed to analyze the system-level implications of the power loss, crosstalk, and device losses for designing optical NoC [4, 8, 16, 36]. In this work, we are focused on developing techniques to improve the bandwidth utilization of photonic networks.

Dynamically managing network bandwidth can minimize energy consumption and maximize network utilization. 3D-NoC utilized a multi-layered, split crossbar-based photonic architecture to dynamically allocate bandwidth using link and buffer utilization [22]. In heterogeneous architectures with metallic interconnects, a feedback directed virtual channel partitioning mechanism that allocates different numbers of virtual channels to each core type has been proposed [17]. The proposed mechanism allocates at least one virtual channel to the CPU and thereby prevents memory-intensive GPU traffic from starving the CPU traffic of network resources. Furthermore, there have been several proposed techniques to allocate shared resources, such as the cache, the main memory, and the network in heterogenous architectures [12, 14, 17]. In Reference [29], dynamic bandwidth allocation is implemented using a R-SWMR link by globally allocating bandwidth when an application on a core is changed. Once the change in bandwidth occurs, the bandwidth is reallocated using a token arbitration mechanism implemented on a separate controller waveguide.

In this work, we propose a novel approach to bandwidth allocation by clustering four GPUs and two CPUs around one router. By clustering, we propose a local arbitration that takes advantage of the buffer information already available at each router. Based on the buffer utilization, we distribute the bandwidth to CPUs and GPUs locally without requiring global coordination.

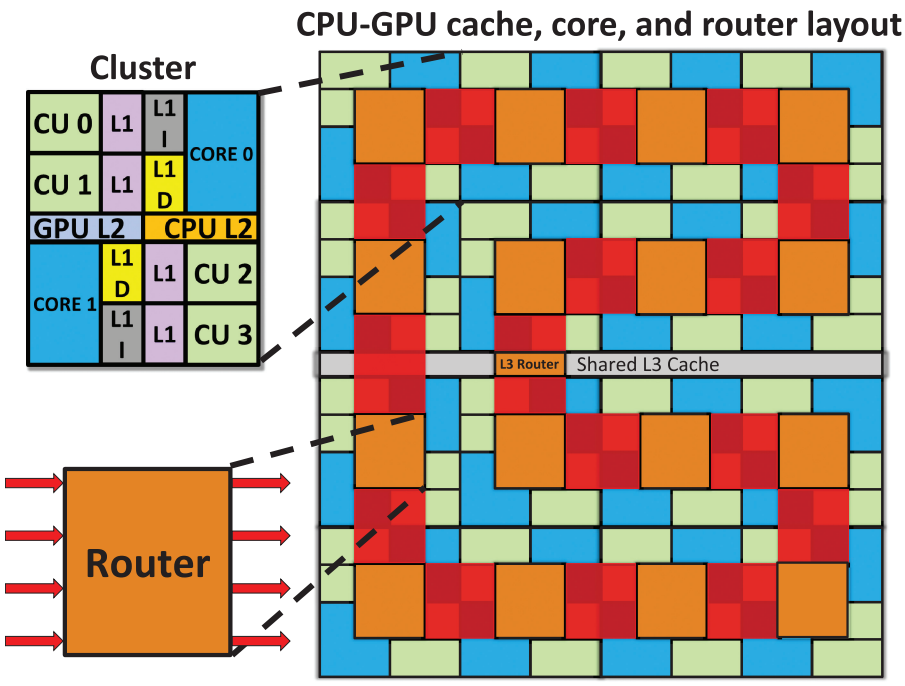


Fig. 1. **Cluster** consists of 2 CPU cores with private L1 data and instruction cache, 4 GPU CUs with private L1 cache, and L2 caches for CPU and GPU all centered around a single router. There are 16 clusters, and all clusters are connected to a L3 router via optical waveguides.

3 PROPOSED ARCHITECTURE: SHARP

In this section, we will discuss the proposed SHARP architecture, router pipeline, and arbitration techniques. Additionally, we will discuss our proposed reconfigurable design that dynamically reallocates bandwidth to improve performance.

3.1 SHARP Architecture: Layout

Figure 1 represents our proposed SHARP architecture, which consists of 32 CPU cores and 64 GPU cores. In SHARP, a *cluster* is made up of two CPU cores, four GPU computational unit (CUs), one router, and corresponding L1/L2 caches. We pick GPUs to be twice the number of CPUs, since the area of a single compute unit is half that of the CPU (see Table 4) [13]. We propose a checkerboard pattern such that each router is directly connected to four CPU and two GPU cores. In this design, CPUs and GPUs contend locally at the router. Under high traffic scenarios for one core type, contention is more manageable, since both cores contend at each router. If the two cores were segregated such that all CPUs were in one half and the GPUs in the other half, then there will be traffic imbalance, as one half would see high traffic, whereas the other half will see low traffic. In our checkerboard design, this scenario is avoided as contention is balanced under all traffic conditions.

Each CPU core has its own private L1 instruction and data caches and each GPU CU has its own private L1 cache. Within each cluster, there is a shared CPU L2 cache and a shared GPU L2 cache. The intra-cluster communication occurs with CPU and GPUs accessing their shared L2 cache. The router at each cluster connects to the shared L3 cache. All 16 routers are organized in a 4×4 grid

and the shared L3 cache is connected using the optical crossbar employing a SWMR approach. The purpose of the L3 cache is to decrease the amount of time needed to communicate between the CPU and GPU sides of the chip. The cache coherence protocol is NMOESI [31] and the L3 cache is split evenly between the CPU and GPU cores. Multi2Sim implements a six-state coherence protocol called NMOESI. This protocol is an extension of the well-known five-state MOESI protocol, where a new non-coherent N state represents the new possible condition of a block being in shared, read-write state. In order for the CPU to communicate with the GPU, the necessary data needs to be copied from the CPU bank of the L3 cache to the GPU bank. For the GPU to communicate with CPU, the necessary data needs to copied from GPU bank to CPU bank. To scale up the design to larger core counts, more optical layers could be added to communicate to different layers of the chip similar to 3D-NoC architecture [22].

We propose a 3D layout that consists of two layers. The first layer consists of the CPU cores, GPU CUs, the caches, and the CMOS routers. The second layer consists of the optical interconnects divided into data and reservation waveguides. We rely on prior work that has shown how electronic and photonic wafers are manufactured separately and then bonded using face-to-face oxide bonding. The proposed thru-oxide bonding (TOVs) have resulted in orders of magnitude in reduction in parasitic capacitance and two orders of magnitude higher density compared to micro-bump flip-chip photonic-electronic integration [28].

3.2 Intra- and Inter-Router Communication

We chose an optical link with reservation assist (R-SWMR) for the purpose of implementing inter-core communication. Under R-SWMR, the transmitting router uses the reservation waveguide to broadcast the signal to the remaining routers connected on the optical link informing of the intended destination. The optical signal can be broadcast using a Y-junction splitter such as a star splitter that removes a percentage of the power such that every node receives the broadcast signal. The splitting ratios can be adjusted such that uneven splitting can be implemented by designing the waveguide carefully. This is crucial, since all nodes receive the same information on the reservation waveguide. Then, only the intended destination listens on the channel while the transmitter sends the data. Figure 2 shows the router architecture for SHARP. Figure 3(a) represents the router pipeline used within the router architecture. When a packet is generated from either the CPU or the GPU core, it is placed in an input buffer and its route is computed (RC). Next, the reservation broadcast (RB) is converted into an optical format (E/O) and coupled to the reservation waveguide. The packet then requests the crossbar in the switch allocation (SA) and traverses the crossbar (BW_S). The E/O conversion occurs by using the electrical drivers to modulate the ring resonators coupling the signal to the optical link. This is when the optical link traversal portion of the router pipeline occurs (OL). At the destination, the reverse process takes place, where the optical signal is filtered and converted into electrical format using a combination of photodetector, TIA and voltage amplifiers (O/E). The packet is written into the buffer (BW_D) where it can be transferred to its intended destination routed via the switch allocation (SA).

Figure 3(b) is a routing example for SHARP. After the RC occurs for each core type the RB is coupled to the reservation waveguide. This process is represented by the yellow line highlighting the reservation waveguide in Figure 3(b). This process notifies which router is supposed to receive the CPU and GPU packets. Each router reads the reservation packet to determine if they need to be listening to the data line in the subsequent cycles. As shown in Figure 3(b), routers R_1 and R_3 ignore the reservation packet, because they are not the destination for either the CPU or GPU data coming from R_0. Correspondingly, the R_1 and R_3 Micro-Ring Resonators (MRRs) are switched off to save energy. It must be noted that MRRs stay switched off unless they are to receive new packets. Therefore, every cycle, MRRs will be tuned off until the reservation waveguide informs

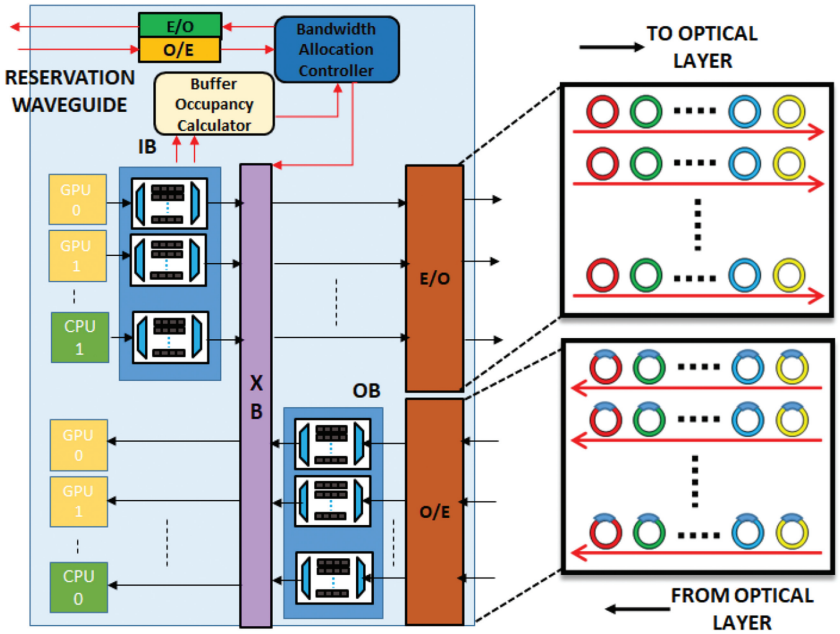


Fig. 2. Electrical to optical and optical to electrical router microarchitecture with reservation waveguide interface: IB = Input Buffer, OB = Output Buffer, O/E = Optical to Electrical conversion, and E/O = Electrical to Optical conversion.

that the packet is being sent the appropriate router. R_2 (receiving the CPU data) and R_{N-1} (receiving the GPU data) will tune the MRRs per the dynamic allocation bits set by the sending router in the reservation packet. This will give R_2 and R_{N-1} the ability to receive the corresponding data from the sending router.

To illustrate with an example, consider Figure 4(a), which shows intra-cluster communication where GPU_1 (cluster 0) connected to router 0 needs to communicate with GPU_0 (cluster 0) also connected to router 0. In Step 1, GPU_1 sends a packet to IB where the RC computes the output port. After RC, SA allocates the switch on the XB in step 2, after which the packet reaches the GPU_0. In Figure 4(b), inter-cluster communication is shown where a packet from GPU_1 (cluster 0) connected to router 0 needs to communicate with GPU_0 (cluster 1) connected to router 1. In Step 1, GPU_1 sends the packet to the IB associated with GPU_1. In Step 2, RC computes the output port and a reservation broadcast (RB) is performed on the reservation waveguide. In this step, the destination router is notified of the incoming packet. In Step 3, the packet arbitrates at the crossbar (BW_S) and reach the E/O where the packet is converted into optical signal. In Step 4, the optical packet is transmitted to the receiver, i.e., router 1. As the reservation packet informed router 1 of the expected packet, only router 1 is tuned to receive the packet. The incoming packet is converted into electrical domain and the packet is written into the OB in router 1. In Step 5, the packet arbitrates for the crossbar at router 1 (BW_D) and the packet reaches the destination GPU_0 in cluster 0.

3.3 Proposed Dynamic Bandwidth

As the GPU has the tendency to flood the network [21], care must be taken while designing the dynamic bandwidth allocation algorithm to prevent the GPU from starving the CPU of the network

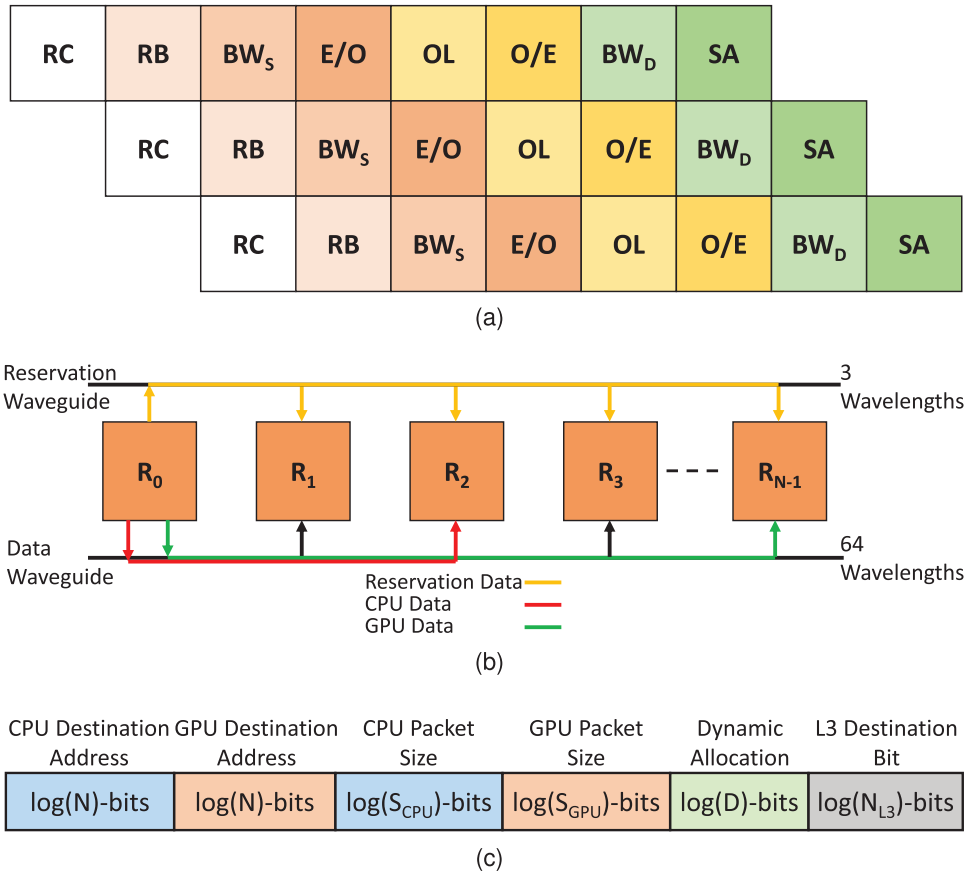


Fig. 3. (a) Router Pipeline. (b) Communication path for CPU-GPU R-SWMR link. (c) Reservation packet.

resources. Our proposed dynamic bandwidth allocation algorithm was designed with the following goals:

- The algorithm should work with minimal hardware additions.
- The algorithm should operate locally within the confines of each router and thereby avoid complex global management.
- The algorithm should prevent the GPU from blocking CPU traffic within the router architecture.
- The algorithm should allow simultaneous transmission of CPU and GPU packets regardless of the packets' destination.

We propose a dynamic variant of the SHARP architecture, which we have labeled *SHARP-Dyn*. A local management of resources was chosen for SHARP-Dyn to mitigate the overhead (e.g., write contention) that is often associated with global bandwidth management techniques [25]. Figure 2 shows the minimal hardware addition to implement the R-SWMR link. When a packet is generated from either the L2 caches or from one of the cores, it is placed in an input buffer. Credit counters track the number of packets occupying the input buffers at each router. The buffer occupancy calculator sums the values of the credit counters for the CPU and GPU packets at the router and sends

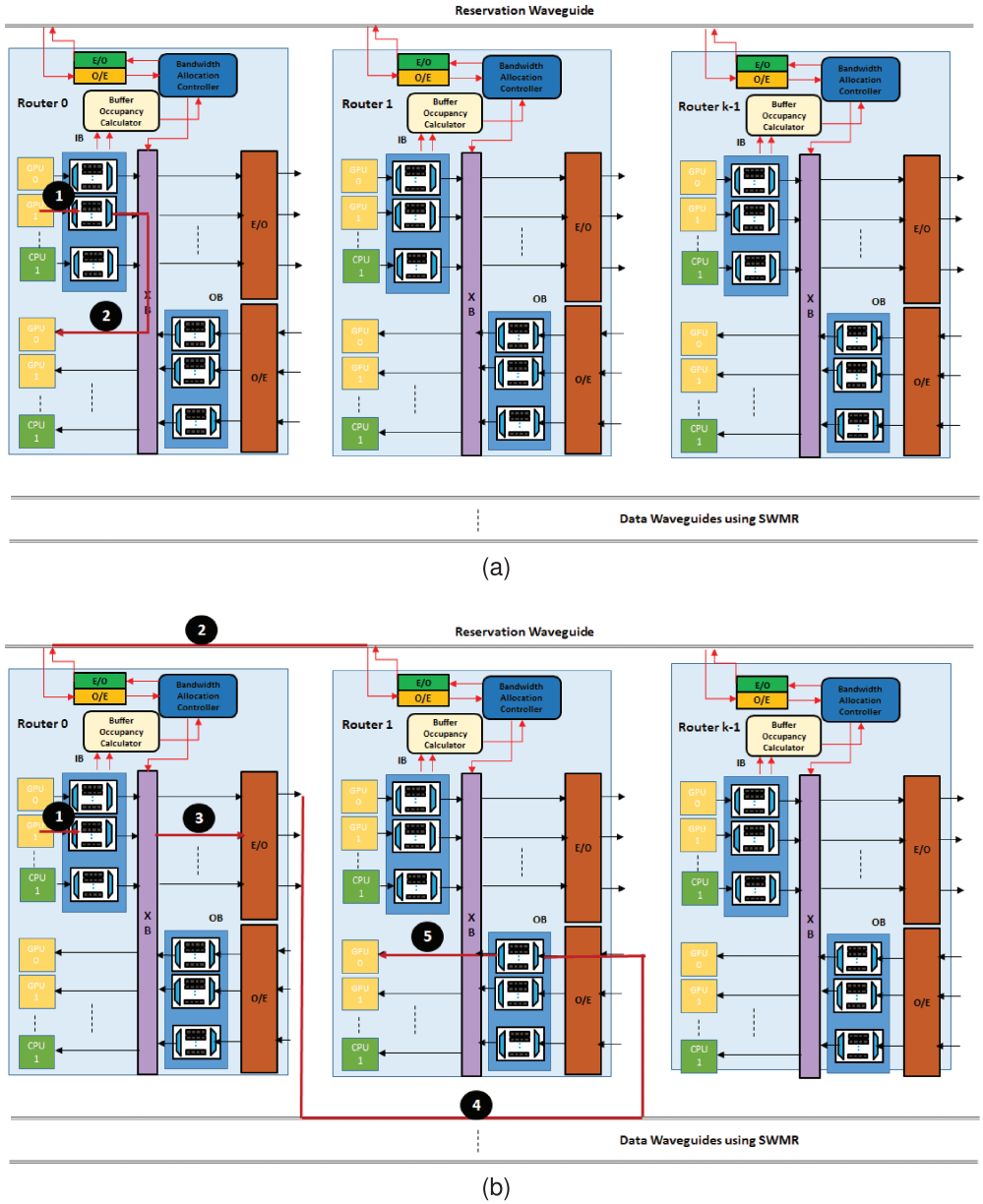


Fig. 4. Panel (a) shows intra-cluster communication and panel (b) shows inter-cluster communication.

the information to the bandwidth allocation controller (BAC). The BAC determines the amount of bandwidth to assign to each core type by using the number of buffer slots occupied. Next, the BAC will generate a reservation packet as seen in Figure 3(c) [25]. Figure 3(c) is the reservation packet used to implement dynamic bandwidth allocation mechanism in SHARP-Dyn. The number of bits in the reservation packet can be calculated with $ResPacket_{\text{size}}$=log(2 × N × S_{CPU} × S_{GPU} × D × N_{L3}), where N is the number of non-L3 routers in the network, S_{CPU} is the number of different CPU

Table 1. Dynamic Bandwidth Allocation Algorithm

Step 0:	For each individual router R_ω for routers R_0 through $R_{max\text{-}1}$ complete steps 1 through 5
Step 1:	Calculate $\beta_{ocup\text{-}CPU\omega}$
Step 2:	Calculate $\beta_{ocup\text{-}GPU\omega}$
Step 3:	Determine the amount of bandwidth to be allocated to the CPU and GPU core types:
	If $\beta_{GPU} = 0$ and $\beta_{CPU} > 0$
	$GPU_{Bandwidth} = 0\%\ Bandwidth$
	$CPU_{Bandwidth} = 100\%\ Bandwidth$
	Else if $\beta_{CPU} = 0$ and $\beta_{GPU} > 0$
	$GPU_{Bandwidth} = 100\%\ Bandwidth$
	$CPU_{Bandwidth} = 0\%\ Bandwidth$
	Else if $\beta_{GPU} < \beta_{GPU\text{-}UpperBound}$
	$GPU_{Bandwidth} = 25\%\ Bandwidth$
	$CPU_{Bandwidth} = 75\%\ Bandwidth$
	Else if $\beta_{CPU} < \beta_{CPU\text{-}UpperBound}$
	$GPU_{Bandwidth} = 75\%\ Bandwidth$
	$CPU_{Bandwidth} = 25\%\ Bandwidth$
	Else
	$GPU_{Bandwidth} = 50\%\ Bandwidth$
	$CPU_{Bandwidth} = 50\%\ Bandwidth$
Step 4:	Send reservation packet via reservation-assisted SWMR link
Step 5:	Transmit data using specified wavelengths on the first-come, first-serve basis

packet types (i.e., request and response) that can be sent through the network, S_{GPU} is the number of different GPU packet types that can be sent through the network, D is the number of different dynamic allocation possibilities for the data being sent (D=5 for the proposed algorithm), and N_{L3} is the number of L3 routers in the network. The number of wavelengths needed for the reservation waveguide can be determined using the $ResPacket_{size}$, optical data rate, network frequency, and the number of routers.

The proposed algorithm in Table 1 is executed by every router during every cycle in the network. In the first step of the algorithm, the router calculates the buffer occupancy. This occupancy calculation is done for each input buffer $\beta_{Ocup\text{-}i}$ that is connected to a core or L2 cache. To determine the amount of bandwidth to assign to each core type $\beta_{ocup\text{-}CPU\omega}$ and $\beta_{ocup\text{-}GPU\omega}$ has to be calculated by summing the buffer occupancy for each core type at each router. This can be seen in the following formulas:

$$\beta_{ocup\text{-}CPU\omega} = \frac{\sum_{i=0}^{k-1} Buf_i \times a_i}{Buf max_{CPU}}, \tag{1}$$

$$\beta_{ocup\text{-}GPU\omega} = \frac{\sum_{i=0}^{j-1} Buf_i \times a_i}{Buf max_{GPU}}, \tag{2}$$

$$Buf_\omega = Buf max_{CPU} + Buf max_{GPU}, \tag{3}$$

where Buf_i is the individual buffer either associated with CPU or GPU, $k = Buf max_{\text{CPU}}$=maximum number of buffer slots for CPU, $j = Buf max_{\text{GPU}}$=maximum number of buffer slots for GPU, Buf_ω is the total number of buffers at each router Rω, $0 \leq \omega \leq R_{\text{max}-1}$, and a_i is 1 if the buffer slot is occupied. $\beta_{\text{CPU-UpperBound}}$ and $\beta_{\text{GPU-UpperBound}}$ are thresholds that are used to make the decisions within the algorithm. Utilizing a brute force method, the optimal $\beta_{\text{CPU-UpperBound}}$ and $\beta_{\text{GPU-UpperBound}}$ are determined experimentally on a separate set of benchmarks than the ones used in the results section of this article. The optimal $\beta_{\text{CPU-UpperBound}}$ was determined to be 16% of the total CPU input buffer space while the optimal $\beta_{\text{GPU-UpperBound}}$ was determined to be 6% of the total GPU input buffer space. Using the calculated values for $\beta_{\text{ocup-CPU}}$ and $\beta_{\text{ocup-CPU}}$ in comparison with $\beta_{\text{CPU-UpperBound}}$ and $\beta_{\text{GPU-UpperBound}}$, the bandwidth is assigned to each core type when the reservation packet is created. Due to the temporal sensitivity of the CPU, precedence is given over the GPU by considering it first for the 75% bandwidth allocation in Step 5 of Table 1. After sending and receiving the reservation packet, the corresponding routers will tune the MRRs to receive the data packets. In terms of bandwidth allocation implementation, the bandwidth is divided into different wavelengths, i.e., when allocating 50% bandwidth between CPU and GPU, 50% of the wavelengths are allocated to CPU corresponding to laser array (LA_{0-15}, LA_{16-31}) and the remaining 50% of the bandwidth will be allocated to GPU corresponding to laser array (LA_{32-47}, and LA_{48-63}), since we have a total of 64 wavelengths.

3.4 Variations of SHARP Architecture

In this subsection, we describe a few variations of the SHARP architecture that will be compared against the proposed SHARP-Dyn architecture. The first architecture is called *SHARP Core Segregation* (SHARP-CoSeg), which uses a photonic crossbar and segregates the CPU and GPU cores to two halves of the chip as shown Figure 5. By segregating CPU and GPU cores, we avoid all interaction between the two core types (L3 cache router and main memory interactions are excluded). The second architecture is called *SHARP first-come-first-serve* (SHARP-FCFS) and has a similar layout as SHARP-Dyn (see Figure 1). The key difference between SHARP-Dyn and SHARP-FCFS is that SHARP-FCFS has no dynamic bandwidth allocation. Furthermore, in SHARP-FCFS, the packets are sent on a first-come, first-serve basis using 100% of the link bandwidth regardless of which core injects the packet. The third architecture is called *SHARP Bandwidth Split* (SHARP-BanSp) and has a similar layout as SHARP-Dyn (see Figure 1). SHARP-BanSp divides the link bandwidth evenly between the CPU and GPU cores at each router regardless of traffic flow or buffer utilization.

All four architectures utilize R-SWMR links [25]. The reservation packets in SHARP-CoSeg and SHARP-FCFS are similar to Figure 3(c). This packet consists of a single destination address, single packet size, and L3 destination bit, which reduces the reservation packet by almost half. For the SHARP-FCFS architecture, the CPU and GPU data will not be sent on the same cycle, eliminating the need for two destination addresses and packet sizes in the reservation packet. For the SHARP-CoSeg architecture the two core types do not reside at the same router, eliminating the need for two destination addresses and packet sizes in the reservation packet.

4 PERFORMANCE EVALUATION

In this section, we evaluate the performance of four variations (SHARP-Dyn, SHARP-FCFS, SHARP-CoSeg, SHARP-BanSp) of our proposed SHARP architecture. We also compare SHARP and its variations against CMESH, PNoC [29], and Firefly [25] architectures for real-traffic traces. With each architecture, we run a multitude of CPU and GPU simulations. Each simulation combines one CPU and one GPU benchmark to make a benchmark pair. Each benchmark pair will be evaluated by varying both the number of wavelengths per waveguide as well as the injection rate to gauge the performance of the networks. Last, we evaluate and compare the area and energy

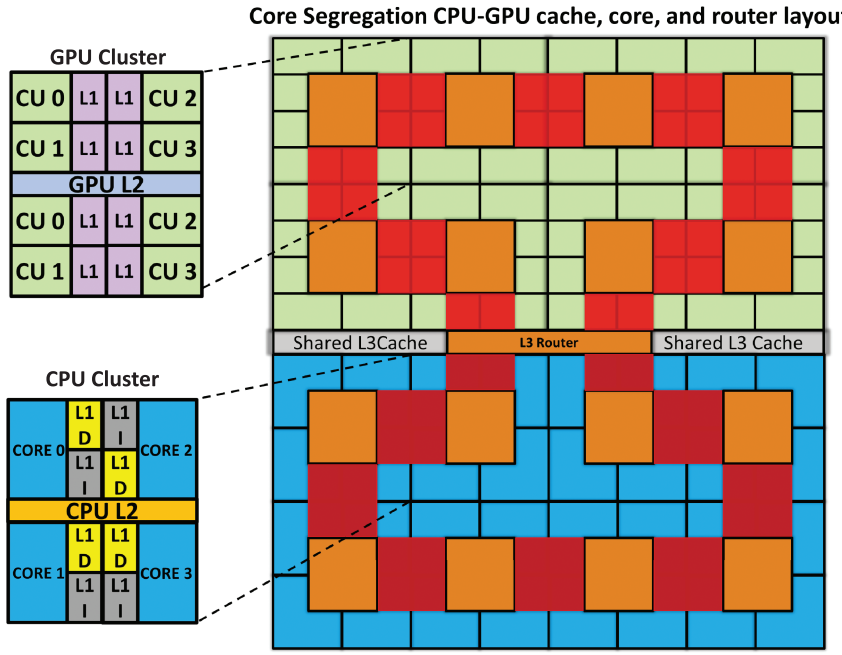


Fig. 5. SHARP-CoSeg chip layout. The CPU and GPU cores and their resources are separated between two halves of the chip.

consumption and compute the Energy Delay Product (EDP) per packet of each network. As the variation in area requirement for SHARP-Dyn and the other architectures only change within the Router and Optical Interface Layer, we have omitted this plot, but we present the data in Table 4.

4.1 Simulation Setup

All real network simulation traces were collected using Multi2Sim [31]. Multi2Sim is a cycle-accurate simulation framework that captures every network transaction and its effect on the CPU and GPU pipelines, cache interactions, and router pipelines. Each network trace was collected by running a single benchmark on Multi2Sim for a total of four CPU benchmarks and four GPU benchmarks. The CPU benchmarks where chosen from the PARSEC 2.1 benchmark suite [5] and the SPLASH2 benchmark suite [35]. The four CPU benchmarks were chosen for their variation in parallelization, size of problem set, and data usage [5]. The GPU benchmarks are chosen from OpenCL SDK benchmarks provided by AMD through Reference [31]. The goal of the simulations is to see the impact on network throughput and latency for various combinations of CPU and GPU workloads. Each line in the trace is made up of a source, destination, packet type (request or response), and a network cycle number corresponding to when the packet was sent into the network. One CPU and one GPU trace is combined using the network cycle number given to each packet by Multi2Sim. The simulation specifications can be seen in Table 2. While GPUs have been used to accelerate CPU applications, in this work, we run CPU and GPU workloads separately on CPU and GPU clusters and evaluate their interaction on the network [13, 18]. This is identical to References [13, 18], where both CPU and GPU traffic run independently, and it is the interference and resource sharing that is analyzed. For example, CPU is latency sensitive and GPU is bandwidth sensitive and both these papers evaluate the impact of running them together on the same

Table 2. Architecture Specifications

CPU		GPU	
Cores	32	Computation Units	64
Threads per Core	4	Frequency (GHz)	2
Frequency (GHz)	4	L1 Cache Size (kB)	64
L1 Instruction Cache Size (kB)	32	L2 Cache Size (kB)	512
L1 Data Cache Size (kB)	64		
L2 Cache Size (kB)	256		

Shared Components	
Network Frequency (GHz)	2
L3 Cache Size (MB)	8
Main Memory Size (GB)	16

Table 3. Benchmark Information

Core Type	Abbreviation	Benchmark Name
CPU	FA	Fluid Animate
	fmm	Fast Multipole Method
	Rad	Radiosity
	x264	x264
GPU	DCT	Discrete Cosine Transforms
	Dwrt	One-dimensional Haar Wavelet Transform
	QRS	Quasi Random Sequence
	Reduc	Reduction

network. Similarly, we have analyzed the impact of running multiple CPU and GPU benchmarks on photonic NoCs. In this work, we assume that the network and router microarchitecture operates at 2Ghz, similar to prior work [15].

The traces where combined with the intent of mimicking high workloads for server applications and demonstrating the outcome when there is heavy interaction between the two core types. Abbreviations for each benchmark can be seen in Table 3. Figure 6 is the packet percentage between each core type for the Real Traffic Benchmark Pairs. These benchmark pairs were chosen for the wide variation of packet densities with respect to the two core types. Nine Synthetic Traffic benchmark pairs were used in this article to demonstrate how each architecture performs under controlled traffic patterns. The CPU's synthetic traffic timing was constant throughout each benchmark while the GPU's synthetic traffic timing was varied to mimic the bursty nature of the GPU core type. Each benchmark pair was run through a cycle-accurate network simulator to compare the throughput and latency of all architectures. Using traffic traces with a cycle-accurate network simulator allowed us to isolate the network interaction and gauge the impact of the two traffic types on the network. Each simulation was run with 64, 32, and 16 wavelengths with each wavelength operating at a 16Gbps data rate. We consider different numbers of wavelengths to test different scenarios. To further gauge the interaction between the two core types, the simulations were run with increased injection rates to determine how each architecture responds under higher load. The injection rate variation is based on the trace packet's injection time. Therefore, we inject packets at a higher rate and this translates into 2, 4, 8, 16, and 32 times when compared to the trace collection. We refer to this as the Workload Injection Rate. It must be noted that when running at

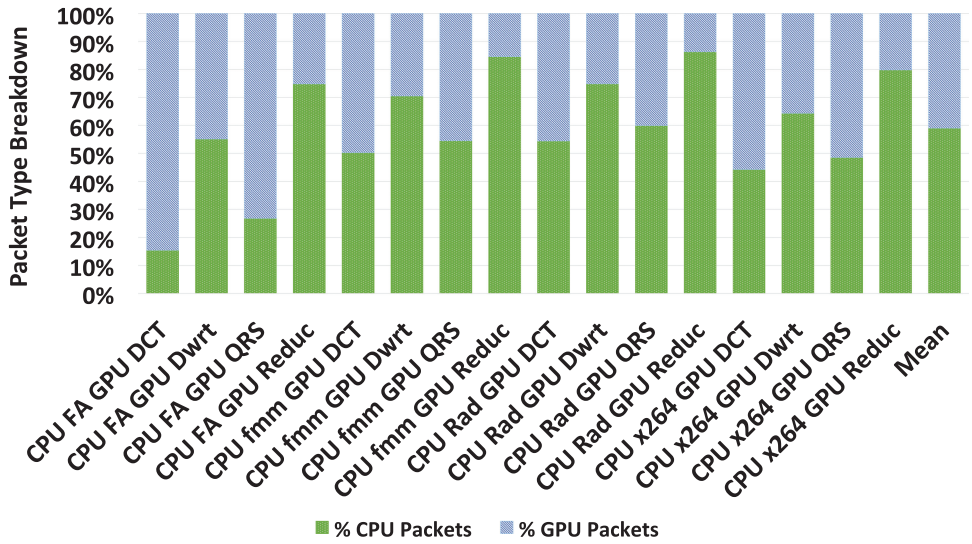


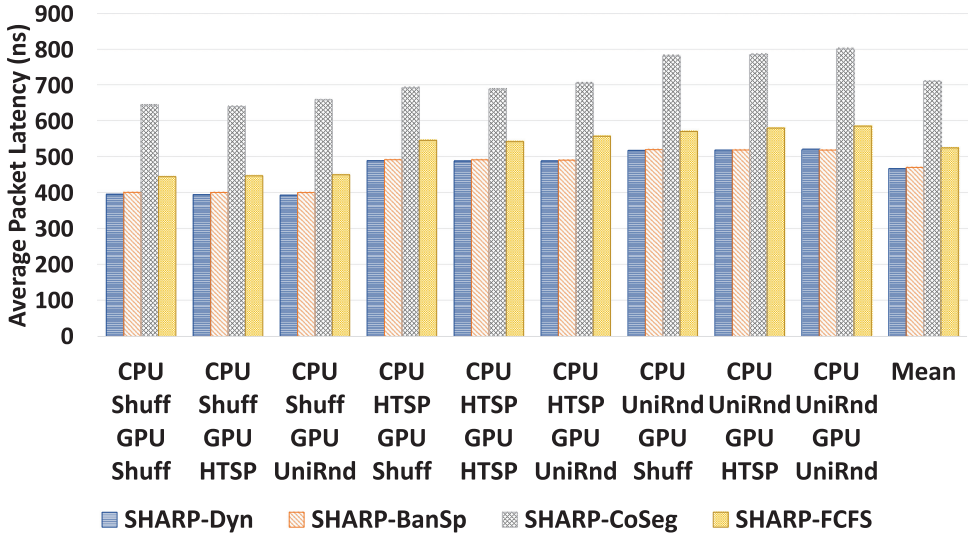
Fig. 6. CPU-GPU packet breakdown for each traffic trace.

higher network load, the packet latency does not become infinity, since all packets will reach their destination. Therefore, with increasing the workload injection rate, we are evaluating the impact on network latency and throughput when the traffic intensity increases.

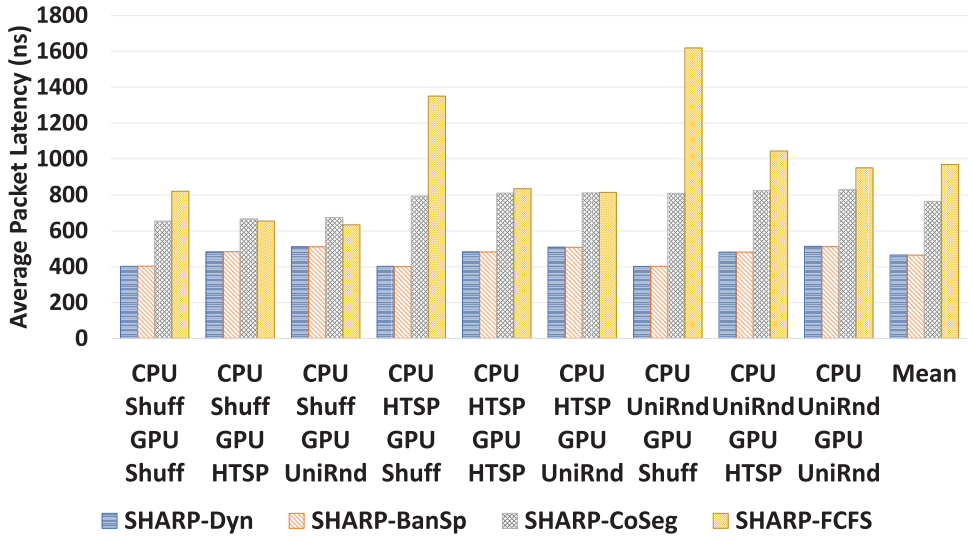
The router energy calculations were acquired through DSENT 0.91 [30]. The optical power calculations were attained using the values from Table 5 [22, 38]. The laser power results, along with the ring heating power [22], were used to calculate the energy per bit for the photonic interconnects. SHARP-Dyn's dynamic bandwidth allocation requires 8-bit addition to determine how to allocate bandwidth between the cores. The energy consumption for the 8-bit addition is 0.03pJ per add [9]. The energy per bit results for SHARP-Dyn's dynamic allocation mechanism is not displayed, because the mean energy consumed for this process accounts for 0.824% of the total energy per bit for the 64-wavelength simulations. The values in Figure 14 show the average energy per bit to operate the network connections and the routers. The EDP per packet represent the product of the average energy consumed by each packet with the time each packet spends in the network.

4.2 Simulation Results

The simulation results are broken up into four categories: throughput, latency, energy per bit, and energy-delay product (EDP). The throughput is shown in bytes per cycle to have a fair comparison of the network throughput without the data being skewed by the different packet sizes. Latency represents the amount of time the packet spends within the network. The energy per bit demonstrates the amount of energy used by the network with reference to the throughput of the network. The EDP is used to compare the average energy spent per packet with the average time each packet spends within the network. The four optical architecture will be compared to a baseline concentrated MESH (CMESH) design. The CPU and GPU core types will be segregated to separate halves of the chip with the interaction between the two core type residing at the L3 cache. The channel width for the CMESH was set at 256 bits to balance the throughput and energy consumption when compared to the photonic architectures. Due to the multi-hop nature and the increased latency per packet of the CMESH architecture, the latency numbers for the CMESH were not included in the latency plots. This was done to make the plot easier to read.



(a)



(b)

Fig. 7. Synthetic Traffic Latency. (a) Average time a CPU packet spends in the network for 64 wavelengths. (b) Average time a GPU packet spends in the network for 64 wavelengths. Shuff=Perfect Shuffle, HTSP=Random Traffic with a Hotspot, and UniRnd=Uniform Random.

4.2.1 Synthetic Traffic Results. Figures 8 and 7 represent the throughput and latency for synthetically generated network traffic. Three traffic patterns chosen are Perfect Shuffle (Shuff), Uniform Random (UniRnd), and Random Traffic with a Hotspot (HTSP). Synthetic traffic demonstrates the network and dynamic allocation's performance under controlled traffic patterns. The CPU injection rate is set to force the network into saturation, while the GPU injection time varies randomly within the simulation. The GPU synthetic traffic is designed to have periods of time with

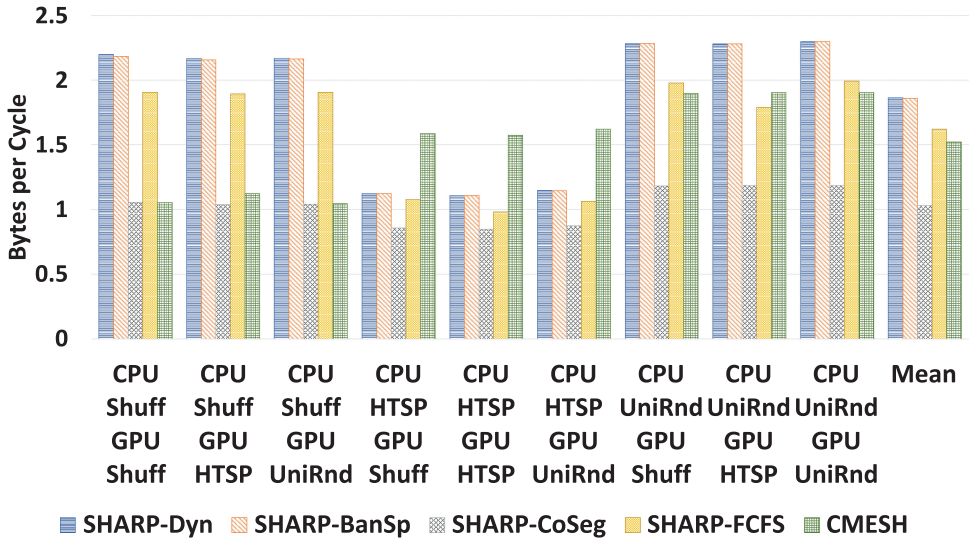


Fig. 8. Synthetic traffic throughput: Shuff=Perfect Shuffle, HTSP=Random Traffic with a Hotspot, and UniRnd=Uniform Random

a high injection rate. In contrast, periods of time will occur when the GPU cores will not inject any packets. This demonstrates how SHARP-Dyn's dynamic allocation mechanism handles the bursty nature of the GPU traffic while the network is at saturation from the CPU traffic. SHARP-Dyn demonstrates a 22.3% throughput increase over CMESH on average for all synthetic benchmark pairs with a maximum increase of 104%. The mean throughput improvement when comparing SHARP-Dyn to SHARP-CoSeg is 80.9%, while SHARP-Dyn demonstrates a 14.9% improvement over SHARP-FCFS. Additionally, SHARP-FCFS exhibits a 57.4% throughput increase over SHARP-CoSeg. This demonstrates the potential benefit of clustering the two core types over alienating them to separate halves of the chip. SHARP-Dyn's mean throughput improvement over SHARP-BanSp is 0.19%. The minimal throughput improvement is due to the network running at saturation, which in turn creates high network contention between the two core types. For hotspot traffic, CMESH performed better for certain cases due to path diversity as mesh architecture provides better resource utilization. Similarly, for perfect shuffle, the pattern creates worst-case scenarios, due to which the performance drops. When there is high network contention and increased buffer occupancy between the CPU and GPU, SHARP-Dyn more often resides within the fifth bandwidth allocation state from Step 5 in Table 1. When comparing the latencies in Figure 7, SHARP-Dyn has a 34.2% CPU latency decrease over SHARP-CoSeg. The SHARP-FCFS's CPU latency improvement over SHARP-CoSeg demonstrates that the checkerboard pattern spreads the network load across all the routers and decreases the amount of time a packet spends within the network. The high injection rate of the synthetic traffic causes the SHARP-CoSeg architecture to increase the contention at the CPU routers, which increases the delay of the CPU packets. Couple the checkerboard layout with SHARP-Dyn's dynamic allocation mechanism, SHARP-Dyn demonstrates an 11.0% CPU latency improvement over SHARP-FCFS.

4.2.2 Real Traffic Results. Figure 9 shows the throughput for various architectures under different workloads using 64 wavelengths per waveguide. The work load distribution per core type can be seen in Figure 6. The means in Figures 10 and 9 are an average for all 16 real-traffic simulations.

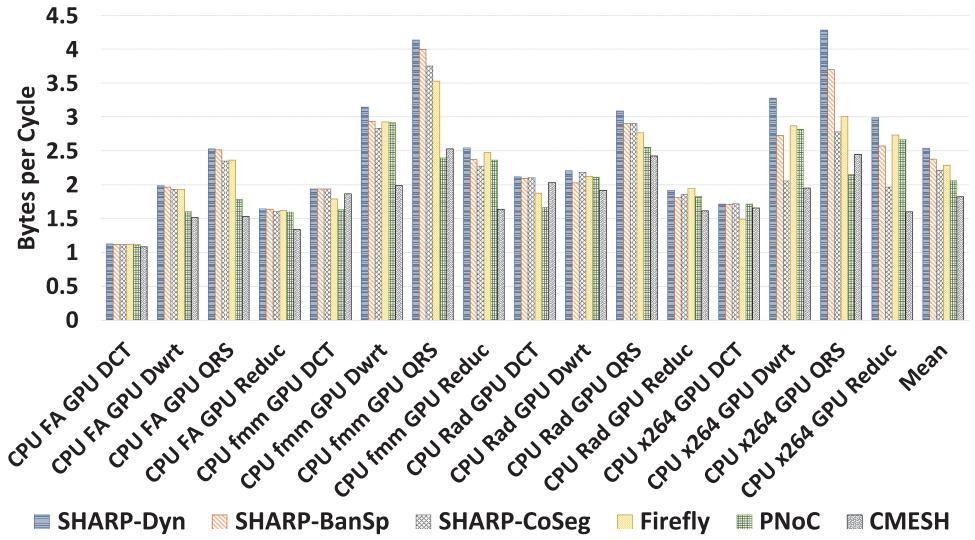


Fig. 9. Real traffic throughput results for 64 wavelengths.

It can be seen in Figure 9 that SHARP-Dyn outperforms the remaining architectures. SHARP-Dyn outperforms CMESH by 34% on average. SHARP-Dyn's mean improvement in throughput over SHARP-BanSp, SHARP-CoSeg, and Firefly architectures are 6.9%, 14.9%, and 11.2%, respectively. While Firefly architecture implements R-SWMR, it allocates bandwidth based on demand or first come, first serve, and therefore it does not take load into consideration. However, SHARP-Dyn takes CPU and GPU workloads into consideration when allocating bandwidths, and therefore it improves performance considerably. PNoC architecture as proposed reallocates bandwidth on the end of a task between the CPU and GPU. However, SHARP-Dyn reallocates bandwidth on the network load and therefore performs considerably better. For example, SHARP-Dyn outperforms PNoC by more than 20% in terms of throughput. Figure 10(a) represents the CPU network latency for all four photonic architectures. SHARP-Dyn's average CPU latency improvement is 76.6% and GPU latency improvement is 70.4% over CMESH. When comparing SHARP-Dyn's architecture to the three other photonic architectures, SHARP-Dyn's has a maximum CPU latency improvement of 58.4% for the CPU FA GPU DCT real-traffic benchmark pair. The GPU latency for SHARP-CoSeg demonstrates an improvement over SHARP-Dyn, but this is to the detriment of the latency sensitive CPU cores as see in Figure 10.

Figure 11 is the mean throughput for each architecture at 64 wavelengths as the injection rate is increased. SHARP-Dyn demonstrates a 7.6% throughput increase over SHARP-BanSp when the Workload Injection Rate is at 32 times the initial injection rate. The minimum average throughput gain, at 5.9%, that SHARP-Dyn demonstrates over SHARP-BanSp occurs when the Workload Injection Rate is increased by four times. The increased contention has an adverse affect on SHARP-CoSeg and SHARP-FCFS. With a Workload Injection Rate of 16, SHARP-Dyn has an increase of throughput over SHARP-CoSeg and SHARP-FCFS of 16.9% and 19.5%, respectively. The increase in injection rate has a minimal effect on the CMESH architecture with an increase of less than 1% from the Workload Injection Rate of 1 to 32. When the network is at high contention with the Workload Injection Rate at 32, SHARP-Dyn demonstrates a 45.6% throughout increase over CMESH. SHARP-Dyn improves the throughput in resource constrained environments, since it dynamically allocates bandwidth. Once injection rate has been increased, the maximum performance benefits can be achieved, and therefore there is no increase in throughput. Figure 12

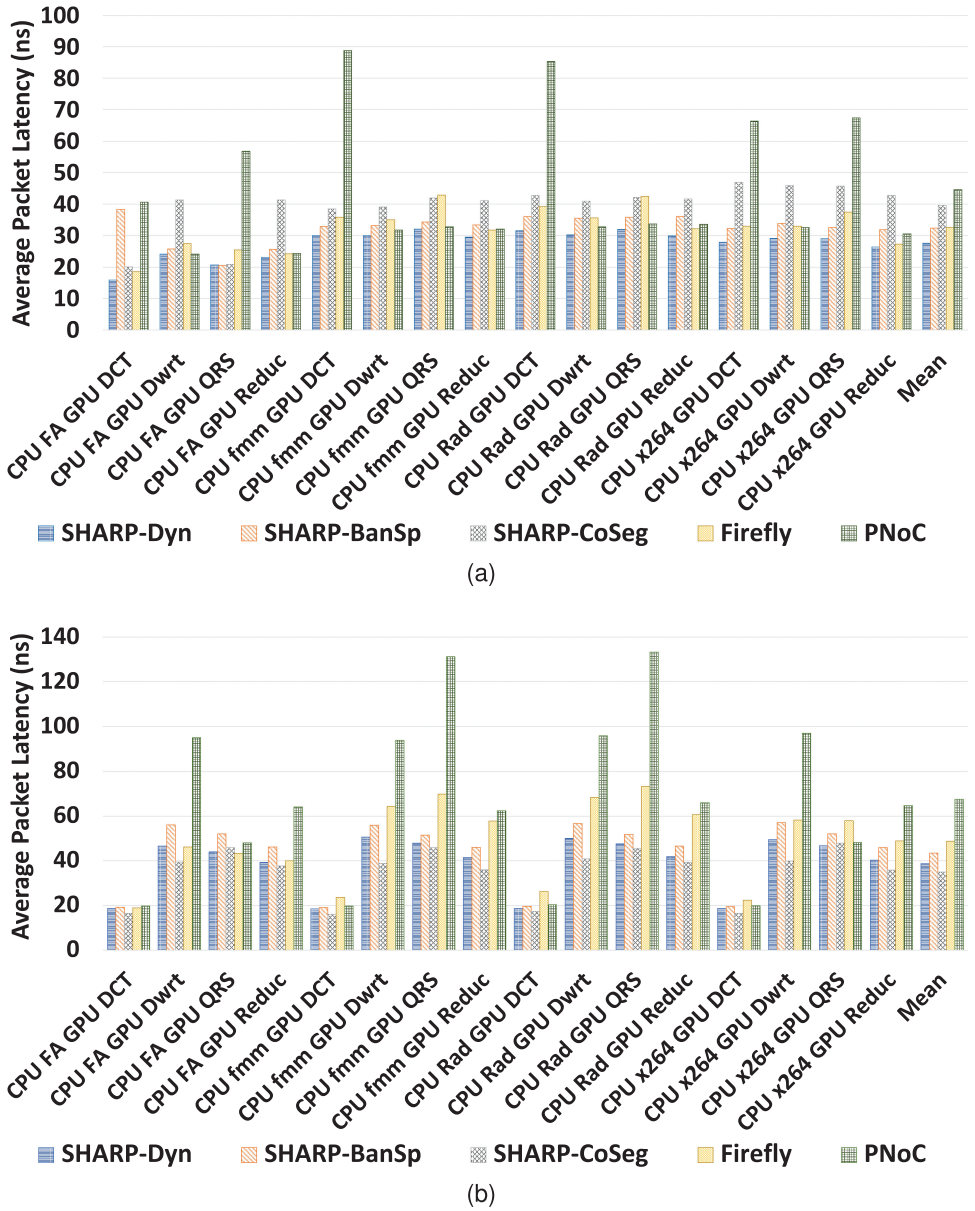


Fig. 10. Real Traffic Latency. (a) Average time a CPU packet spends in the network for 64 wavelengths. (b) Average time a GPU packet spends in the network for 64 wavelengths.

represents the percentage increase in throughput for all 16 real-traffic benchmark pairs when the bandwidth is set to 64, 32, and 16 wavelengths compared against an increase in the Workload Injection Rate. As the contention increases, the SHARP-Dyn architecture spends more time in the 50/50 bandwidth state from Table 1. As demonstrated in Figure 12, the network contention increases due to the bandwidth decrease; SHARP-Dyn's throughput improvement decreases when compared to SHARP-BanSp. This demonstrates the benefit of the dynamic bandwidth allocation at

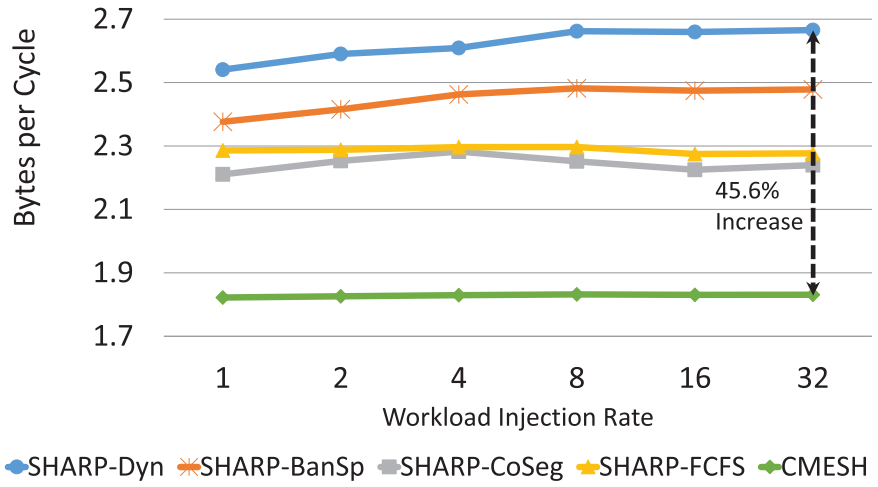


Fig. 11. Average throughput with an increase in injection rate for 64 wavelengths.

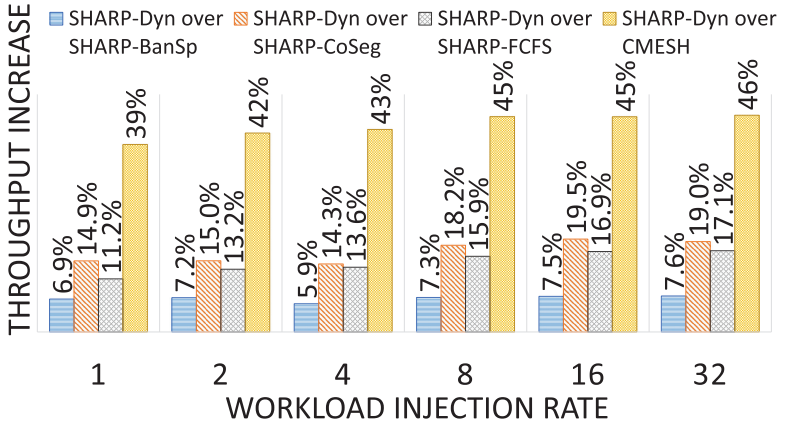


Fig. 12. Percentage of throughput increase for 64 wavelengths when comparing SHARP-Dyn to SHARP-BanSp, SHARP-CoSeg, and SHARP-FCFS on real traffic.

lower contention and SHARP-Dyn's behavior when the network is at higher contention. Figure 13 represents the percentage throughput increase for SHARP-Dyn over the other four architectures. As expected, the gap between SHARP-Dyn and SHARP-BanSp decreases as the contention in the network increases. The percentage throughput increase from 64 to 32 wavelengths demonstrates SHARP-Dyn's ability to handel the network load.

4.2.3 Area and Power Results for Real Traffic. Table 4 shows the area overhead for the proposed SHARP-Dyn architecture, which was calculated using McPAT and GPUWatch [19, 20]. The router and link energy per bit calculations were determined using DSENT 0.91 [30] for all five architectures. The optical power calculations were attained using 1dB for modulator insertion, 1dB/cm waveguide loss, 1dB for coupling loss, 0.1dB for photo detection, 500μW/ring for ring modulation, 26μW/ring for ring heating and 10% laser efficiency as seen in Table 5. The laser power results, along with the ring heating power [22, 23], were used to calculate the energy per bit for the

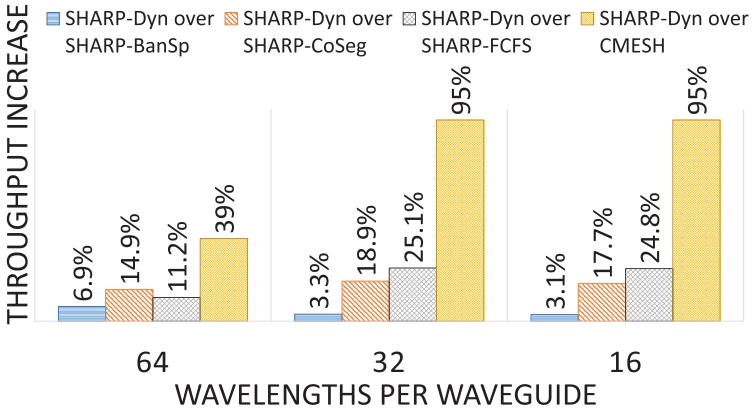


Fig. 13. Percentage of throughput increase for 64, 32, and 16 wavelengths when comparing SHARP-Dyn to SHARP-BanSp, SHARP-CoSeg, and SHARP-FCFS on real traffic.

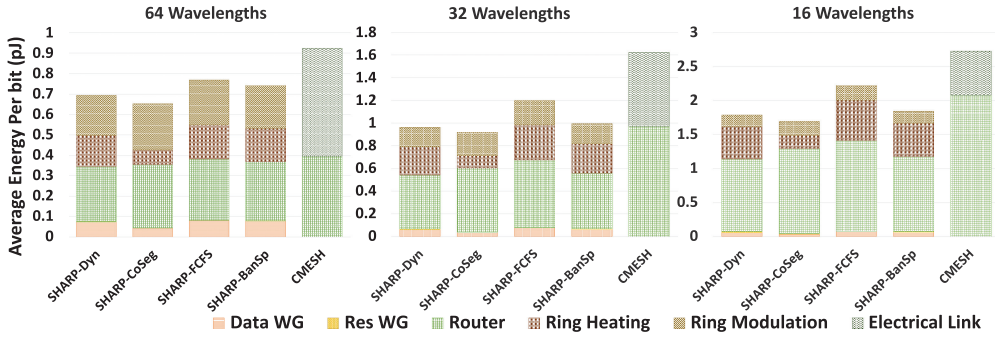


Fig. 14. Average energy per bit for different wavelengths (64, 32, 16).

Table 4. Area for SHARP-Dyn

Photonic and Electronic Component	Area
Cluster	25 mm^2
L2 Cache per Cluster	2.1 mm^2
Optical Components (MRRs and Waveguides)	24.4 mm^2
Waveguide Width [30]	5.28 μm
MRR Diameter [34]	3.3μm
L3 Cache	8.5 mm^2
Router	0.342 mm^2
Dynamic Allocation	0.576 mm^2

photonic interconnect. The values in Figure 14 show the average energy per bit expended to operate the network connections and the routers. Each photonic architecture's energy per bit cost is shown with bandwidth constraints of 16, 32, and 64 wavelengths per waveguide. The CMESH bandwidth constraints are relative to the number of wavelengths starting with 256 bit electrical connections. The energy per bit decrease from CMESH to SHARP-Dyn is 25% while the EDP per packet has decreased 80%. Additionally, the average energy per bit in Figure 14 shows a 6.4%

Table 5. Loss and Power Values for Optical Energy Calculations

Component	Value	Unit
Modulator Insertion	1	dB
Waveguide	1.0	dB/cm
Coupler	1	dB
Splitter	0.2	dB
Filter Through	1.00e-3	dB
Filter Drop	1.5	dB
Photodetector	0.1	dB
Receiver Sensitivity	−15	dBm
Ring Heating	26	μW/ring
Ring Modulating	500	μW/ring

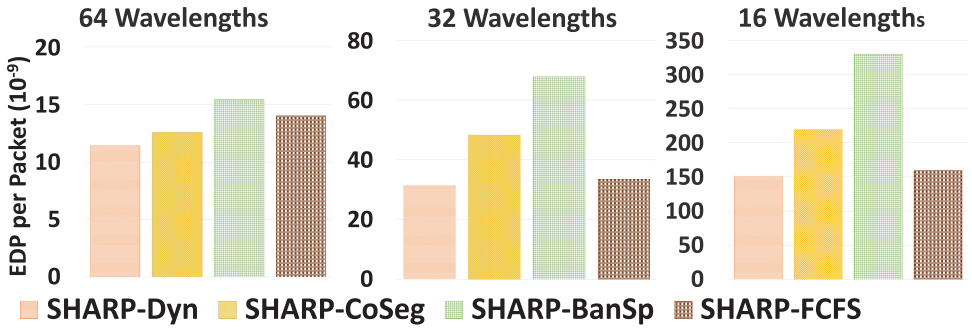


Fig. 15. Energy delay product per packet for different wavelengths (64, 32, 16).

decrease from the SHARP-Dyn to the SHARP-CoSeg network for 64 wavelengths while demonstrating a 9.1% increase in EDP per packet. The energy per bit decrease is due to the decrease in hardware needed for SHARP-CoSeg. The segregated core types communicate to half of the chip, which uses less laser power then the other photonic architectures. This result demonstrates that the decrease in energy spent on a packet doesn't necessarily decrease the amount of time a packet spends within the network. Furthermore, Figure 14 shows a 6.5% energy per bit decrease from SHARP-BanSp to SHARP-Dyn and a 9.7% energy per bit decrease from SHARP-FCFS to SHARP-Dyn for 64 wavelengths. When the bandwidth is constrained from 64 to 32 wavelengths, SHARP-Dyn shows a 19.7% and 3.2% energy per bit decrease when compared to SHARP-BanSp and SHARP-FCFS. SHARP-Dyn demonstrates a 40.7% and 34.4% decrease in energy per bit when compared to CMESH, respectively, for 32 and 16 wavelengths. Figure 15 shows the energy-delay product (EDP) for 64, 32, and 16 wavelenths. SHARP-Dyn demonstrates a 91.9% and 88.8% decrease in EDP per packet when compared to CMESH, respectively, for 32 and 16 wavelengths.

5 CONCLUSIONS

In this work, we have demonstrated the advantage of utilizing photonic interconnect for a heterogeneous architecture while implementing dynamic bandwidth allocation using the R-SWMR approach. SHARP-Dyn and SHARP-FCFS demonstrate the advantages of clustering two core types around one router. On synthetic traffic, SHARP-FCFS exhibits a 57.4% increase in throughput over SHARP-CoSeg. Meanwhile, SHARP-Dyn demonstrates an 80.9% throughput increase on synthetic traffic. The high injection rate of the synthetic traffic creates contention for the resources at each

router. The results on the synthetic benchmark pairs show that clustering two core types at one router can spread the traffic across the network, alleviating the contention at each router. SHARP-Dyn increases the throughput on real traffic while decreasing the energy per bit cost over SHARP-FCFS, SHARP-BanSp, SHARP-CoSeg, and CMESH for varying link bandwidths. The real-traffic simulation results produced a 6.9%–14.9% increased throughput for SHARP-Dyn over the other three photonic architectures at 64 wavelengths. Additionally, SHARP-Dyn demonstrates a 34% performance improvement over the baseline CMESH while decreasing the energy per bit by 25%. SHARP-Dyn has shown a 6.4% and 9.7% energy per bit improvement over SHARP-BanSp and SHARP-FCFS, respectively, while the bandwidth is set to 64 wavelengths.

REFERENCES

[1] AMD. 2015a. AMD Accelerated Parallel Processing (APP) Software Development Kit (SDK). Retrieved from http://developer.amd.com/sdks/amdappsdk/.

[2] AMD. 2015b. Carrizo. Retrieved from http://www.amd.com/en-us/who-we-are/corporate-information/events/isscc.

[3] C. Batten, A. Joshi, J. Orcutt, A. Khilo, B. Moss, C. Holzwarth, M. Popovic, H. Li, H. Smith, J. Hoyt, F. Kartner, R. Ram, V. Stojanovic, and K. Asanovic. 2008. Building manycore processor-to-DRAM networks with monolithic silicon photonics. In *Proceedings of the 16th IEEE Symposium on High Performance Interconnects*. 21–30.

[4] S. Le Beux, H. Li, I. O'Connor, K. Cheshmi, X. Liu, J. Trajkovic, and G. Nicolescu. 2014. Chameleon: Channel efficient optical network-on-chip. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'14)*. 1–6. DOI: http://dx.doi.org/10.7873/DATE.2014.317

[5] C. Bienia and K. Li. 2009. PARSEC 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*.

[6] S. Borkar. 2013. Exascale computing—A fact or a fiction? In *Proceedings of the IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS'13)*. 3–3.

[7] W. Choi, K. Duraisamy, R. G. Kim, J. R. Doppa, P. P. Pande, R. Marculescu, and D. Marculescu. 2016. Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In *Proceedings of the International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES'16)*. 1–10. DOI: http://dx.doi.org/10.1145/2968455.2968510

[8] E. Fusella and A. Cilardo. 2017. H2ONoC: A hybrid optical electronic NoC based on hybrid topology. *IEEE Trans. Very Large Scale Integr. Syst.* 25, 1 (Jan 2017), 330–343. DOI: http://dx.doi.org/10.1109/TVLSI.2016.2581486

[9] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'14)*. 10–14. DOI: http://dx.doi.org/10.1109/ISSCC.2014.6757323

[10] Intel. 2015a. Sixth generation Intel Core i7 Processors (formerly Skylake). Retrieved from http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html?wapkw=skylake.

[11] Intel. 2015b. The next generation of computing has arrived: Performance to power amazing experiences. Retrieved from http://www.intel.com/content/dam/www/public/us/en/documents/guides/mobile-5th-gen-core-app-power-guidelines-addendum.pdf.

[12] Thomas B. Jablin, James A. Jablin, Prakash Prabhu, Feng Liu, and David I. August. 2012. Dynamically managed data for CPU-GPU architectures. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO'12)*. ACM, New York, NY, 165–174.

[13] Onur Kayiran, Nachiappan Chidambaram Nachiappan, Adwait Jog, Rachata Ausavarungnirun, Mahmut T. Kandemir, Gabriel H. Loh, Onur Mutlu, and Chita R. Das. 2014. Managing GPU concurrency in heterogeneous architectures. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. 114–126.

[14] Y. Kim, J. Lee, J. E. Jo, and J. Kim. 2014. GPUdmm: A high-performance and memory-oblivious GPU architecture using dynamic memory management. In *Proceedings of 20th IEEE International Symposium on Computer Architecture (HPCA'14)*. 546–557. DOI: 10.1109/HPCA.2014.6835963

[15] N. Kirman, M. Kirman, R. K. Dokania, J. F. Martinez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi. 2006. Leveraging optical technology in future bus-based chip multiprocessors. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. 492–503. DOI: http://dx.doi.org/10.1109/MICRO.2006.28

[16] S. Koohi and S. Hessabi. 2014. All-optical wavelength-routed architecture for a power-efficient network on chip. *IEEE Trans. Comput.* 63, 3 (March 2014), 777–792. DOI: http://dx.doi.org/10.1109/TC.2012.171

[17] Jaekyu Lee, Si Li, Hyesoon Kim, and Sudhakar Yalamanchili. 2013a. Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. *ACM Trans. Des. Autom. Electron. Syst.* 18, 4 (Oct. 2013), 48:1–48:28.

[18] Jaekyu Lee, Si Li, Hyesoon Kim, and Sudhakar Yalamanchili. 2013b. Design space exploration of on-chip ring interconnection for a CPU-GPU heterogeneous architecture. *J. Parallel Distrib. Comput.* 73, 12 (2013), 1525–1538.

[19] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. *SIGARCH Comput. Archit. News* 41, 3 (June 2013), 487–498.

[20] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. ACM, New York, NY, 469–480.

[21] Sparsh Mittal and Jeffrey S. Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.* 47, 4 (July 2015), 69:1–69:35.

[22] R. Morris, A. K. Kodi, and A. Louri. 2012. Dynamic reconfiguration of 3D photonic networks-on-chip for maximizing performance and improving fault tolerance. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'12)*. 282–293.

[23] C. Nitta, M. Farrens, and V. Akella. 2011. Addressing system-level trimming issues in on-chip nanophotonic networks. In *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture*. 122–131. DOI: http://dx.doi.org/10.1109/HPCA.2011.5749722

[24] NVIDIA. 2015. Tegra X1. Retrieved from http://www.nvidia.com/object/tegra-x1-processor.html.

[25] Yan Pan, Prabhat Kumar, John Kim, Gokhan Memik, Yu Zhang, and Alok Choudhary. 2009. Firefly: Illuminating future network-on-chip with nanophotonics. *SIGARCH Comput. Archit. News* 37, 3 (June 2009), 429–440.

[26] David A. Patterson and John L. Hennessy. 2013. *Computer Organization and Design: The Hardware/Software Interface* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA.

[27] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar. 2011. A 2 Tb/s 6×4 mesh network for a single-chip cloud computer With DVFS in 45 nm CMOS. *IEEE J. Solid-State Circ.* 46, 4 (April 2011), 757–766. DOI: http://dx.doi.org/10.1109/JSSC.2011.2108121

[28] K. T. Settaluri, S. Lin, S. Moazeni, E. Timurdogan, C. Sun, M. Moresco, Z. Su, Y. H. Chen, G. Leake, D. LaTulipe, C. McDonough, J. Hebding, D. Coolbaugh, M. Watts, and V. Stojanovič. 2015. Demonstration of an optical chip-to-chip link in a 3D integrated electronic-photonic platform. In *Proceedings of the 41st European Solid-State Circuits Conference (ESSCIRC'15)*. 156–159.

[29] A. Shah, N. Mansoor, B. Johnstone, A. Ganguly, and S. L. Alarcon. 2014. Heterogeneous photonic network-on-chip with dynamic bandwidth allocation. In *Proceedings of the 27th IEEE International System-on-Chip Conference (SOCC'14)*. 249–254.

[30] Chen Sun, C.-H. O. Chen, G. Kurian, Lan Wei, J. Miller, A. Agarwal, Li-Shiuan Peh, and V. Stojanovic. 2012. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the 6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12)*. 201–210.

[31] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. 335–344.

[32] Rafael Ubal, Dana Schaa, Perhaad Mistry, Xiang Gong, Yash Ukidave, Zhongliang Chen, Gunar Schirner, and David Kaeli. 2014. Exploring the heterogeneous design space for both performance and reliability. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. 181:1–181:6.

[33] 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 (Jan 2008), 29–41. DOI: http://dx.doi.org/10.1109/JSSC.2007.910957

[34] Dana Vantrease, Robert Schreiber, Matteo Monchiero, Moray McLaren, Norman P. Jouppi, Marco Fiorentino, Al Davis, Nathan Binkert, Raymond G. Beausoleil, and Jung Ho Ahn. 2008. Corona: System implications of emerging nanophotonic technology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA'08)*. IEEE Computer Society, Washington, DC, 153–164. DOI: http://dx.doi.org/10.1109/ISCA.2008.35

[35] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*.

[36] Xiaowen Wu, Jiang Xu, Yaoyao Ye, Zhehui Wang, Mahdi Nikdast, and Xuan Wang. 2014. SUOR: Sectioned undirectional optical ring for chip multiprocessor. *J. Emerg. Technol. Comput. Syst.* 10, 4, Article 29 (June 2014). DOI: http://dx.doi.org/10.1145/2600072

[37] Hui Zhao, Mahmut Kandemir, Wei Ding, and Mary Jane Irwin. 2011. Exploring heterogeneous NoC design space. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'11)*. 787–793.

[38] Amir Kavyan Kavyan Ziabari, Jose L. Abellán, Rafael Ubal, Chao Chen, Ajay Joshi, and David Kaeli. 2015. Leveraging silicon-photonic NoC for designing scalable GPUs. In *Proceedings of the 29th ACM on International Conference on Supercomputing (ICS'15)*. 273–282.

Received June 2017; revised January 2018; accepted February 2018