2018 IEEE International Symposium on High Performance Computer Architecture

Extending the Power-Efficiency and Performance of Photonic Interconnects for Heterogeneous Multicores with Machine Learning

Scott Van Winkle, Avinash Kodi, Razvan Bunescu
School of Electrical Engineering and Computer Science
Ohio University, Athens, Ohio 45701
Email: sv247901@ohio.edu, kodi@ohio.edu, bunescu@ohio.edu

Ahmed Louri
Department of Electrical and Computer Engineering
George Washington University, Washington DC 20052
Email: louri@gwu.edu

***Abstract*—As communication energy exceeds computation energy in future technologies, traditional on-chip electrical interconnects face fundamental challenges in the many-core era. Photonic interconnects have been proposed as a disruptive technology solution due to superior performance per Watt, distance independent energy consumption and CMOS compatibility for on-chip interconnects. Static power due to the laser being always switched on, varying link utilization due to spatial and temporal traffic fluctuations and thermal sensitivity are some of the critical challenges facing photonics interconnects. In this paper, we propose photonic interconnects for heterogeneous multicores using a checkerboard pattern that clusters CPU-GPU cores together and implements bandwidth reconfiguration using local router information without global coordination. To reduce the static power, we also propose a dynamic laser scaling technique that predicts the power level for the next epoch using the buffer occupancy of previous epoch. To further improve power-performance trade-offs, we also propose a regression-based machine learning technique for scaling the power of the photonic link. Our simulation results demonstrate a 34% performance improvement over a baseline electrical CMESH while consuming 25% less energy per bit when dynamically reallocating bandwidth. When dynamically scaling laser power, our buffer-based reactive and ML-based proactive prediction techniques show 40 - 65% in power savings with 0 - 14% in throughput loss depending on the reservation window size.**

Keywords*—Network-on-Chips, Photonics, Power Scaling, Machine Learning.

I. Introduction

Voltage and frequency scaling combined with aggressive transistor scaling has paved the way for many-core processors as the method for improving the performance and power-efficiency of HPC workloads as demonstrated by industry and academia. As the demand for higher performance continues, the parallelism and the speedup offered by GPUs have become an attractive option when compared to CPU-only cores. Such heterogeneous platforms, where CPUs and GPUs are integrated are advantageous since the programmer can pick either the serialized nature of CPUs or the parallel nature of GPUs depending of the application characteristics and thereby, improve the performance. For example, different commercial chips have demonstrated the power of CPU-GPU heterogenous architectures such as Intel's Broadwell [1] and Skylake [2], NVIDIA's Tegra X1 [3], and AMD's Carrizo [4]. CPU and GPU cores share multiple resources within the chip's architecture including the network bandwidth, last level cache, memory controllers, and main memory. Resource management, allocation and interaction between these resources impact the performance and energy-efficiency of the entire chip.

One of the critical challenge exists in the design of the interconnection network that connects the heterogeneous cores with the caches and memory hierarchy. While traditional electronics has been used for many-core architectures, energy-efficiency, resource allocation and scale-out of many-cores puts enormous pressure on the communication fabric. First, the communication energy scales slower than compute energy as showed by Shekar Borkar which makes compute more energy-efficient than data movement [5]. Second, with heterogeneous cores, GPUs tend to overwhelm the network resources with memory requests that are bursty in nature and therefore, the network must balance the requests between the two cores types to ensure fairness and minimize latency [6], [7], [8]. Third, technology scaling will increase the number of cores that can be integrated on the same chip which will further stress the limited on-chip network bandwidth [9]. Therefore, traditional on-chip network design faces several fundamental challenges in the many-core era.

In the light of the above problems, and as on-chip interconnects scale to support heterogeneous cores, alternative interconnect technologies such as silicon photonics are under serious consideration for meeting the networking challenges of future many-core processors [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. A number of critical advantages of silicon photonics have been demonstrated including high bandwidth density with wavelength division multiplexing (WDM), high-speed modulation (10-40 Gbps), low energy consumption for implementing complex network topologies and CMOS compatibility. As the technology continues to mature, working prototypes of the technology have been published and documented [10]. Few of the challenges facing the adoption of this disruptive technology are static laser power consumption, thermal sensitivity, wall-plug efficiency of the lasers, integration of photonic components and others [24], [25], [10]. To tackle the static laser power, SLaC proposed on-chip lasers that have faster turn-on times to create energy consumption proportional to network traffic [18]. By exploiting the temporal and spatial distribution of the network traffic, prior work evaluated bandwidth scaling techniques to improve performance and energy-efficiency [26], [27]. While the prior

2378-203X/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCA.2018.00048

work in improving energy-efficiency of photonic interconnects is significant, no work has combined bandwidth scaling, power scaling and machine learning techniques to improve both power and performance of photonic interconnects for heterogeneous interconnects.

In this paper, we propose **PEARL** - **P**ower-**E**fficient Photonic **A**rchitecture with **R**econfiguration via **L**earning which effectively combines heterogeneous cores (CPUs and GPUs) using a checkerboard pattern to implement simultaneously both dynamic bandwidth and power scaling techniques. The checkerboard patterns combines both CPUs and GPUs into a single router where bandwidth reconfiguration is implemented locally without any global coordination. We propose to use reservation-assist single-writer-multiple-reader (R-SWMR) that enables fine-grain bandwidth reconfiguration by considering a sliding window based buffer occupancy. To reduce the static power due to the laser and the heating power, we propose to implement dynamic power scaling that utilizes on-chip lasers with fast turn-on times to implement coarse-grain power scaling that controls individual lasers (64, 48, 32, 16 and 8 wavelengths). Dynamic power scaling is a reactive technique that uses the buffer occupancy of the prior window to predict the power level for the next window

While power scaling is a reactive technique, we also propose a proactive technique where we determine the power levels by predicting the number of packets injected by the heterogeneous cores using machine learning (ML) algorithms. ML is usually comprised of two phases: (1) in the training phase, which is often done offline, the ML algorithms automatically build predictive models by learning from the training examples; (2) in the test phase, these models are applied to samples not seen during training with the goal of accurately predicting target variables. With learning algorithms, we achieve different power-performance trade-offs with different reconfiguration window sizes with the best result achieved for a window size of 500 cycles. Our simulation results demonstrate a 34% performance improvement over a baseline electrical CMESH while consuming 25% less energy per bit when dynamically reallocating bandwidth. When dynamically scaling laser power, our buffer-based reactive and ML-based proactive prediction techniques show 40 - 65% in power savings with 0 - 14% in throughput loss depending on the reservation window size.

The major contributions of this paper are as follows:

1. Photonic Interconnects for Heterogeneous Multicores: We propose photonic interconnects for heterogeneous multicores using a checkerboard pattern that clusters CPU-GPU cores together and implements bandwidth reconfiguration using local router information without global coordination. 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.

2. Dynamic Power Scaling Techniques: As the laser and trimming power dominate photonic link, we propose a dynamic power scaling technique that utilizes the buffer occupancy as a metric to predict the laser power for the next reservation window. This reactive technique where the buffer occupancy of the prior epoch window determines the ideal laser power for the next epoch window utilizes on-chip lasers that have a shorter turn-on time making the power scaling technique feasible.

3. Machine Learning for Power Scaling: We further improve the power-performance trade-offs by using machine learning algorithm for scaling the power consumed by the photonic link. We propose to use linear regression algorithm in order to predict the number of packets that will be injected into each router for the next window. Using the predicted number of packets, we scale the number of wavelengths and thereby the power consumed by the photonic link.

II. Related Work

A. Photonic Interconnects for CPUs and GPUs

Previous research has shown that both CPUs and GPUs with large number of cores can take advantage of the power-efficiency, lower latencies, and higher bandwidth offered by silicon photonics [12], [13], [14], [15], [16]. Crossbar-based architectures such as Corona, 3D-NoC and others use Multiple Write Single Read (MWSR) which connects multiple source routers to a single destination router. A token-based arbitration mechanism is used to prevent multiple routers from communicating on the photonic channel simultaneously. Firefly reduces the hardware complexity by integrating electrical interconnects for shorter distances [15]. In Firefly, a Single Write Multiple Reader (SWMR) crossbar is used to connect single source router with multiple destination router using a reservation-assisted implementation. The purpose of the reservation packet is to inform the rest of the network which router is receiving the next data packet. This has two effects - (1) The on-chip network no longer needs a complex token arbitration mechanism associated with MWSR. (2) The R-SWMR decreases the laser energy cost on the data waveguide by having only one router listening on the channel to receive the packet. Photonic interconnects for GPUs have used similar token-based MWSR [28]. Recently GPU architectures using dual photonic crossbars have been proposed for 128 computational units (CUs). In the dual crossbar architecture, MWSR is implemented between L1 and L2 which facilitates many-to-few communication pattern and SWMR is implemented between L2 and L1 which facilitates few-to-many communication pattern [13]. In our proposed work, we use reservation-based SWMR to reduce the hardware complexity and control while minimizing the latency for both CPU and GPU workloads.

B. Bandwidth Sharing in CPUs and GPUs

Dynamically managing resources can minimize the energy consumption and maximize the utilization of network resources. Token-based arbitration was optimized in several MWSR architectures to minimize the latency penalty and share the network bandwidth effectively by providing additional timeslots to nodes with more workload [29], [30], [12]. In 3D-NoC, a dynamic bandwidth allocation mechanism was implemented using link and buffer utilization to route traffic to different layers of the chip to maximize throughput and tolerate faults [16]. Photonic bandwidth was dynamically allocated between CPUs and GPUs using R-SWMR communication mechanism with a table-based task allocation [27]. Such a coarse-grain bandwidth allocation could occur only at the boundaries of the task and not at runtime as proposed in this work. Techniques used for dynamic bandwidth allocation

in homogenous architectures can easily be adapted to heterogenous architectures. A feedback directed virtual channel partitioning mechanism which allocates different numbers of virtual channels to both CPUs and GPus has been proposed in [31]. This 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. Moreover, there has been several proposed ways to manage the cache, main memory, and data transfer mechanisms in heterogenous architectures [31], [32], [33], [34], [7].

C. Power Scaling in Photonic NoCs

As static laser power has become a major roadblock, some of the recent work has targeted techniques to power-gate the laser source. While prior work assumed external WDM lasers, these lasers have an efficiency of 5-8% and turn-on time can exceed 1 μ sec making it infeasible to be used for on-chip control [35]. Recent work has demonstrated using InP-based Fabry-Perot lasers than can be switched on within 2 $nsec$, which is in the range of typical processing cores [36], [37]. In SLaC [18], buffer utilization is used to power-gate on-chip laser sources when the application exhibits low periods of network activity. With network connected as a flattened butterfly, alternate routes are available without isolating routers. ATAC proposed to turn-off lasers when idle and turns them on with enough power for one receiver upon receiving a transmit request [38]. PROBE used a prediction mechanism to turn waveguides off to save external laser power [39]. Few works have used machine learning to predict the voltage and frequency levels for electrical NoCs using supervised and reinforcement learning techniques [40], [41]. However, no work has used ML to predict power levels of photonic NoCs for heterogeneous multicores to improve power-efficiency and performance.

III. PEARL Architecture

In this section, we discuss the proposed PEARL architecture, router microarchitecture, dynamic bandwidth scaling, power scaling techniques and the machine learning algorithm to implement power-efficient photonic links for heterogeneous multicores.

A. Architecture

1) Photonic Interconnects: A photonic link requires (i) lasers to generate the carrier signal, (ii) modulators and drivers to encode the data, (iii) medium (waveguides, fibers, free space) for signal propagation, (iv) photodetectors to detect light and (v) back-end signal processing (transimpedance amplifiers (TIA), voltage amplifiers, clock and data recovery) to recover the transmitted bit. Figure 1(a) shows the photonic interconnect which connects two cores A and B with four different wavelengths which are generated by the laser and modulated/demodulated by the microring resonators (MRRs). MRRs have a small footprint (12 μm), lower power (0.1 mW) and can modulate in excess of 18 Gbps with 80 ps modulator delay. Silicon waveguides, which have a smaller pitch of 5.5 μm, a lower propagation time of 10.45 ps/mm and a signal attenuation of 1.3 dB/cm are chosen due to ease of integration with other on-chip photonic components. Finally, germanium-doped photodetectors are used to detect the multiplexed signal.

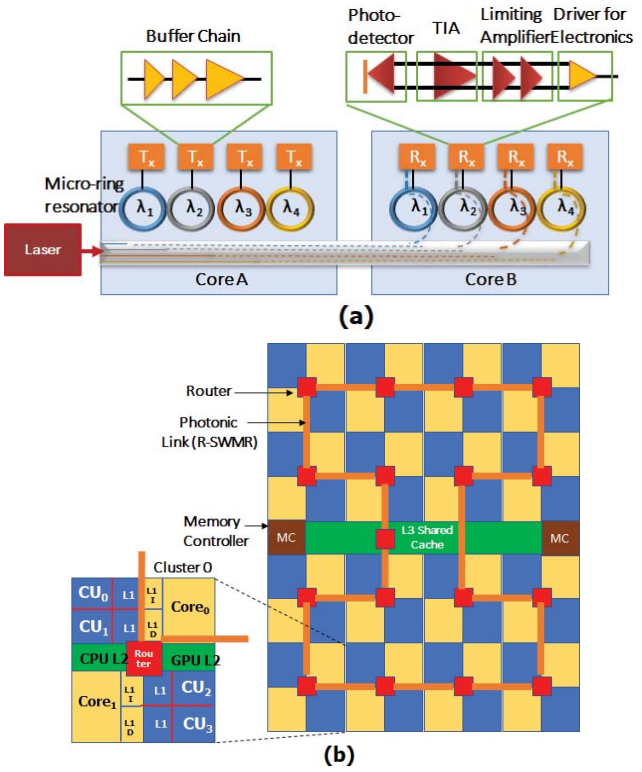

Fig. 1: (a) Photonic link connecting cores A and B. (b) Proposed PEARL architecture consisting of two CPU cores, four GPU computation units (CUs), L3 cache, memory controller (MCs), router and interconnected by photonic links.

Due to thermal sensitivity, ring heaters are used to ensure that the wavelength drift is avoided and signals can be accurately detected; however other solutions including athermal design, runtime thermal optimization and backend switching have been proposed [42], [43], [44]. In this work, we propose on-chip InP-based Fabry Perot lasers with short turn-on delay. On-chip lasers with dimensions 50 μm $\times$ 300 μm $\times$ 5 μm with each channel operating 128 wavelengths have been shown to accommodate as many as 256 cores concentrated into 64 tiles [45]. We assume a smaller configuration of 16 routers with each cluster consisting of 2 CPUs and 4 GPUs in PEARL architecture.

2) PEARL Layout: Figure 1(b) shows our proposed PEARL architecture which consists of 32 CPU and 64 GPU cores interconnected by the photonic link. In PEARL, a *cluster* is made up of two CPU cores, four GPU computational unit (CUs), one router and corresponding L1/L2 caches. This is similar to prior heterogeneous multicore configurations that consider a CPU to occupy twice as much area as one compute unit and connect a single router for both core types [6], [46]. We propose a 28 nm CMOS-SOI process operating at 1.0 V with CPUs operating at 4 GHz and GPUs operating at 2 GHz. We propose a checkerboard pattern such that each router is directly connected to two CPU and four GPU cores. Under high traffic scenarios for one core type, contention is more manageable since both cores contend locally at each

TABLE I: Architecture Specifications

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

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

TABLE II: Area overhead for PEARL[48], [49], [50]

Photonic and Electronic Component	Area
Cluster (CPU, GPU and L1 cache)	25 mm^2
L2 Cache per Cluster	2.1 mm^2
Optical Components (MRRs and Waveguides)	24.4 mm^2
Waveguide Width [51]	5.28 μm
MRR Diameter [12]	3.3 μm
L3 Cache	8.5 mm^2
Router	0.342 mm^2
On-Chip laser per router	0.312 mm^2
Dynamic Allocation	0.576 mm^2
Machine Learning	0.018 mm^2 [49]

router. 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 router at each cluster connects to the shared L3 cache. Table I shows the architecture specifications used in PEARL architecture and Table II shows the area overhead for various architecture components (this also includes the area overhead for dynamic allocation scheme and machine learning components which will be discussed later). The cluster includes the CPUs/GPUs and private L1 caches for both core types. The optical components include the transmitting MRRs, receiving MRRs and the waveguides connecting the router.

When dealing with multiple caches spread throughout the architecture, cache coherence becomes pivotal to multiprocessor applications. A cache coherence protocol ensures that the data received is the most recent version. The cache coherence protocol used with PEARL is NMOESI [47]. All 16 routers are organized in a 4 $\times$ 4 grid and the shared L3 cache is connected using an optical crossbar. 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 L3 cache is split evenly between the CPU and GPU cores. 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 be copied from GPU bank to CPU bank. The L3 cache is connected to two memory controllers (MCs). In order 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 [16].

3) 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. Then, only the intended destination listens on the channel while the transmitter sends the data. Figure 2 shows the router 0 in PEARL 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) stage 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. We have 4 sets of laser array (LA_{0-15}, LA_{16-31}, LA_{32-47}, and LA_{48-63}) which transmits the signal onto the channel. This creates 4 sets with 64, 48, 32 and 16 wavelengths providing different bandwidths. We will further split the lowest level of 16 wavelengths into 8 wavelengths when we implement the power scaling technique. Since we implement R-SWMR, there are 16 inputs (routers 1 to 16 and L3 cache) and they are broken into 4 sets of photodetectors (PD_{0-15}, PD_{16-31}, PD_{32-47}, and PD_{48-63}). At the destination, the reverse process takes place, where the optical signal is filtered and converted into electrical format using a combination of photo-detector, 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). The switch has 16 inputs from the photodetectors, 8 inputs from the L1 caches and L2 cache from the CPUs and GPUs, one output to the laser, and 8 outputs to the CPUs and GPUs as shown. The serializer/deserializer is used to convert the 128 bits of information into 64 wavelengths for E/O and O/E conversions.

B. Dynamic Bandwidth Scaling

As the GPU has the tendency to flood the network [52], care must be taken while designing the dynamic bandwidth allocation algorithm to prevent the GPU from starving the CPU of network resources. Our proposed dynamic bandwidth allocation algorithm is designed with the following goals: (i) the algorithm should work with minimal hardware additions, (ii) the algorithm should operate locally within the confines of each router and thereby avoid complex global management, (iii) the algorithm should prevent the GPU from blocking CPU traffic within the router architecture, and (iv) the algorithm should allow simultaneous transmission of CPU and GPU packets regardless of the packets' destination.

To implement dynamic bandwidth scaling, we pick local management of link bandwidth to mitigate the overhead (i.e. write contention) that is often associated with global bandwidth management techniques [15]. Figure 2 shows the minimal hardware addition to implement the R-SWMR link. When a packet is generated from either the L2 or L1 caches, it is placed in the corresponding input buffer. The buffer occupancy sums the number of buffer slots occupied by packets injected from the CPU and GPU cores and this information is sent to the dynamic bandwidth allocator (DBA). DBA determines the amount of bandwidth to assign to each core type by using the number of buffer slots occupied. Next, the DBA generates a reservation packet similar to R-SWMR communication scheme [15]. The number of bits in the reservation packet can be calculated with $ResPacket_{\text{size}}$=log(2 $\times$ N $\times$ S_{CPU} $\times$ S_{GPU} $\times$ D $\times$ N_{L3}) where N is the number of non-L3 routers in the

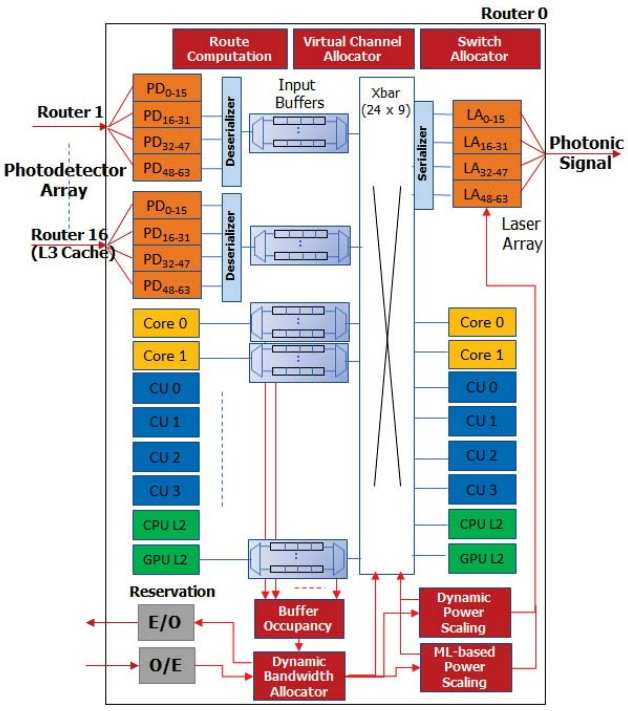

Fig. 2: Proposed router microarchitecture for PEARL. Buffer occupancy feeds to Dynamic Bandwidth Allocator (DBA) which in turn controls either reactive dynamic power scaling or proactive ML-based power scaling.

network, S_{CPU} is the number of different CPU 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 dynamic bandwidth allocation 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.

Algorithm 1 which implements dynamic bandwidth allocation is executed by every router, Rω, $0 \leq \omega \leq R_{max\text{-}1}$ during each cycle. 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:

$$\beta_{ocup\text{-}CPU\omega} = \frac{\sum_{i=0}^{k-1} Buf_i \times a_i}{Bufmax_{CPU}} \quad (1)$$

$$\beta_{ocup\text{-}GPU\omega} = \frac{\sum_{i=0}^{j-1} Buf_i \times a_i}{Bufmax_{GPU}} \quad (2)$$

$$Buf_\omega = \beta_{ocup\text{-}CPU\omega} + \beta_{ocup\text{-}GPU\omega} \quad (3)$$

where $k = Bufmax_{CPU}$ = maximum number of buffer slots for CPU, $j = Bufmax_{GPU}$ = maximum number of buffer slots for GPU, Buf_{total} (= $Bufmax_{CPU}$ + $Bufmax_{GPU}$) is

Algorithm 1 Dynamic bandwidth and power scaling.

For each router R_ω for routers R_0 through $R_{max\text{-}1}$

1) Calculate $\beta_{CPU} = \beta_{ocup\text{-}CPU\omega}$
2) Calculate $\beta_{GPU} = \beta_{ocup\text{-}GPU\omega}$
3) Allocate bandwidth to CPU and GPU:
   a) **if** $\beta_{GPU} = 0$ and $\beta_{CPU} > 0$ **then**
      $GPU_{Bandwidth}$ = 0% *Bandwidth*
      $CPU_{Bandwidth}$ = 100% *Bandwidth*
   b) **else if** $\beta_{CPU} = 0$ and $\beta_{GPU} > 0$ **then**
      $GPU_{Bandwidth}$ = 100% *Bandwidth*
      $CPU_{Bandwidth}$ = 0% *Bandwidth*
   c) **else if** $\beta_{GPU} < \beta_{GPU\text{-}UpperBound}$ **then**
      $GPU_{Bandwidth}$ = 25% *Bandwidth*
      $CPU_{Bandwidth}$ = 75% *Bandwidth*
   d) **else if** $\beta_{CPU} < \beta_{CPU\text{-}UpperBound}$ **then**
      $GPU_{Bandwidth}$ = 75% *Bandwidth*
      $CPU_{Bandwidth}$ = 25% *Bandwidth*
   e) **else**
      $GPU_{Bandwidth}$ = 50% *Bandwidth*
      $CPU_{Bandwidth}$ = 50% *Bandwidth*
4) Send reservation packet via SWMR link
5) Transmit data using allocated bandwidth on FCFS
6) **if** $Current_{Cycle}$ mod RW = 0 **then**
   Proceed to Steps 7 and 8
   else
   Return to Step 0
7) For each RW, sum the total buffer occupancy β_{total}
8) At the end of RW, determine WL for Router R_ω:
   a) **if** $\beta_{total} > Threshold_{upper}$ **then**
      WL= 64 *Wavelengths*
   b) **else if** $\beta_{total} > Threshold_{mid\text{-}upper}$ **then**
      WL= 48 *Wavelengths*
   c) **else if** $\beta_{total} > Threshold_{mid\text{-}lower}$ **then**
      WL= 32 *Wavelengths*
   d) **else if** $\beta_{total} > Threshold_{lower}$ **then**
      WL= 16 *Wavelengths*
   e) **else**
      WL= 8 *Wavelengths*

the total number of CPU and GPU buffers at each router Rω, and a_i is 1 if the buffer slot is occupied. $\beta_{CPU\text{-}UpperBound}$ and $\beta_{GPU\text{-}UpperBound}$ are used as thresholds to determine the bandwidth allocation within the algorithm. Utilizing a brute force method, the optimal $\beta_{CPU\text{-}UpperBound}$ and $\beta_{GPU\text{-}UpperBound}$ are determined experimentally on a separate set of benchmarks than the ones used in the results section of this paper. The optimal $\beta_{CPU\text{-}UpperBound}$ was determined to be 16% of the total CPU input buffer space while the optimal $\beta_{GPU\text{-}UpperBound}$ was determined to be 6% of the total GPU input buffer space. Using the calculated values for β_{CPU} and β_{GPU} in comparison with $\beta_{CPU\text{-}UpperBound}$ and $\beta_{GPU\text{-}UpperBound}$, the bandwidth is assigned to each core type and the reservation packet is created. Due to the temporal sensitivity of the CPU, precedence is given to CPU by considering it first for the 75% bandwidth allocation within Step 3 of Algorithm 1. After sending and receiving the reservation packet, the corresponding routers tune the MRRs to receive the data packets. In order to decide how best to split the bandwidth allocation, we considered a wide range of configurations where bandwidth was allocated in steps of 6.25%, 12.5% and 25% and determined that 25% performed the best.

C. Power Scaling Techniques

The dynamic power scaling scheme described in this section is a reactive technique that utilizes buffer occupancy within a reservation window to determine the next reservation

window's laser power. The proposed dynamic power scaling technique is built on top of the PEARL architecture and Steps 6 through 8 within Algorithm 1 scales the number of wavelengths to implement power scaling. Steps 0 through 5 are executed every cycle to dynamically allocate the bandwidth. Step 6 checks to see if the network has reached the end of the reservation window, RW. If Step 6 evaluates that the reservation window is complete, the power scaling mechanism executes Steps 7 and 8. Step 7 sums up the running buffer occupancy for all the buffers across the reservation window ($\beta_{\text{total}} = \sum(\text{Buf}_{\omega}/\text{Buf}_{total})/\text{RW}$). The power scaling has four thresholds which creates five laser power states. We assume a 2 ns turn on delay for all laser scaling applications. The additional components for the dynamic/ML power scaling can be seen in Figure 3. In this figure the laser banks are divided up into four groups of 16 lasers. This would create the 64, 48, 32, and 16 wavelength states for the dynamic power scaling. In order to utilize the 8 wavelength state, one of the 16 wavelength banks would have to be split in half along with additional control mechanisms. Within Figure 3, the serializer receives the data from the crossbar and passes it onto the multiplexer for E/O conversion. When operating at full bandwidth, 128 bit flit takes two cycles for data transmission as 32 bits of data are split evenly across the four multiplexer. When scaling to 48 wavelengths, one additional 32 bit chunk will be transferred after two cycle delay, therefore taking four clock cycles. Similarly, with 32 wavelengths, the delay will be the same as 48 wavelengths. With 16 wavelengths, it would take eight clock cycles to transmit the flit of 128 bits. The thresholds used for the dynamic power scaling were chosen to balance performance (throughput) and power saving and can be changed to favor either throughput or power. The laser power increases almost linearly with the number of wavelengths and provides different power levels. In PEARL, we implement dynamic bandwidth reconfiguration at the finer granularity, power scaling is implemented at the coarser granularity. Intuitively, bandwidth can be scaled between the cores by re-allocating wavelengths, whereas switching power levels at the finer granularity can cause fluctuations which can cause congestion. Implementing the four-bank design also allows for reducing the trimming power along with the laser for additional power saving [24].

D. Machine Learning (ML) for Power Scaling

The purpose of ML within the context of the laser power scaling is to provide a proactive method for determining the amount of laser power needed at each router within a specified reservation window. The ML model will be used to predict the number of packets that will be injected into each router. Therefore, both the dynamic laser scaling discussed in the previous subsection 3.3 and ML will predict the ideal power level for each RW. ML will replace steps 6 to 8 in Algorithm 1. Once the number of packets is determined, one of five wavelength states will be selected for each router. This ML technique will be used instead of dynamic power scaling model discussed in the previous section. It must be noted that in both dynamic power scaling technique and ML-based power scaling technique, dynamic bandwidth is scaled as per the steps 0 to 5 in Algorithm 1.

1) Regression Model Derivation: A ridge regression model is used to predict the number of packets injected into the network for a given reservation window. The following formula can be used to calculate the cost function for a set of predicted values [53].

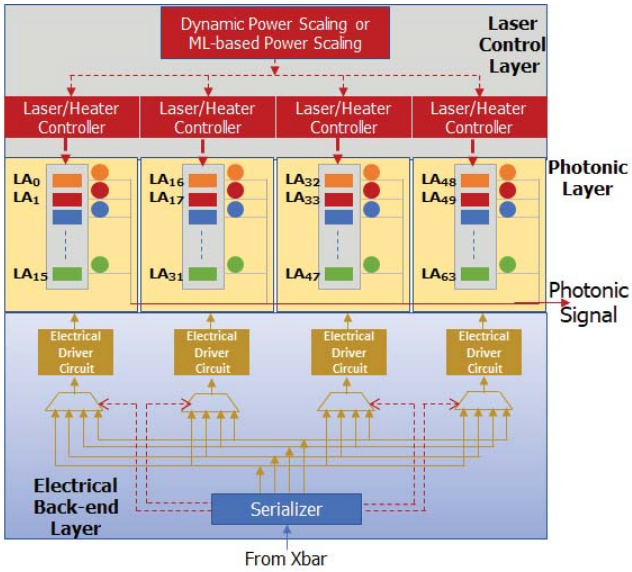

Fig. 3: Dynamic power scaling with on-chip laser with proposed implementation.

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\phi(x_n) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \tag{4}$$

where $\|\mathbf{w}\|^2 \equiv \mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + ... + w_N^2$ represents the norm of the weight vector used by the regression model and the λ is the regularization coefficient. The t_n represents the label for the training example x_n and $\mathbf{w}^T\phi(x_n)$ is the label predicted by the regression model. The trained weight vector $\mathbf{w}$ is the solution of the following convex optimization problem [53]:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \tilde{E}(\mathbf{w}) \tag{5}$$

In order to solve the above equation, the gradient of $\tilde{E}(\mathbf{w})$ must be taken and set to zero. The solution for $\mathbf{w}$ is as follows [53]:

$$\mathbf{w} = (\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\boldsymbol{t} \tag{6}$$

2) Feature Engineering for Dynamic Power Scaling: The features used in the dynamic power scaling model were selected for the purpose of predicting the packets that will be injected into the network. The main constraint for feature selection revolved around the hardware limitations. The features were selected by using the information already present at each router. We need counters at each input buffers, access to packet headers, access to buffers sending packets (CPU/GPU) from the cores and ability to reset the counters at the end of RW. All the above information is input to the ML-based power scaling unit as shown in Figure 2. The L3 router feature (feature 1) is a binary feature used to distinguish between the 16 cluster routers and the L3 cache router. The $Core$ input buffer utilization features (features 2 and 4) represent the buffers

TABLE III: Dynamic Laser Scaling Feature List

1. L3 router
2. CPU Core Input Buffer Utilization
3. Other Router CPU Input Buffer Utilization
4. GPU Core Input Buffer Utilization
5. Other Router GPU Input Buffer Utilization
6. Outgoing Link Utilization
7. Number of Packets Sent to a Core
8. Incoming Packets from Other Routers
9. Incoming Packets from the Cores
10. Request Sent
11. Request Received
12. Responses Sent
13. Responses Received
14. Request CPU L1 instruction
15. Request CPU L1 data
16. Request CPU L2 up
17. Request CPU L2 down
18. Request GPU L1
19. Request GPU L2 up
20. Request GPU L2 down
21. Request L3 22. Response CPU L1 instruction
23. Response CPU L1 data
24. Response CPU L2 up
25. Response CPU L2 down
26. Response GPU L1
27. Response GPU L2 up
28. Response GPU L2 down
29. Response L3
30. Number of Wavelengths

connected to the cores at any given router while the $Other$ router input buffer utilization features (3 and 5) represent the input buffers connected to links from other routers. Features 6 through 13 could have been core type specific but this would increase the number of features by eight and add to the delay and energy to predict the $\beta_{\text{UpperBound}}$ values. There should be enough information in the buffer utilization feature to negate splitting features 6 through 13 into core type specific features. Features 6 and 7 keep track of the packets that stay within the cluster versus the packets that get sent out into the network. Features 8 and 9 sum up the number of packet generated at the given router and the number of packet received from other routers within the network. Features 10 through 13 are used to sum up the total number of request and response packets that move through the router. Features 14 through 30 are used to track the packet movement throughout the network. Each feature has a label of request or response. A request packet is requesting data. A response packet has data. Additionally, the feature is labeled with the core type and the cache with which it is associated. The L2 cache features are labeled with an up or down corresponding to the packet either going to up to an L1 cache or down to an L3 cache.

IV. Performance Evaluation

In this section, we evaluate the power and performance of our proposed dynamic bandwidth and power scaling techniques enhanced with ML for various reservation window sizes on both CPU and GPU workloads. As PEARL can be implemented without dynamic bandwidth scaling, we create two variations - one with dynamic bandwidth scaling implemented called PEARL-Dyn with constant 64 wavelengths operating at all times and PEARL with First-Come First-Serve(PEARL-FCFS) with 64 wavelengths. We also create power scaling variations with different window sizes of 500 and 2000 (Dyn RW500 and Dyn RW2000) and ML-based power scaling with similar window sizes of 500 and 2000 (ML RW 500 and ML RW2000). The two window sizes 500 and 2000 were picked after running the ML and dynamic power scaling model over several window sizes (100-2000) and determining the window size that provided the best power savings while improving performance. While we tried large window sizes (beyond 2000 cycles) to accommodate slow tuning of off-chip lasers that can be a few microseconds, our training did not work well. We compare the wavelength scaling along the four wavelength states of 64, 48, 32 and 16. We introduce a low wavelength state of 8 wavelengths which is the lowest power state of our architecture. The 8WL is introduced as an equivalent to a power-gated design with extremely low power consumption; complete power-gated design creates fluctuations that makes predicting the wavelength state very difficult as buffers fill up very quickly with no outward traffic. We reintroduced the 8WL state after the model was computed to help in save power. The dynamic power scaling, ML-based power scaling, and the PEARL-Dyn architectures utilize the dynamic bandwidth allocation mechanism described in Steps 1 through 5 in Algorithm 1. As a baseline, we compare our proposed PEARL architecture to an electrical concentrated mesh architecture (CMESH). CMESH is designed to have the same bisection bandwidth as the PEARL architectures with constant 64 wavelengths. The router microarchitecture is similar to PEARL 2 with each input port consists of 4 VCs, 4 input buffers per VC, each buffer slot is 128 bits and connects 2 cores, 4 compute units along with L1 and L2 caches as specified in I.

TABLE IV: Benchmarks used for testing ML.

Core Type	Abbre-viation	Benchmark Name
CPU	FA	Fluid Animate
	fmm	Fast Multipole Method
	Rad	Radiosity
	x264	x264
GPU	DCT	Discrete Cosine Transforms
	Dwrt	1-D Haar Wavelet Transform
	QRS	Quasi Random Sequence
	Reduc	Reduction

A. Machine Learning Setup

The number of packets that are being injected into the router is used as the label for the corresponding vector of features for predicting the power levels using ML. This label is chosen over other metrics (i.e., buffer utilization, router utilization, or link utilization) to minimize the effect the wavelength state has on the outcome of the prediction. The buffer, link, and router utilizations vary significantly based on the number of wavelengths assigned to each waveguide. Packets received from other routers are going to dictate the injection of packets at the local router. Thus, minimizing the effect the number of wavelengths has on the local router's injection of packets. If the model was trying to predict the buffer utilization, setting the laser power state to 8 wavelengths would cause the buffer to fill at an increased rate due to the small number of packets leaving the router. In contrast, if the laser power state was set to 64 wavelengths the buffers would have a greater chance of being empty. When predicting the number of packets that will be injected, the core will try to inject a packet regardless of the laser power state.

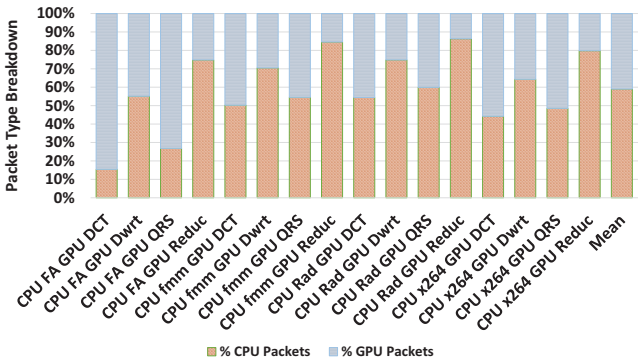

Fig. 4: CPU-GPU packet breakdown for each traffic trace.

TABLE V: Optical components used in estimating the power consumed by PEARL [51], [18], [45]

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

In order to create an accurate machine learning model three data sets must be created: training, validation, and testing. We use two simulators to implement the data collection for this process. The feature data is collected from a modified network simulator running real network traffic. The network traffic is acquired from Multi2Sim [47] full system simulator. Each traffic file consists of one CPU benchmark ran simultaneously with one GPU benchmark. A total of 12 CPU benchmarks are acquired from the PARSEC 2.1 [54] and SPLASH2 [55] benchmark suites. These benchmarks represent a mix of compute and memory-intensive workloads [56]. There are 12 GPU benchmarks selected from the OpenCL SDK benchmark suite. We could not determine whether the GPU benchmarks were compute or memory-bound, however we observed the bursty nature of traffic which is typical of GPU traffic. The training data is created with 6 CPU benchmarks and 6 GPU benchmarks. The benchmarks are combined to create 36 benchmark pairs. The validation data is created using 2 CPU and 2 GPU benchmarks and is used to tune the λ regularization coefficient from Equation 4. The CPU and GPU benchmarks used for the validation data are combined to make 4 benchmark pairs. Lastly, the testing data is created using 4 CPU and 4 GPU benchmarks. These benchmarks are combined to make 16 benchmark pairs. Abbreviations for each benchmarks can be seen in Table IV. The training and validation benchmark pairs are used to gather the feature data described in Table III.

Figure 4 is the packet percentage between each core type for the real traffic benchmark pairs. Although CPU benchmarks create more packets than GPU benchmarks, our dynamic bandwidth allocation ensures that neither core type monopolizes the network bandwidth and the bandwidth allocation is based on demand. The network simulator was used to collect feature data at each router. The feature set is then given a label and the

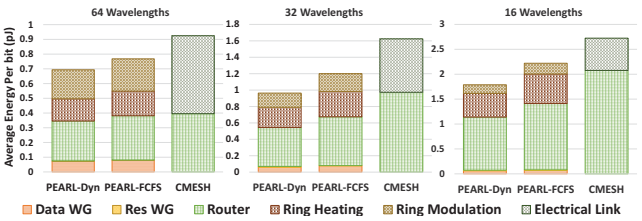

Fig. 5: Energy per bit for competing PEARL networks.

feature counters are set back to zero. The reservation window for the data collection was fixed to 500 or 2000 network cycles. The feature collection for each router is offset by 10 network cycles to prevent all the routers from changing wavelength state within the same network cycle. The initial feature data is collected using randomly generated wavelength states. This is done to avoid influencing the ML process by a predefined pattern. Once an initial regression model is generated, a second feature collection was implemented. This process was done with the wavelength states being generated by the first regression model. The second data collection is designed to best mimic the testing environment for the final results.

B. Power Consumption

The optical components and their losses used in the power estimation of PEARL architecture can be seen in Table V [57], [58]. There are five different wavelength states considered for the dynamic laser scaling: 64, 48, 32, 16, and 8. An aggressive 16 Gbps data rate per wavelength [11], [51] was chosen to achieve the network frequency from Table I. Initially, the 8 wavelength state was omitted during the training and validation of the machine learning model due to the significant decrease in the prediction accuracy. The 8 wavelength state was reintroduced after the model was computed to help save in power consumption. The power values are computed to be 1.16 W, 0.871 W, 0.581 W, 0.29 W and 0.145 W which correspond to 64, 48, 32, 16 and 8 wavelengths respectively.

The thresholds that change the wavelength states can dramatically affect the performance of the network. In the non-ML based approach, buffer utilization was used to predict the target power level for the next RW. The thresholds for the ML was chosen based on the number of packets that can exit the router during any given RW. If the predicted number of packets that are injected into the router exceeds the amount of data that can be sent out of the data waveguide, then the router increases the number of wavelengths used for that RW. This ise described using the following formula:

$$PredictPkt \times PktSz \leq \frac{WL}{state} \times \frac{DataRate}{WL} \quad (7)$$

where $PredictPkt$ is the number of packet the ML model predicts, $PktSz$ is the size of the packets sent, $\frac{WL}{state}$ is the number of wavelengths in the comparing state (in our case 16, 32, 48, and 64), and $\frac{DataRate}{WL}$ is the data chosen for a single wavelength. The number of features dictates the addition and multiplies needed to implement ML. In order to compute the number of packets injected into a router, approximately there will be 30 multiplies and 29 additions. Assuming 16 bit numbers, the total energy needed is 44.6 pJ [49] at a computation

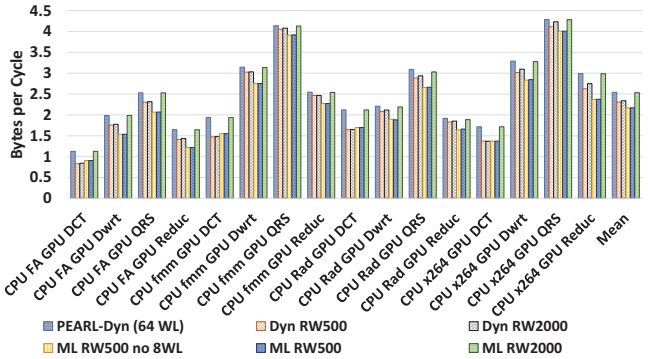

Fig. 6: Throughput comparison of power scaling architectures with 8WL low state.

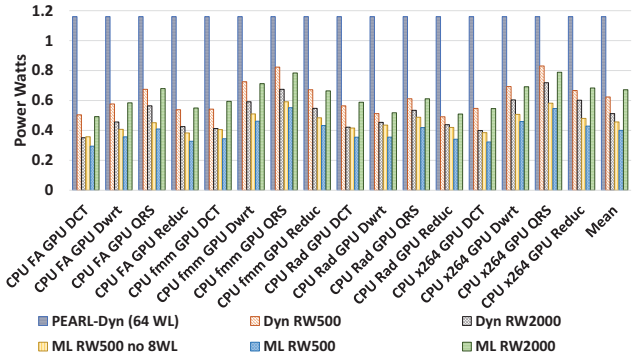

Fig. 7: Average laser power comparison of power scaling architectures with 8WL low state.

time of 5 nanoseconds. The computation time was estimated using Synopsys Design Compiler. An estimate for the power needed for the ML calculation is 178.4 μWatts (46.4 μWatts for adds and 132 μWatts for multiples). This calculation is based on a 500 cycle reservation window and the energy estimates were derived from [49]. Since power consumption and delay will increase with more features, we experimented with lesser features. However, our results neither improved the power nor throughput and therefore, we considered all 30 features in our simulation.

C. Throughput and Power Results

First, we will compare the energy-per-bit of PEARL-Dyn (with no power scaling) to PEARL-FCFS and CMESH architectures as shown in Figure 5. We consider three wavelengths - 64, 32 and 16 - as static configurations and evaluate the energy/bit. For the CMESH, we reduce the bandwidth proportionally to make it comparable to other photonic networks. When the bandwidth is constrained from 64 to 32 wavelengths, PEARL-Dyn shows a 19.7% and 3.2% energy per bit decrease when compared to PEARL-FCFS. PEARL-Dyn demonstrates a 40.7% and 34.4% decrease in energy per bit when compared to CMESH, respectively for 32 and 16 wavelengths. Additionally, PEARL-Dyn demonstrates a 91.9% and 88.8% decrease in energy/bit when compared to CMESH, respectively for 32 and 16 wavelengths. Clearly, constraining the bandwidth helps to improve both the energy as well as the throughput of the network when compared to the CMESH and PEARL-FCFS architectures. With only bandwidth scaling implemented, PEARL-Dyn improves the energy/bit of heterogeneous architectures.

Second, we will compare the throughput and power consumed by different designs - we compare the 64WL (PEARL-Dyn) with no power scaling, Dyn RW500 and Dyn RW2000 that implements only dynamic power scaling with no ML, and ML RW500 no8WL and ML RW2000 that implement ML for reservation window sizes of 500 and 2000. In addition, we also implement a low-power state of 8WL to further save power (ML RW500). Figure 6 and Figure 7 show the breakdown of the throughput and laser power for different window sizes. Figure 8 (a-b) represent the percent of the simulation time each configuration resides in each dynamic laser scaling states for ML only. When looking at the throughput in Figure 6, the best performing configuration was the ML RW2000 with a 0.3% throughput loss compared the 64 WL baseline. When comparing the laser power consumption in Figure 7, ML RW2000 showed 42% improvement over the 64 WL baseline. The reason for this can be seen in Figure 8(b). ML RW 2000 spends just under 30% of the simulation in the 64 WL state. Dyn RW2000 shows 8% loss in throughput when compared to the 64WL baseline. In terms of power savings, Dyn RW2000 shows 55.8% power savings when compared to the 64WL baseline. Therefore, dynamic power scaling without ML saves more power than ML-based technique but has a higher loss in throughput.

The change in reservation window size reduces the throughput loss when implementing the dynamic laser scaling. For a reservation window of 500 cycles, Dyn RW500 has a 1.3% throughput loss over the 64 WL baseline. Dyn RW500 had a 46% power savings over the 64 WL baseline. The ML RW500 and ML RW500 no 8WL configurations perform the same with regard to throughput and both demonstrate a throughput loss of 14%. When the 8 WL state is included, ML RW 500 demonstrates a 65.5% power savings over the 64 WL baseline compared to the ML RW500 no 8WL's 60.7% power savings. This demonstrates that the 8 WL state can improve the power savings of the architecture. If the application needs to maintain the throughput of the network, ML RW 2000 demonstrates a negligible throughput loss with a power savings of 42% when compared to the 64 WL baseline. With ML-based approach, the maximum power saving is achieved with a window size of 500, and adding the 8 wavelength state helps to further improve the power savings. Similarly, window size of 2000 helps in improving the throughput of the architecture by maintaining high throughput with lesser power savings. This provides power-performance trade-offs where a window size of 500 could maximize power savings whereas the window size of 2000 could maximize throughput. Similarly, dynamic power scaling without ML can provide different trade-offs - window size of 2000 saves more power (55%) with throughput loss (8%) where as with window size of 500 shows no throughput loss (1.3%) but with lower power savings (46%).

To evaluate the predictive performance of our algorithms, we computed the normalized root-mean-square error (NRMSE) where 1 represents perfect fit and -∞ corresponds to the worst fit. ML RW500 with and without the 8WL state

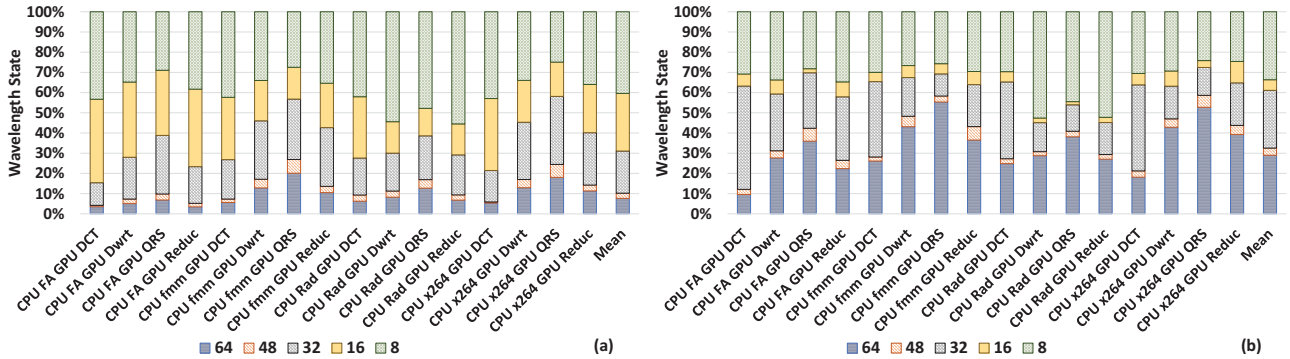

Fig. 8: Wavelength state (a) ML-based power scaling with ML RW500, and (b) ML-based power scaling with ML RW2000.

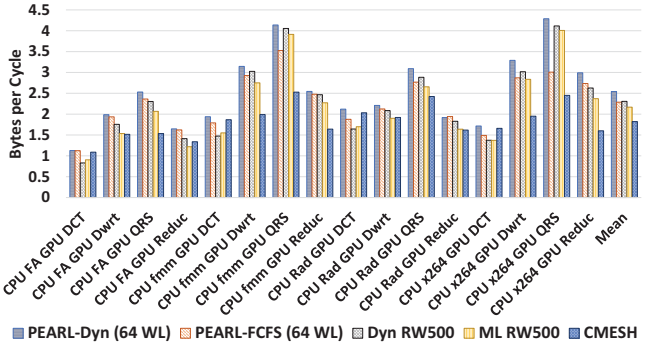

Fig. 9: Throughput comparison for RW500 without 8WL low and baseline architectures.

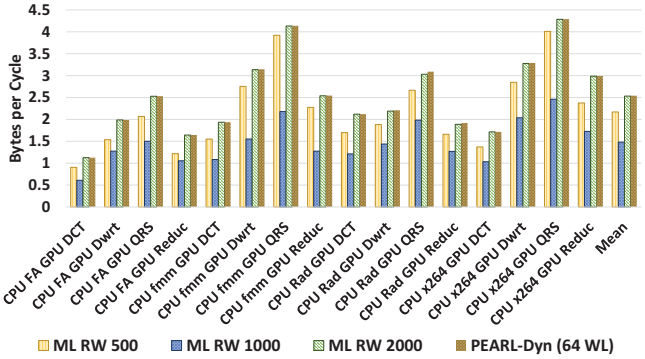

Fig. 10: Throughput comparison of machine learning power scaling architectures for varying reservation windows.

showed a marginal drop in NRMSE from 0.79 in the validation phase to 0.68 in the testing phase, whereas the drop in NRMSE value for the ML RW2000 was more significant as it went from 0.79 in the validation phase to 0.05 in the testing phase. However, for ML RW2000 case, the case where we use the predicted packets to select the highest 64 WL state resulted in a WL state selection accuracy of 99.9%, indicating that by accurately selecting the highest state we were able to obtain the best performance.

Figure 9 shows the throughput when comparing dynamic power scaling designs with and without ML to FCFS and CMESH architectures. The results demonstrate that the dynamic and machine learning power scaling techniques outperform the CMESH topology by 34% and 20% respectively. Moreover, dynamic power scaling technique shows identical throughput with PEARL-FCFS architecture and loses around 8% in throughput when compared to PEARL-Dyn with 64 wavelengths. Figure 10 shows the impact on laser power saving with different reservation window sizes. As seen, the best throughput was achieved for ML2000 window size which predicts the highest wavelength state accurately. For ML RW500 and ML RW1000 window sizes, the throughput drops when compared to the static 64 wavelength state. With a window size of 500, ML makes the best prediction to maximize power savings and with a window size of 2000, we get the best throughput.

Third, we evaluate the sensitivity of laser stabilization on our power and throughput results. From prior work [36] we expect that the laser to stabilize within 2-10 nsec duration with improvement in device technology. We conduct a sensitivity study where laser stabilization period is varied from 2-32 nsec for dynamic power scaling without ML. The power variations was negligible (less than 1%) for different laser turn-on latencies as shown in Figure 11. However, since no data is transmitted during laser stabilization, throughput loss for Dyn RW500 ranges from 0% to 17.9% with an average throughput loss of 7.7% from a 2ns to 4ns turn-on delay. The throughput loss for Dyn RW2000 ranges from 0% to 17.3% with an average throughput loss of 5.3% from a 2ns to 4ns turn-on delay.

V. Conclusions

In this work, we designed a photonic interconnect that adapts to network traffic generated by heterogeneous multicores by dynamically scaling both the link bandwidth as well as the power consumed. The proposed bandwidth scaling allows fair sharing of network resources between CPUs and GPUs and our proposed PEARL-Dyn improves throughput by over 34% when compared over PEARL-FCFS and CMESH architectures. To save static power, we designed two techniques - dynamic power scaling and ML-based power scaling - that operates on top of dynamic bandwidth scaling technique. When dynamically scaling laser power, our buffer-based reactive and

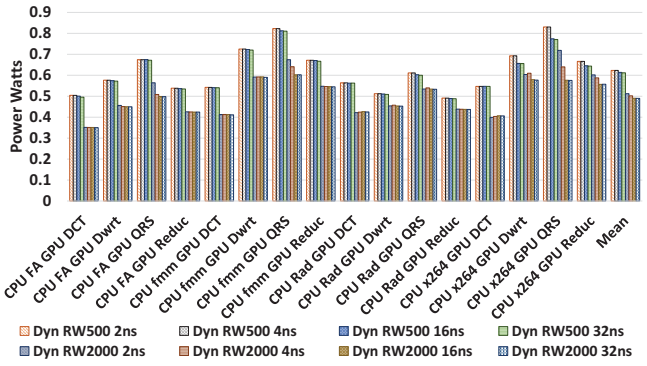

Fig. 11: Average laser power comparison while varying laser turn on time for 2, 4, 16, and 32 nanoseconds.

ML-based proactive prediction techniques show 40 - 65% in power savings with 0 - 14% in throughput loss depending on the reservation window size. Both dynamic power scaling and ML-based technique show power-performance trade-offs at different window sizes. We believe that ML-based research can further optimize the power-performance of photonic NoCs by improving the prediction accuracy.

VI. Acknowledgement

This research was partially supported by NSF grants CCF-1054339 (CAREER), CCF-1420718, CCF-1318981, CCF-1513606, CCF-1703013, CCF-1547034, CCF-1547035, CCF-1540736, and CCF-1702980. We thank the anonymous reviewers for their excellent feedback.

References

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

[2] Intel. 6th generation intel core i7 processors (formerly skylake). [Online]. Available: http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html?wapkw=skylake

[3] NVIDIA. Tegra x1. [Online]. Available: http://www.nvidia.com/object/tegra-x1-processor.html

[4] AMD. Carrizo. [Online]. Available: http://www.amd.com/en-us/who-we-are/corporate-information/events/isscc

[5] S. Borkar, "Exascale computing - a fact or a fiction?" in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 3–3.

[6] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Managing gpu concurrency in heterogeneous architectures," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47, 2014, pp. 114–126.

[7] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, 2009, pp. 163–174.

[8] R. Ubal, D. Schaa, P. Mistry, X. Gong, Y. Ukidave, Z. Chen, G. Schirner, and D. Kaeli, "Exploring the heterogeneous design space for both performance and reliability," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14, 2014, pp. 181:1–181:6.

[9] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel, T. Andreas, B. Liu, A. T. Tran, E. Adeagbo, and B. M. Baas, "Kilocore: A 32-nm 1000-processor computational array," *IEEE Journal of Solid-State Circuits*, vol. PP, no. 99, pp. 1–12, 2017.

[10] C. Sun, M. T. Wade, Y. Lee, J. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. Shainline, R. Avizienis, S. Lin, B. R. Moss, R. Kumar, F. Pavanello, A. H. Atabaki, H. M. Cook, A. J. Ou, J. Leu, Y.-H. Chen, K. Asanovi, and V. Stojanovic, "Single-chip microprocessor that communicates directly using light," vol. 528, pp. 534–538, 12 2015.

[11] D. A. B. Miller, "Device requirements for optical interconnects to silicon chips," *Proceedings of the IEEE*, vol. 97, no. 7, pp. 1166–1185, July 2009.

[12] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. Beausoleil, and J. Ahn, "Corona: System implications of emerging nanophotonic technology," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 153–164.

[13] A. K. K. Ziabari, J. L. Abellán, R. Ubal, C. Chen, A. Joshi, and D. Kaeli, "Leveraging silicon-photonic noc for designing scalable gpus," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15, 2015, pp. 273–282.

[14] N. Kirman and J. F. Martínez, "A power-efficient all-optical on-chip interconnect using wavelength-based oblivious routing," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 15–28, Mar. 2010.

[15] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary, "Firefly: Illuminating future network-on-chip with nanophotonics," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 429–440, Jun. 2009.

[16] R. Morris, A. Kodi, and A. Louri, "Dynamic reconfiguration of 3d photonic networks-on-chip for maximizing performance and improving fault tolerance," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, 2012, pp. 282–293.

[17] H. Jia, Y. Xia, L. Zhang, J. Ding, X. Fu, and L. Yang, "Four-port optical switch for fat-tree photonic network-on-chip," *Journal of Lightwave Technology*, vol. 35, no. 15, pp. 3237–3241, Aug 2017.

[18] Y. Demir and N. Hardavellas, "Slac: Stage laser control for a flattened butterfly network," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 321–332.

[19] A. Joshi, C. Batten, Y.-J. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic, "Silicon-photonic clos networks for global on-chip communication," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, May 2009, pp. 124–133.

[20] 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, "Building manycore processor-to-dram networks with monolithic silicon photonics," in *2008 16th IEEE Symposium on High Performance Interconnects*, Aug 2008, pp. 21–30.

[21] A. Shacham, K. Bergman, and L. P. Carloni, "Photonic networks-on-chip for future generations of chip multiprocessors," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1246–1260, Sept 2008.

[22] Z. Wang, Z. Pang, P. Yang, J. Xu, X. Chen, R. K. V. Maeda, Z. Wang, L. H. Duong, H. Li, and Z. Wang, "Moca: An inter/intra-chip optical network for memory," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: ACM, 2017, pp. 86:1–86:6. [Online]. Available: http://doi.acm.org/10.1145/3061639.3062286

[23] S. Beamer, C. Sun, Y.-J. Kwon, A. Joshi, C. Batten, V. Stojanović, and K. Asanović, "Re-architecting dram memory systems with monolithically integrated silicon photonics," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 129–140, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1816038.1815978

[24] C. Nitta, M. Farrens, and V. Akella, "Addressing system-level trimming issues in on-chip nanophotonic networks," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 122–131.

[25] M. Georgas, J. Leu, B. Moss, C. Sun, and V. Stojanovi, "Addressing link-level design tradeoffs for integrated photonic interconnects," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, Sept 2011, pp. 1–8.

[26] R. Morris, A. K. Kodi, and A. Louri, "3d-noc: Reconfigurable 3d photonic on-chip interconnect for multicores," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, Sept 2012, pp. 413–418.

[27] A. Shah, N. Mansoor, B. Johnstone, A. Ganguly, and S. L. Alarcon,

"Heterogeneous photonic network-on-chip with dynamic bandwidth allocation," in *2014 27th IEEE International System-on-Chip Conference (SOCC)*, Sept 2014, pp. 249–254.

[28] N. Goswami, Z. Li, A. Verma, R. Shankar, and T. Li, "Integrating nanophotonics in gpu microarchitecture," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 425–426. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370878

[29] Y. Pan, J. Kim, and G. Memik, "Featherweight: Low-cost optical arbitration with qos support," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 105–116. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155633

[30] A. Zulfiqar, P. Koka, H. Schwetman, M. Lipasti, X. Zheng, and A. Krishnamoorthy, "Wavelength stealing: An opportunistic approach to channel sharing in multi-chip photonic interconnects," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 222–233. [Online]. Available: http://doi.acm.org/10.1145/2540708.2540728

[31] J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 4, pp. 48:1–48:28, Oct. 2013.

[32] T. B. Jablin, J. A. Jablin, P. Prabhu, F. Liu, and D. I. August, "Dynamically managed data for cpu-gpu architectures," ser. CGO '12. New York, NY, USA: ACM, 2012, pp. 165–174.

[33] Y. Kim, J. Lee, J. E. Jo, and J. Kim, "Gpudmm: A high-performance and memory-oblivious gpu architecture using dynamic memory management," Feb 2014, pp. 546–557.

[34] X. Chen, L.-W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W.-M. Hwu, "Adaptive cache management for energy-efficient gpu computing," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47, 2014, pp. 343–355.

[35] S. Tanaka, S. H. Jeong, S. Sekiguchi, T. Kurahashi, Y. Tanaka, and K. Morito, "Highly-efficient, low-noise si hybrid laser using flip-chip bonded soa," in *2012 Optical Interconnects Conference*, May 2012, pp. 12–13.

[36] E. Kotelnikov, A. Katsnelson, K. Patel, and I. Kudryashov, "Highpower single-mode ingaasp/inp laser diodes for pulsed operation," *Proceedings of SPIE*, vol. 8277 827715, pp. 1–6, 2012.

[37] M. J. R. Heck and J. E. Bowers, "Energy efficient and energy proportional optical interconnects for multi-core processors: Driving the need for on-chip sources," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 20, no. 4, pp. 332–343, July 2014.

[38] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal, "Atac: A 1000-core cache-coherent processor with on-chip optical network," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 477–488. [Online]. Available: http://doi.acm.org/10.1145/1854273.1854332

[39] L. Zhou and A. K. Kodi, "Probe: Prediction-based optical bandwidth scaling for energy-efficient nocs," in *2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, April 2013, pp. 1–8.

[40] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "Svr-noc: A performance analysis tool for network-on-chips using learning-based support vector regression model," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 354–357. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485374

[41] W. Choi, K. Duraisamy, R. G. Kim, J. R. Doppa, P. P. Pande, R. Marculescu, and D. Marculescu, "Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '16. New York, NY, USA: ACM, 2016, pp. 13:1–13:10. [Online]. Available: http://doi.acm.org/10.1145/2968455.2968510

[42] L. Zhou, K. Kashiwagi, K. Okamoto, R. P. Scott, N. K. Fontaine, D. Ding, V. Akella, and S. J. B. Yoo, "Towards athermal optically-interconnected computing system using slotted silicon microring resonators and rf-photonic comb generation," *Applied Physics A*, October 2008.

[43] S. Manipatruni, R. Dokania, B. Schmidt, N. Droz, C. Poitras, A. Apsel, and M. Lipson, "Wide temperature range operation of micron-scale silicon electro-optic modulators," *Optics Letters*, vol. 33, no. 19, September-October 2008.

[44] M. Georgas, J. Leu, B. Moss, C. Sun, and V. Stojanovic, "Addressing link-level design tradeoffs for integrated photonic interconnects," in *CICC*, 2011, pp. 1–8.

[45] J. L. Abelln, A. K. Coskun, A. Gu, W. Jin, A. Joshi, A. B. Kahng, J. Klamkin, C. Morales, J. Recchio, V. Srinivas, and T. Zhang, "Adaptive tuning of photonic devices in a photonic noc through dynamic workload allocation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 5, pp. 801–814, May 2017.

[46] J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Design space exploration of on-chip ring interconnection for a cpu-gpu heterogeneous architecture," *J. Parallel Distrib. Comput.*, vol. 73, no. 12, pp. 1525–1538, 2013.

[47] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12, 2012, pp. 335–344.

[48] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "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*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 469–480.

[49] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[50] X. Zhang and A. Louri, "A multilayer nanophotonic interconnection network for on-chip many-core communications," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10, 2010, pp. 156–161.

[51] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, 2012, pp. 201–210.

[52] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 69:1–69:35, Jul. 2015.

[53] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[54] C. Bienia and K. Li, " PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors ," in *Proc. of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[55] S. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proc. of the 22nd International Symposium on Computer Architecture*, June 1995.

[56] C. Bienia, S. Kumar, and K. Li, "Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *2008 IEEE International Symposium on Workload Characterization*, Sept 2008, pp. 47–56.

[57] K. Aisopos, C.-H. O. Chen, and L.-S. Peh, "Enabling system-level modeling of variation-induced faults in networks-on-chips," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 930–935. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024931

[58] J. Ahn, M. Fiorentino, R. G. Beausoleil, N. Binkert, A. Davis, D. Fattal, N. P. Jouppi, M. McLaren, C. M. Santori, R. S. Schreiber, S. M. Spillane, D. Vantrease, and Q. Xu, "Devices and architectures for photonic chip-scale integration," *Applied Physics A*, vol. 95, no. 4, pp. 989–997, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00339-009-5109-2