Hybrid Network-on-Chip Architectures for Accelerating Deep Learning Kernels on Heterogeneous Manycore Platforms

Wonje Choi*, Karthi Duraisamy*, Ryan Gary Kim†, Janardhan Rao Doppa*, Partha Pratim Pande*, Radu Marculescu†, Diana Marculescu†

*School of EECS
Washington State University
Pullman, WA 99164, U.S.A.
{wchoi1, kduraisa, jana, pande}@eecs.wsu.edu

†ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
{rgkim, radum, dianam}@cmu.edu

Abstract

In recent years, designing specialized manycore heterogeneous architectures for deep learning kernels has become an area of great interest. However, the typical on-chip communication infrastructures employed on conventional manycore platforms are unable to handle *both* CPU and GPU communication requirements efficiently. Hence, in this paper, our aim is to enhance the performance of heterogeneous manycore architectures through the design of a hybrid NoC consisting of both wireline and wireless links. To this end, we specifically target the resource-intensive backpropagation algorithm commonly used as the training method in deep learning. For backpropagation, the proposed hybrid NoC achieves 1.9X reduction in network latency and improves the network throughput by a factor of 2 with respect to a highly optimized mesh NoC. These network level improvements translate into 25% savings in full system energy-delay-product (EDP). This demonstrates the capability of the proposed hybrid and heterogeneous manycore architecture in accelerating deep learning kernels in an energy-efficient manner.

CCS Concepts

C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Interconnection architectures, I.5.1 [**Computing Methodologies**]: Neural Networks, C.2.1 [**Network Architecture and Design**]: Wireless communication.

Keywords

Manycore, Heterogeneous, NoC, Deep learning, Backpropagation

1. Introduction

Deep learning techniques have seen great success in diverse application domains including speech processing, computer vision, natural language processing, and even mastering the game of *Go* [1][2]. While the fundamental ideas of deep learning have been around since the mid-1980s [3], the two main reasons for their recent success are: 1) availability of large-scale training data; and 2) advances in computer architecture to efficiently train large-scale neural networks using this training data.

Deep learning refers to a class of machine learning algorithms, where the goal is to train a non-linear function approximator represented as a neural network architecture by using input-output pairs of training data. Backpropagation (short for "backward propagation of errors") is a fundamental part of deep learning that is used to train various types of neural networks [3]. Backpropagation is an iterative optimization algorithm; at each iteration, it first computes the predicted output by forwarding the input data through the network using the current weights (forward pass). Then, it passes the gradient of the prediction error backwards through the network (backward pass). Lastly, backpropagation updates the weights of the network using the gradient(s) from the prediction error through a variant of the stochastic gradient descent (SGD) optimization [3].

In the backpropagation algorithm, the computations associated with different neurons in the same layer exhibit high parallelism. By exploiting this parallelism, the data-intensive operations associated with backpropagation can be significantly accelerated using GPU cores [4]. However, the execution of the backpropagation algorithm also involves high volumes of data exchanges between the CPUs and GPU accelerators [5]. In a discrete GPU system, the communication between the CPUs and GPUs is carried out by using off-chip interconnects that exhibit high data-transfer latency and high power consumption [6][7][8]. A heterogeneous single chip multiprocessor (CMP) solution in which the CPUs and GPUs are interconnected through the on-chip network will avoid such expensive off-chip data transfers and lead to improved system performance.

We note that conventionally, data centers and high performance computing (HPC) clusters are employed to solve deep learning applications. However, the design of data centers and HPC clusters is dominated by power, thermal, and area constraints. Hence, we envision a Datacenter-on-Chip (DoC) architecture specifically targeting deep learning applications where the entire system (or a large part thereof) can be designed using a heterogeneous manycore-based single-chip architecture. It is well understood that with this massive level of integration, traditional Network-on-Chip (NoC) architectures, e.g., mesh, tree, ring, cannot provide a scalable, low latency and energy-efficient communication backbone, which is essential for solving the deep learning problems targeted in this work [9]. On the other hand, wireless NoCs (WiNoCs) are capable of achieving an energy–efficient and low-latency communication infrastructure for massive manycore chips [10][11]. It is already demonstrated that WiNoC outperforms conventional wireline NoC architectures in terms of achievable bandwidth and energy dissipation [12]. Consequently, inspired by the successes of WiNoC, as our main contributions: *i)* we explore heterogeneous (i.e., combination of CPUs and GPUs) systems combined with hybrid (i.e., combination of wired and wireless links) NoC architectures for deep learning, and *ii)* present a generic design methodology that can be instantiated for *any* combination of neural networks and application domains.

The remainder of the paper is organized as follows. In Section 2, we present some of the relevant works and highlight our novel

CASES '16, October 01-07, 2016, Pittsburgh, PA, USA
 ISBN 978-1-4503-4482-1/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2968455.2968510

contributions. In Section 3, we explain the salient features of the backpropagation benchmark considered in this work. In Section 4, we explain the necessary characteristics of the NoC architecture required to efficiently execute the backpropagation algorithm on the heterogeneous platform considered here. Section 5 discusses the experimental results to demonstrate the efficiency of the proposed NoC over the traditional wireline counterparts. Finally, Section 6 concludes the paper by summarizing the findings and pointing toward future directions.

2. Related Work and Novel Contribution

NoC-enabled homogeneous CMP architectures targeting neuroscience applications have already been explored. For instance, a massively parallelized CMP platform incorporating a customized NoC architecture was used to implement spiking neural networks [13]. Multicast-aware mesh NoC architectures have been proposed for reconfigurable neural networks [14][15]. Due to the highly parallelizable nature of the neural networks, these applications have already been demonstrated to be more efficient on discrete GPU systems rather than traditional multi-CPU CMPs [4][16]. The design of a Commodity Off-The-Shelf (COTS) system for HPC targeting deep learning was proposed in [17]: a cluster of GPU servers with Infiniband interconnects and a message-passing interface (MPI) show promise over large CPU-only based systems. Recently, the architecture of a machine learning "supercomputer" [16] that achieves higher performance and lower energy dissipation than a modern GPU-based system was also proposed. The system relies on a multichip design, where each node is significantly cheaper than a typical GPU while achieving comparable or higher number of operations per unit time in a smaller package.

Prior work on discrete GPU platforms has been focused on improving the system performance by enhancing their NoC architectures [18][19][20]. In a GPU system, processes executed in each GPU core are usually independent of other GPUs' processes, resulting in low inter-GPU communication [18]. Typically, GPUs only communicate with a few shared memory controllers (MC), causing a *many-to-few* traffic pattern (i.e., many GPU cores communicating with a few MCs) [18][19][20]. In this case, MCs can potentially become traffic hotspots and lead to performance bottlenecks. Prior research demonstrated that suitable placement of the MCs can help alleviate the associated traffic congestion [18][19]. To prevent traffic imbalance among the links, a checkerboard mesh NoC with a suitable routing strategy was recently proposed [18]. Similarly, to avoid link traffic overlap between requests and their replies, an asymmetric virtual channel partitioning and monopolization technique for the discrete GPU NoC was proposed [19]. The advantage of using a clustered mesh NoC (with 4 L1s per cluster) over non-clustered mesh and crossbar architectures (i.e., all L1s in a single cluster) for discrete GPU systems was demonstrated [20].

As explained above, backpropagation involves heavy CPU-GPU communication that is best suited for a NoC-enabled heterogeneous CPU-GPU CMP platform rather than a traditional discrete-GPU system with expensive off-chip CPU-GPU data transfers [8]. Due to the differences in the thread-level parallelism of CPUs and GPUs, the NoC employed for heterogeneous systems is expected to handle traffic patterns with varying Quality of Service (QoS) constraints [21]. CPUs are highly sensitive to the memory access times and hence, communications involving CPUs require low-latency data exchanges. On the other hand, GPU communication demands high bandwidth [21]. Modern NoC designs for discrete GPU systems typically attempt to only maximize the overall bandwidth of the system. Consequently, these NoC designs are unsuitable for heterogeneous CMP architectures incorporating multiple CPUs and GPUs on the same die. It has been shown that the shared memory resources in a heterogeneous system are often monopolized by the GPUs, leading to significant degradation in CPU memory access latency and high execution time penalties [22]. Hence, an efficient on-chip network designed for heterogeneous CMPs should balance and fulfill the different QoS standards required for both CPU and GPU communications.

The NoC design for CPU-GPU heterogeneous systems have yet to be studied thoroughly. A system-level discussion regarding the NoC design for CPU-GPU heterogeneous architectures was presented in [21]. However, this work only considered a ring interconnect that is known to be inefficient for large-scale systems.

A virtual channel partitioning scheme was proposed in order to achieve low-latency CPU-related memory accesses in the presence of largely latency-insensitive GPU communication [23]. However, this strategy can affect the GPU throughput by partitioning the physical resources. As we will explain later, our work employs wireless links dedicated for CPU-MC communication to avoid such network contention without affecting GPU throughput.

Thus far, most of the NoCs targeting discrete GPU systems are based on conventional wired NoC architectures. The achievable performance benefit from the proposed strategies is restricted due to the inherent limitations associated with these NoCs. Traditional wireline NoCs (such as mesh) use multi-hop, packet-switched communication that leads to high network latencies [9]. To overcome these limitations, small-world network-inspired wireless NoC architectures have been proposed [10][11]. Indeed, by employing a few long-range wireless shortcuts, these architectures enable low latency communication even among the computing cores that are physically far apart. Previous works [24][25][26] have investigated the feasibility of the on-chip wireless communication. The viability of on-chip wireless communication has been demonstrated through prototypes [12]. Photonic network architectures for high bandwidth on-chip communication have also been proposed [27]. However, on-chip photonic networks have yet to be successfully integrated into large-scale systems. In addition, a recent study on emerging on-chip interconnects concluded that the Radio-Frequency (RF) links, e.g., mm-wave wireless and surface-wave interconnects, are more power and cost efficient than on-chip optical links [28]. Between the two RF interconnects, on-chip wireless technology is more mature and fully CMOS compatible [11][28].

In this work we improve the state-of-the-art by presenting a hybrid (wireline + wireless) on-chip interconnection architecture that can meet the communication demands of a heterogeneous CMP platform consisting of both CPU and GPU cores.

3. Backpropagation Benchmark

In this section, we briefly outline the features of the CUDA-based implementation of the backpropagation algorithm from the Rodinia benchmark suite [5]. We describe the neural network model and its instantiation for the image recognition task[1] that is part of the benchmark; how to make predictions with given network weights; and how to learn the network weights from a given training set using the backpropagation learning algorithm.

Neural network model: A neural network is a graph with nodes and edges. It can be seen as a non-linear function approximator that is parameterized by the weights associated with the edges (commonly referred to as "network weights"). The Rodinia backpropagation benchmark implements a fully-connected feed-

[1]For the image recognition task associated with the benchmark, the input corresponds to the facial image of a person and the output is a discrete label identifying the person.

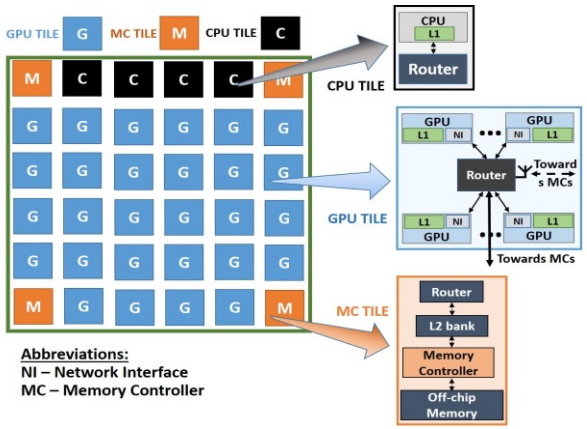

Fig. 1. Illustration of a heterogeneous architecture incorporating CPUs, GPUs, and MCs. Each tile contains a network router enabling NoC interconnections. Because of L1 concentration, GPU tiles incorporate 4 GPU cores and 4 L1s in each tile.

forward neural network with a hidden layer. It includes an input layer *I* consisting of input units (say *m*); hidden layer *H* consisting of hidden units (say *n*); output layer *O* consisting of output units (say *p*); and directed edges connecting each input unit and hidden unit pair, and each hidden unit and output unit pair (total of *m*n* + *n*p* edges). The number of input units for the considered image recognition task is same as the number of pixels in the image, and the number of output units is only one (a discrete label). There is a weight *w* (parameter) associated with each edge of the network, and each node takes the weighted sum of its inputs and computes the output using a non-linear *activation function* γ. This computation is highly data-intensive and easily parallelizable using GPU cores.

This neural network model is very expressive as every bounded continuous function can be approximated with little error [29]. Since the backpropagation training works only when each and every function node is differentiable, the activation function γ should be chosen accordingly. The Rodinia benchmark employs the sigmoid function:

$$\gamma(v) = \frac{1}{(1+e^{-v})} \quad (1)$$

where *v* is the weighted sum of the input edges at a function node.

Prediction with given network weights: Given an input image *x* and network weights *W*, we propagate the input forward through the network, compute the output activation of every hidden unit; and then, compute the output of the network to predict output $\hat{y}$.

Learning network weights: Given a set of *T* training input-output pairs (x_k, y_k), we want to learn the weights of the network such that the error of the predicted outputs of the neural network (as described above) is very small. The benchmark employs an iterative optimization training algorithm (backpropagation) to learn the network weights. At each iteration, for each input example x_k from the training data, we make a prediction $\hat{y}_k$ by making a forward pass through the network using the current network weights (as described above); we then compare the predicted output $\hat{y}_k$ with the correct output y_k to see if there is an error, and if there is an error, we make a backward pass through the network to compute the gradient vector δ_k. Finally, we perform a weight update using the cumulative gradient as follows:

$$\delta = \sum_{k=1}^{T} \delta_k \quad (2)$$

$$W = W + \eta * \delta \quad (3)$$

where η is the learning rate and *W* is the weight vector corresponding to network weights. It is clear that during both the forward and backward passes, a lot of parallelism exists within the computation between two layers that can be exploited while designing hardware accelerators. Moreover, the data propagation between the layers causes significant amount of network traffic.

The computation and communication patterns between any two consecutive layers of a neural network will follow patterns similar to that observed in the Rodinia benchmark suite described above. Hence, the findings of this work are generalizable for *any* neural network with multiple layers.

4. NoCs for Heterogeneous Platforms

In CPU-GPU heterogeneous manycore architectures, the NoC mainly handles *many-to-few* communication patterns. Each processing core in the system maintains its own L1 cache. The L1 caches mainly exchange data with a limited number of memory controller (MC) blocks. Also, inter-GPU traffic is insignificant in comparison to the L1 to MC communication volume [18][20]. Each MC incorporates a Last Level Cache (LLC) and a mechanism to access the main memory. Fig. 1 illustrates a CPU-GPU heterogeneous architecture[2] with 4 CPUs, 28 GPUs, and 4 MCs. As shown in Fig. 1, in order to perform efficient many-to-few communication, we attach multiple GPUs to a single router (called concentration) [20]. With concentration, multiple L1 banks connect to a single router, which is then connected to the MC blocks. The concentration lowers the inter-router hop count between the L1s and MCs and hence reduces the network latency. Moreover, concentration lowers the number of NoC routers needed, leading to lower area overheads and reduced wiring complexities [20].

In a heterogeneous CMP, the traffic requirements vary depending on the type of the nodes involved in the data exchange. The CPU-MC communications are primarily latency-sensitive while the GPU-MC communications are more throughput-sensitive [21]. In CPUs, long waits for memory accesses lead to stalled processor cycles resulting in execution time penalties. On the other hand, GPU's low-cost context switching makes GPUs less susceptible to GPU-MC communication latency. However, each GPU core consists of multiple thread execution units, requiring large streams of data exchanges between the GPU and MC, leading to high throughput requirements [21]. Considering the above-mentioned facts, an NoC designed for heterogeneous CPU-GPU systems must be optimized to ensure that the CPU-MC communication latency is minimized while the overall NoC throughput is maximized. Hence, in this work, we consider these two objectives and jointly optimize them while designing the overall interconnection architecture.

4.1 Mesh NoC

It was shown that on a mesh NoC for handling *many-to-few* communication patterns, placing the MCs closer to the middle rather than along the chip edges reduces traffic congestion in the links and ensures better overall NoC throughput [18]. However, this mesh NoC was designed for a homogeneous GPU-based system. In the case of the heterogeneous architecture, which is the focus of this work, we need to consider the placement of the CPU cores in addition to the placement of the MCs to achieve both low latency CPU-MC communication and high NoC throughput. However, as we will show in Fig. 7, even in a mesh optimized for both CPU and MC placements, there exist a few links that are heavily utilized when compared to the rest of the links present in the NoC. During high traffic, such links will become bandwidth bottlenecks, negatively affecting the overall system performance.

[2] It should be noted that the tile locations in this heterogeneous architecture figure are just for illustration purposes. They are not optimized for any specific performance metric.

The presence of these bandwidth bottlenecks can be attributed to the multi-hop nature of the mesh NoC architecture, which leads to high traffic aggregation in the intermediate routers and links.

In order to address the inherent multi-hop nature of mesh NoCs, design of wireless NoCs (WiNoCs) has been proposed [10][11]. The salient feature of the WiNoC is that the wireless links establish single-hop shortcuts between physically distant cores, thereby improving the hop-count and subsequently, the latency, throughput and energy dissipation of the whole system. Hence, our aim is to design a customized low-hop WiNoC targeted for the heterogeneous architecture under consideration.

4.2 Proposed Hybrid NoC

In this section, we highlight the design principles of a hybrid NoC architecture comprising of both wireline and wireless links customized for the CPU-GPU heterogeneous computing platform. We call this proposed NoC a Wireless-enabled Heterogeneous NoC (WiHetNoC). In this network, we intend to use single-hop wireless links between the CPUs and MCs. The GPU-MC communication is handled through a combination of wireline and wireless links that are tailored to the traffic pattern. Aside from enabling low latency CPU-MC data exchanges, the use of dedicated wireless links for CPU-MC communication makes the WiHetNoC design agnostic of the CPU and the MC placements. This is because the wireless links are able to guarantee direct single-hop communication regardless of the physical distance between the transceivers as long as they are within the communication range.

In the following subsections, we describe the overall network design methodology for the WiHetNoC. We also describe the insertion of wireless links as long-range shortcuts and the communication protocols that are employed in WiHetNoC.

4.2.1 Optimizing Network Connectivity

To maximize the throughput of a NoC with a given number of links, the network connectivity should be established such that the average inter-router hop count is minimized while the system is free from bandwidth bottlenecks.

For an NoC with R routers and L links, we can find U_k, the expected utilization of a link k by using the following equation:

$$U_k = \sum_{i=1}^{R}\sum_{j=1}^{R} f_{ij}p_{ijk},\ p_{ijk} = \begin{cases} 1, & if\ i,j\ communicate\ along\ link\ k \\ 0, & otherwise \end{cases} \quad (4)$$

where, p_{ijk} indicates which links are used for the communication from router i to router j. The p_{ijk} value can be found by using the network connectivity and NoC routing protocol. The value f_{ij} denotes the frequency of interaction between routers i and j.

In our WiHetNoC, we establish the network connectivity such that both the mean link utilization ($\overline{U}$) and the standard deviation among the link utilizations (σ) are minimized. Essentially, our main objective is to find d^* in (5).

$$d^* = \arg\, min_{d\in D}\, f(\overline{U}(d), \sigma(d)) \quad (5)$$

where D is the set of all wireline connectivity possible under given constraints, and d^* is the connectivity obtained from minimizing $\overline{U}$ and σ. For the sake of completeness, we show the formulae for determining $\overline{U}$ and σ below.

$$\overline{U} = \frac{1}{L}\sum_{k=1}^{L} U_k = \frac{1}{L}\sum_{k=1}^{L}\left(\sum_{i=1}^{R}\sum_{j=1}^{R} f_{ij}p_{ijk}\right) = \frac{1}{L}\sum_{i=1}^{R}\sum_{j=1}^{R}\left(f_{ij}\sum_{k=1}^{L} p_{ijk}\right) = \frac{1}{L}\sum_{i=1}^{R}\sum_{j=1}^{R} f_{ij}h_{ij} \quad (6)$$

$$\sigma = \sqrt{\frac{1}{L}\sum_{k=1}^{L}(U_k - \overline{U})^2} \quad (7)$$

Here, h_{ij} denotes the minimum distance in number of hops from router i to router j with the given network connection. Intuitively, h_{ij} is equal to the total number of links used in the communication between router i and router j.

As it can be observed from the above equations, $\overline{U}$ is directly proportional to the traffic weighted hop count (given by $\sum_i^R\sum_j^R f_{ij}h_{ij}$). Thus, minimizing $\overline{U}$ also minimizes the inter-router hop count, leading to high network throughput. On the other hand, minimizing σ ensures that the link utilizations in the WiHetNoC are well balanced. Unlike the mesh NoC in which only a few MC links are highly active, all the incoming and outgoing links are almost equally utilized in WiHetNoC. This in turn ensures that the NoC is free from the bandwidth bottlenecks.

Since minimizing both $\overline{U}$ and σ together is a multi-objective optimization (MOO), here we employ the Archived Multi-Objective Simulated Annealing (AMOSA) [30] to solve (5). AMOSA is a simulated annealing (SA) based algorithm in which the optimization process is guided with the help of an archive of solutions [30]. In AMOSA, during each optimization step, a perturbation is created in one of the archived solutions to generate a new configuration. Depending on the comparative quality of this new configuration over all the solutions in the archive, AMOSA then updates the archive [30]. Thus, on completion of AMOSA, we obtain a set of archived candidate configurations.

We note that although the AMOSA-based optimization has been used in this work as an example, any other MOO technique can be utilized in place of AMOSA. Exploring the best possible MOO for heterogeneous architectures is out of the scope of this paper and is a part of our future investigations.

In this work, our aim is to create an application-specific WiHetNoC architecture by optimizing the traffic weighted hop count. Hence, the overall connectivity is irregular in nature, i.e., all routers and link characteristics are entirely dependent on the application's traffic pattern. However, while establishing the link connectivity in the WiHetNoC, we need to follow certain restrictions. First, we limit the average number of inter-tile communication ports per router (K_{avg}) to four so that the WiHetNoC does not introduce any additional router port overhead when compared to a conventional mesh. Next, we need to restrict the maximum number of ports in a router (K_{max}) so that no particular router becomes unrealistically large. It was already noted that the MCs in a heterogeneous NoC are traffic hotspots with heavy volumes of incoming and outgoing messages. Increasing K_{max} allows the number of router ports attached to an MC to increase, and hence improves the MC router bandwidths. However, high K_{max} values can lead to large routers, which result in high network energy consumptions. Moreover, large routers make the whole system highly vulnerable to failures. Consequently, in this

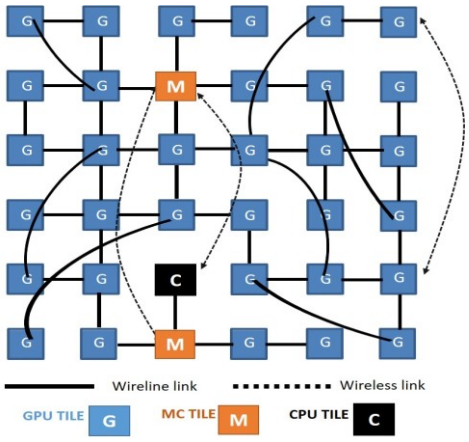

Fig. 2. Illustration of WiHetNoC Connectivity.

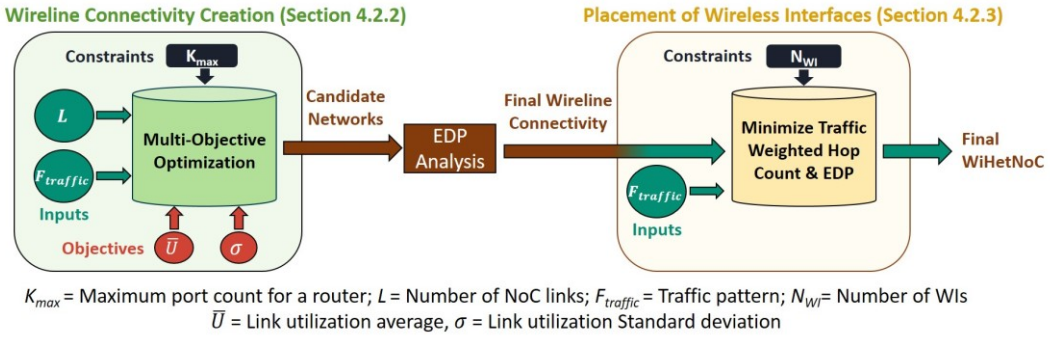

Fig. 3. WiHetNoC design flow. Required input parameters and the objective functions optimized in the creation of WiHetNoC are also shown.

work, we consider a K_{max} range of 4 to 7. First, for each K_{max} value, we create a candidate network set through AMOSA for minimizing both the mean link utilization and the standard deviation among the link utilizations. Then, among all these candidate networks, we choose the network with the lowest energy delay product (EDP) as the optimum wireline connectivity for the WiHetNoC.

4.2.2 Wireless Link Placement

In WiHetNoC, the CPU-MC communication is handled using dedicated millimeter (mm)-wave and sub-THz on-chip wireless links operating in the 10-220 GHz range. Moreover, as we illustrate in Fig. 2, in this WiHetNoC there are several long wireline interconnections. As these links are extremely costly in terms of power and delay, we employ the wireless links to connect the GPU and MC routers that are separated by long distances. In practice, depending upon the available wireless resources, we can only make a limited number of the longest links wireless, while the other links need to remain wireline. Current state-of-the-art WiNoCs employ wireless channels working in the mm-wave range of 10-100 GHz [11]. In order to achieve simultaneous wireless transmissions and maximize the usage of the wireless medium, we need to have multiple non-overlapping wireless channels. It has been shown that it is possible to create three non-overlapping channels working in the 30, 60 and 90 GHz within the mm-wave range [11]. However, considering the data-intensive application at hand, to increase the wireless throughput, we increased the wireless channel range to the Sub-THz frequencies. By enhancing the frequency range of operation, we are able to create two more additional non-overlapping channels working at 140 and 220 GHz. It should be noted that by using the current CMOS technology both the frequency range and number of channels could be increased further. However, the proposed design methodology presented in this work is oblivious to these physical design parameters, i.e., the number of wireless channels can be increased or decreased without modifying the proposed algorithm. Using these five channels we overlay the wireline connectivity with the wireless links such that a few routers get an additional wireless port. The wireless ports have a wireless interface (WI) tuned to one of the five different frequency channels.

Given the total number of WIs allowed (N_{WI}), we use a WI placement strategy that focuses on minimizing traffic-weighted hop-count [32]. Following this methodology and by varying N_{WI}, we find both the optimum number of wireless interfaces and the best locations in the WiHetNoC. The optimum value of N_{WI} is discussed later in Section 5, experimental results.

To summarize, the overall design flow to create the WiHetNoC architecture is shown in Fig. 3.

4.2.3 Components of the Wireless Interface

The two principal components of a wireless interface are the antenna and the transceiver. WiHetNoC uses a metal zigzag antenna that has been demonstrated to provide the best power gain with the smallest area overhead [11]. A detailed description of the transceiver circuit is out of the scope of this paper. The wireless interface is completely CMOS compatible and no new technology is needed for its implementation. In the 28nm technology node, for data rates of 16 Gbps, wireless links dissipate 1.3 pJ/bit over a 20mm communication range.

4.2.4 Communication Protocols

In this section, we explain the routing and the wireless medium access control (MAC) protocols that are adopted for the proposed WiHetNoC. The proposed NoC principally has an irregular application-specific topology and requires a topology agnostic routing method. We follow ALASH (Adaptive Layered Shortest Path) routing [10]. ALASH is built upon the layered shortest path (LASH) algorithm [31]. The ALASH protocol improves the LASH layering function by considering the expected traffic patterns. We follow the priority layering function explained in [10]. Priority layering allocates as many virtual layers as possible to source-destination pairs with high traffic intensities. This improves the adaptability of messages under high traffic intensities by providing greater routing flexibility. We employ a distributed MAC protocol to resolve the channel access contention among the wireless nodes present in WiHetNoC [32].

5. Experimental Results and Analysis

We employ gem5-gpu, a heterogeneous full system simulator to obtain processor- and network-level information [33]. The gem5-gpu combines two well-known simulators; Gem5, a manycore CPU simulator and GPGPU-sim, a detailed GPGPU simulator. The gem5-gpu provides flexible network configurations and supports inter-CPU-GPU cache coherence protocols. The gem5-gpu also provides a customizable interconnection model through the Garnet network[34]. We have modified this Garnet network topology to implement the WiHetNoC architecture. We use gem5-gpu in the full-system simulation mode. In the full-system simulations we consider a NVIDIA Fermi-like architecture for GPU cores and the

Parameters	Configuration
GPU / Shader Core clock	0.7GHz / 1.4GHz
SIMT width	8
GPU Private L1 I cache	64kB
GPU Private L1 D cache	64kB
CPU Clock	2.5GHz
CPU Private L1 I cache	64kB
CPU Private L1 D cache	64kB
Shared L2 cache size	1MB per MC
DRAM	2GB

Table 1. System configurations

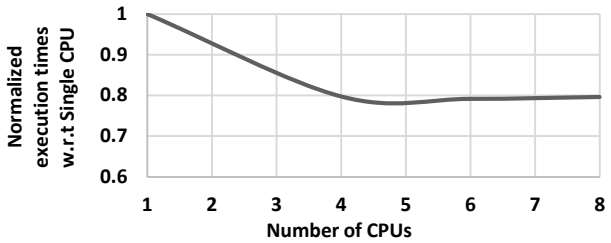

Fig. 4. Execution times of the considered backpropagation benchmark with varying number of CPUs.

standard x86 architectures for CPU cores. We employ MESI two-level cache coherence protocol. Each CPU and GPU streaming multiprocessor (SM) is provided with private L1 data and instruction caches. The considered memory system also incorporates four LLCs that are shared among all the CPUs and SMs. Table 1 provides detailed configurations of the architecture considered in this work. We use GPUWattch to obtain detailed processor power profiles from the gem5-gpu statistics [35]. We use a generic three-stage router architecture for all NoCs under consideration. The delay of each stage is constrained within one clock cycle. For routers with more than 4 inter-tile router ports, the output arbitration has one more pipeline stage and the delay of this stage is also constrained within one clock cycle. This is accounted for while determining the latency and energy of HetNoC and WiHetNoC. Moreover, it should be noted that the proposed NoC design and optimization methodology is valid for router architectures with any number of stages.

First, we investigate the optimum number of CPUs for the benchmark at hand. Fig. 4 shows the application run time with different number of CPUs in the system with a traditional mesh NoC. We observed a similar trend with the WiHetNoC. It is evident that beyond four CPU cores there is no improvement in the run time. Hence, we consider four CPU cores in our experiments. This trend in execution time can be attributed to the fact that for the given application, the CPU execution portion does not scale well beyond four parallel threads. On the other hand, the GPU computations are highly parallelized and the optimum number of streaming multiprocessors (SMs) for the backpropagation application is mainly dependent on the given input problem size. Hence, without loss of generality, in this work, we consider a heterogeneous architecture with 112 SMs. For these SMs, we consider a concentration factor of two (i.e., two SMs and their associated L1s are connected to one GPU router). Thus, the considered architecture consists of 56 SM tiles, 4 CPUs and 4 MCs. This gives rise to a system size of 64 tiles, which are arranged in an 8 × 8 configuration.

5.1 Characterization of Mesh NoC

In this section, we analyze the characteristics of the conventional wireline mesh NoC architecture running the backpropagation application. Following the discussions in Section 4, for a mesh NoC with a given number of tiles, we identify the positions of the CPUs and MCs such that the CPU-MC communication latency is minimized while the overall NoC throughput is maximized to enable efficient GPU-MC data exchange (under the *many-to-few* communication pattern).

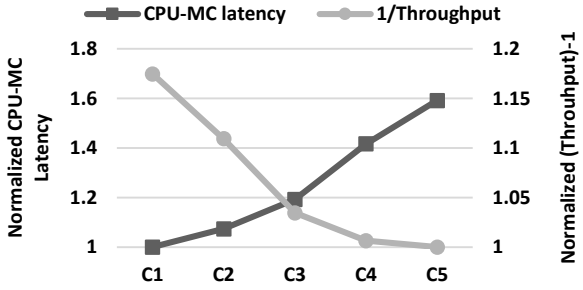

Fig. 5. Latency and throughput trade-off among the candidate configurations (C1-C5) for the mesh NoC.

5.1.1 Determining CPU & MC Placements in Mesh

We employ the AMOSA algorithm (mentioned in Section 4.2) to determine the optimal positions of CPUs and MCs in a mesh NoC (with GPU nodes occupying the remaining tiles) to jointly optimize CPU-MC communication latency and the overall NoC throughput. On completion of the AMOSA optimization, we obtain a final set of candidate configurations, each with a different (CPU-MC communication latency, NoC throughput) pair. From Fig. 5, we can observe that there is a latency-throughput trade-off among the five mesh NoC candidate configurations, denoted as C1-C5 (the configuration with the best CPU-MC communication latency has the worst NoC throughput and vice versa). In order to observe the effect this trade-off has on the system performance, we execute the backpropagation application with all different candidate configurations. We also consider two non-optimized mesh configurations to provide a comparative performance evaluation with respect to the optimized mesh candidate solution set. In these two non-optimized configurations the MCs and the CPUs are simply placed along the edges. The first configuration is called *TB* where the CPUs and the MCs are located on two opposite edges, e.g., top and bottom of the die. In the second configuration, denoted as *BOT*, all CPUs and MCs are placed at the bottom of chip. Fig. 6 shows the application run times of the following seven NoC configurations; five optimized mesh candidate configurations (denoted as C1-C5), *TB* mesh and *BOT* mesh. It can be observed from Fig. 6 that the C3 configuration achieves the least application run time. Hence, in all further analyses, we consider the C3 configuration for mesh NoC. We should also note that the two non-optimized configurations *TB* and *BOT* have higher run times than all of the optimized configurations. However, the execution time improvement of optimized mesh is limited compared to *BOT* mesh since we are only optimizing the locations of the MCs and CPU tiles here. As we will discuss in Section 5.3, we are able to greatly increase the achievable gain over *BOT* mesh by using a WiHetNoC (Section 4.2) designed with AMOSA. This is due to the fact that in the design of WiHetNoC we also optimize the underlying network topology and hence the achievable performance gain is much more.

5.1.2 Link Utilization in Mesh

Figure 7 shows the locations of the CPU and MC tiles in the final optimized (C3 configuration) mesh NoC. We can observe from Fig. 7 that in the optimized mesh NoC, the MCs and CPUs are clustered in the middle so that the CPU-MC communication latency is minimized. Moreover, placing the MCs closer to the middle

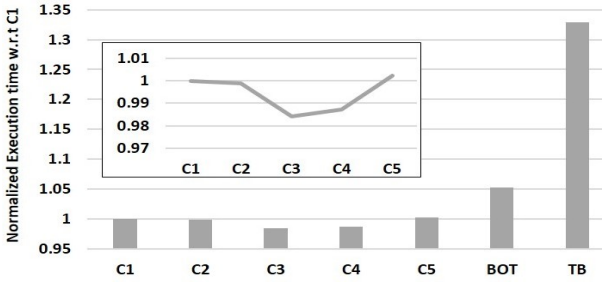

Fig. 6. Application runtimes for candidate configurations (C1-C5) of the optimized mesh NoC along with two non-optimized configurations (*BOT, TB*).

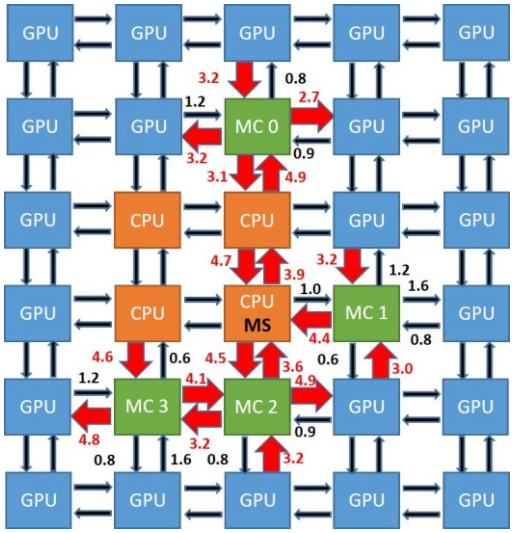

Fig. 7. The central 6×5 portion of the optimized Mesh NoC indicating the link utilizations and the locations of CPUs and MCs. All link utilizations are normalized with respect to the mean link utilization. The red arrows indicate the bandwidth bottlenecks whose utilization is at least 100% more than the mean.

ensures that handling of GPU-MC traffic is distributed among multiple MC ports, leading to higher NoC throughput.

In Fig. 7, we also show the link utilizations in the optimized mesh NoC. All link utilizations are normalized with respect to the mean link utilization. It is evident that even in this optimized NoC a few links are more heavily utilized when compared to the rest of the links. This happens due to the XY routing mechanism traditionally considered for mesh NoCs. The incoming vertical and outgoing horizontal links associated with the MCs have 300% to 400% higher traffic densities than the overall average.

In order to alleviate the traffic congestion caused by aggregation of traffic along the mesh NoC links, one can also adopt a combination of minimal XY and minimal YX routing as proposed in [19]. However, such an approach cannot eliminate the bandwidth bottlenecks from the optimized mesh NoC under the heterogeneous computing induced traffic patterns. To elaborate more, Fig. 8 shows the average traffic weighted hop count and the standard deviation among the link utilizations for all the following NoC configurations: optimal mesh, *TB* mesh, *BOT* mesh, and four different WiHetNoC candidate architectures (these WiHetNoC candidates are explained later in detail). As it can be observed from this figure, the optimal mesh NoC (with both XY and the XY+YX routing protocols) shows lower standard deviation among the link utilizations and lower traffic weighted hop count than the non-

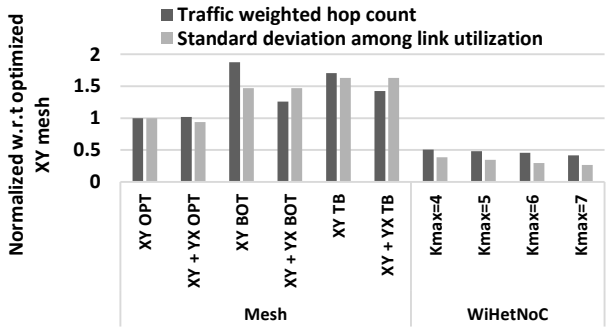

Fig. 8. Traffic-weighted hop count and standard deviation among the link utilizations for the optimized mesh (denoted as OPT), TB mesh, BOT mesh, and WiHetNoCs (for each considered K_{max}). For mesh NoCs we show the results for both XY and XY+YX routing schemes.

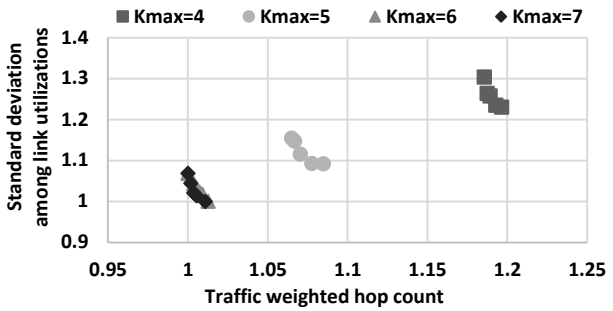

Fig. 9. Normalized traffic-weighted hop count and standard deviation among the link utilizations of various candidate wireline configurations for WiHetNoC (K_{max} values ranging from 4 to 7).

optimized mesh configurations. However, when compared with WiHetNoC candidates, both the standard deviation and the traffic weighted hop count of the optimized mesh NoC are at least 2X higher and this indicates the presence of traffic hotspots that can lead to bandwidth bottlenecks.

It can be observed from Fig. 7 that in the optimized mesh NoC, since the MCs are placed around the CPU cluster, the CPU tile routers also act as intermediate routers forwarding the GPU-MC traffic. With continuous streams of data flowing from MCs towards GPUs, the above-mentioned traffic forwarding causes traffic congestion in the CPU tile routers leading to undesired increase in the CPU-MC communication latency. To resolve this issue, the MCs can be placed away from the CPUs such that the CPU tiles are not involved in the GPU-MC traffic forwarding. However, due to the lack of long-range shortcuts in mesh NoC, placing the MCs away from the CPUs increases the number of hops required for CPU-MC communication leading to network latency penalties. Hence, our target is to design an application-specific heterogeneous NoC architecture with suitable long-range shortcuts to improve both CPU-MC latency and the overall NoC throughput. Since the wireless links make the CPU-MC latency agnostic of their placements, we keep the CPUs at the center of the system like the mesh NoC and distribute the four MCs to the center tiles in each of the four quadrants of the system. All the other tiles are occupied by the GPU cores surrounding the CPUs and MCs. The whole system is integrated using the WiHetNoC framework. In the following section we determine the various network parameters of this WiHetNoC.

5.2 Determining WiHetNoC Parameters

In this section, we determine the overall architecture and the network parameters of WiHetNoC.

5.2.1 Router Port Upper Bound

In the WiHetNoC design process, we first find the optimum value of k_{max} (maximum number of inter-tile communication ports in a router). As we discussed in Section 4.2.1, we start with a k_{max} range of 4 to 7. For each k_{max} value we use the above-mentioned AMOSA optimization to find a final candidate solution set. Fig. 9

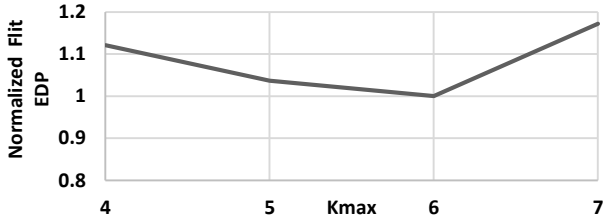

Fig. 10. Variation in network EDP for different router port upper bounds (K_{max}).

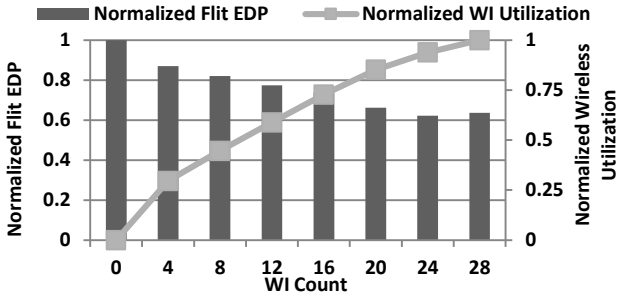

Fig. 11. EDP and Wireless utilization for various WI counts.

shows the mean link utilization ($\overline{U}$) and the standard deviation among the link utilizations (σ) for all the solutions in the final candidate set corresponding to each k_{max} value. All the values in this figure are normalized with respect to the final WiHetNoC configuration. As it can be observed from the graph, as the k_{max} value is increased, the $\overline{U}$ and σ values are lowered. This happens because with higher K_{max} values, more inter-router connections are allowed in each MC, leading to lower average hop counts between GPUs and the MCs. It can also be observed that the solutions corresponding to K_{max}=6 and K_{max}=7 overlap with each other while the solutions corresponding to K_{max}=5 and K_{max}=6 can be clearly distinguished from each other (in Fig. 9). This demonstrates that increasing K_{max} leads to diminishing gains in hop count reductions and there is no gain in exploring beyond K_{max}=7. As explained in Section 4.3.1, the final WiHetNoC wireline connectivity is identified from the candidate networks by comparing their network Energy-Delay-Products (EDP). Average message latency and average message energy values are used in this EDP computation. Fig. 10 shows the EDPs of the optimal networks corresponding to each K_{max}, (thus a total of four optimal networks are shown corresponding to K_{max} values ranging from 4 to 7). From Fig. 10 it is evident that the optimal value for k_{max} is 6. Beyond this value of K_{max}, the EDP increases steadily due to higher router energy consumptions without significant gains in network latency and hop count. Hence, we select the network corresponding to K_{max}=6 as the optimal wireline connectivity for WiHetNoC.

In Fig. 8, we compare the average traffic-weighted hop-count and the standard deviation among the link utilizations for various mesh configurations and the four optimal WiHetNoC architectures (for the four considered K_{max} values). From this figure, we can observe that WiHetNoC enables at least 50% traffic weighted hop count reduction over mesh. Moreover, the standard deviation among the link utilizations for WiHetNoC is low indicating that the traffic is well distributed among WiHetNoC links and hence, the opportunity for experiencing bandwidth bottlenecks is highly reduced in WiHetNoC when compared to mesh.

5.2.2 Number of WIs

Next, we identify the number of WIs needed for the GPU-MC communication in the WiHetNoC architecture. As mentioned

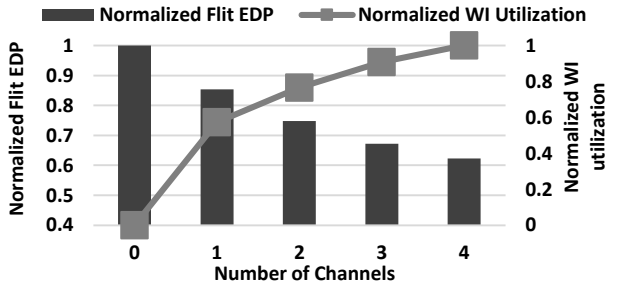

Fig. 12. EDP and WI utilization with various number of channels.

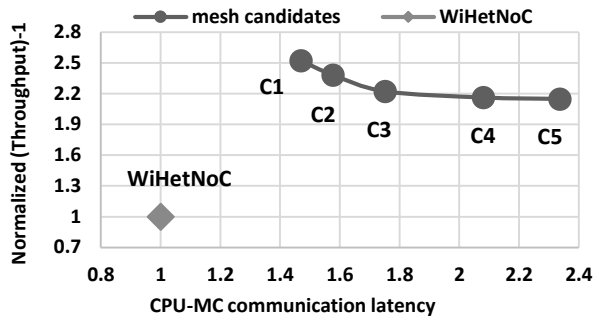

Fig. 13. CPU-MC communication latency and inverse of overall NoC throughput for various candidate configurations (C1-C5) of the mesh NoC and the optimized WiHetNoC.

earlier, we are able to create five non-overlapping channels. We dedicate one channel to achieve single-hop CPU-MC communications. Hence, four wireless channels are available for GPU-MC communication.

Fig. 11 shows the variation in EDP and wireless utilization observed with varying WI counts. The wireless utilization parameter represents the percentage of total messages that are using the wireless channels. As observed from this figure, the EDP initially reduces as the WI count increases as higher number of wireless shortcuts improves the wireless utilization and hence lowers the overall network latency. However, beyond a WI count of 24, with more than six WIs allocated on a single wireless channel, the MAC overhead (and hence the channel access latency) starts to increase [27]. This in turn increases the network EDP for the WiHetNoC when more than 24 WIs (Fig. 11) are used. Hence, in our WiHetNoC, we employ 24 WIs for GPU-MC communication (4 channels are used with 6 WIs operating on each channel). Fig. 12 shows the effects of adding wireless channels for GPU-MC communication on the performance of the WiHetNoC. With increasing number of wireless channels, the amount of data using wireless medium increases and the overall EDP improves. However, the enhancement in wireless utilization and subsequent improvement in EDP slows down beyond a certain number of wireless channels. For the 64-tile system size, increasing the number of wireless channels beyond 4 does not enhance the system performance noticeably as the opportunity for more wireless utilization diminishes.

Since each WI transceiver occupies an area of 0.25mm^2, the additions of WIs introduce a total of 1.82% silicon area overhead for a die with dimensions of 20 mm×20mm.

5.2.3 Characteristics of WiHetNoC

Fig. 13 compares the CPU-MC communication latency and network throughput achieved with the optimized WiHetNoC against all the solutions available in the final candidate set for the optimized mesh (explained earlier in Section 5.1). From this graph,

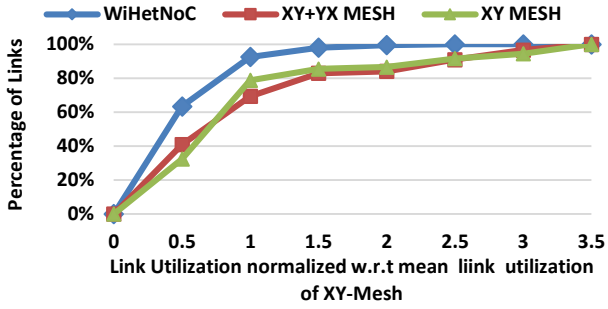

Fig. 14. Cumulative Distribution Function (CDF) of link utilizations for the WiHetNoC, XY mesh, and XY-YX mesh architectures.

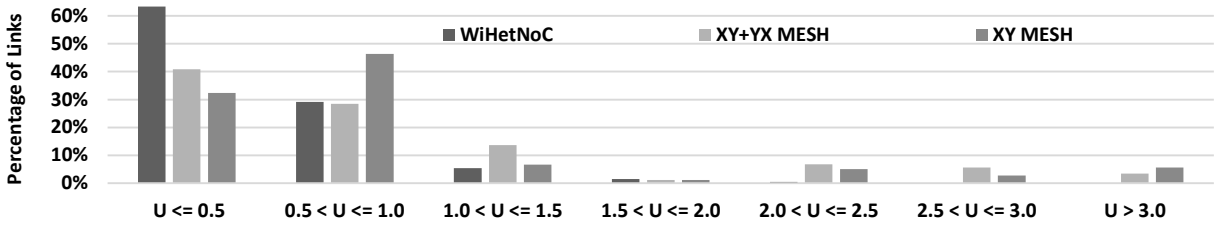

Fig. 15. Probability Density Function (PDF) of the link utilizations in WiHetNoC, XY mesh, and XY+YX mesh architectures.

we can observe that WiHetNoC achieves both a higher NoC throughput and a lower CPU-MC communication latency than all the optimized mesh candidate solutions with similar implementation overhead. The WiHetNoC improves the throughput by a factor of 2 compared to the optimized mesh configuration (C3). This demonstrates the effectiveness of the proposed methodology in designing NoC architectures for heterogeneous platforms with many-to-few communication patterns.

In Fig. 14 and Fig. 15, we show the Cumulative Distribution Function (CDF) and Probability Density Function (PDF) of the link utilizations for the WiHetNoC and the mesh architectures. The utilizations in this figure are normalized with respect to the mean link utilization observed in mesh NoC with XY routing (U=1 represents this mean utilization). For both XY and XY+YX routing schemes, 15% of the optimized mesh NoC links have at least 2x higher utilization when compared to the mean utilization. The XY+YX routing scheme is only helpful in achieving moderate reductions in the number of very highly utilized links. As an example, the percentage of links that have at least 3x higher utilization when compared to the mean is reduced from 6% in XY routing to 4% in XY+YX routing scheme. It is also clear from this figure that for WiHetNoC, more than 90% of the links fall under the mean link utilization of the mesh NoC. Generally, when compared to mesh NoC, the WiHetNoC CDF curve is shifted left indicating a reduction in overall link utilizations, which is obtained through lowered inter-router hop counts. Moreover, as shown in Fig. 15, unlike mesh, WiHetNoC has no links with very high utilizations (no links with U>2). Thus, WiHetNoC is relatively bandwidth bottleneck free.

5.3 Comparative Performance Evaluation

In this section, we present the performance of the WiHetNoC compared to the optimized mesh NoC architecture. In this comparative performance evaluation, we also consider the characteristics of HetNoC: an architecture that uses pipelined long-range metal wires instead of the WiHetNoC's wireless links. Thus, the HetNoC architecture is an exact fully-wireline equivalent of the WiHetNoC architecture.

We first present the network-level analyses showing both the network latency and EDP. Fig. 16 shows the network latency and the EDP of WiHetNoC, HetNoC, and mesh. HetNoC reduces the network latency and EDP by **35%** and **56%** respectively, when compared to the mesh. With the use of long-range shortcuts between physically remote nodes, the HetNoC enables a lower average hop count than the optimized mesh NoC, and hence, achieves significant reductions in intermediate flit counts. However, as we stated in Section 4.3, the long wireline links of the HetNoC suffer from high link latency and energy consumption. In the WiHetNoC architecture, many of these long-range wireline links are replaced with energy efficient wireless links, and hence, WiHetNoC enables **18%** more latency improvement and **31.5%** more EDP improvement when compared to the HetNoC. From Fig. 16, it is evident that the WiHetNoC achieves **47%** lower network latency, and saves the EDP by **70%** compared to the mesh architecture.

Next, we consider the application execution time and the full-system EDP. Fig. 17 shows the execution time and full-system EDP for WiHetNoC, HetNoC, and mesh architectures. The HetNoC achieves **4.5%** execution time improvement over the mesh while the WiHetNoC shows **12.1%** improvement over the mesh NoC. The dedicated wireless channel for CPUs on the WiHetNoC enables a highly efficient data transfer between CPU and MC. Also, the use of wireless shortcuts helps in achieving high-bandwidth and low-latency GPU-MC communications. These benefits translate to a **25%** and **15%** full-system EDP reduction for WiHetNoC when compared to the mesh and HetNoC architectures respectively.

6. Conclusion

In this paper, we have proposed the design of a hybrid NoC-enabled single-chip heterogeneous computing platform for energy-efficiently accelerating an important deep learning kernel. The proposed NoC architecture is able to fulfill the communication requirements of *both* CPU and GPU cores. We also highlight the inherent limitations of a traditional mesh-based NoC in handling the traffic patterns arising from deep learning kernels. By virtue of using single-hop wireless links, the proposed WiHetNoC architecture achieves much better throughput and latency compared to a highly optimized mesh. For the considered backpropagation application, WiHetNoC achieves **25%** lower full system energy-delay-product (EDP) with respect to the mesh and **15%** lower full system EDP when compared to a fully wireline application-specific architecture. It should be noted that this full system EDP improvement comes only from the network level innovation.

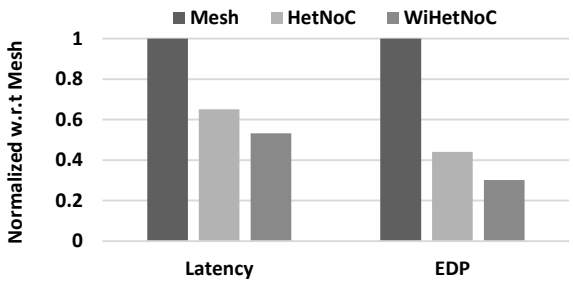

Fig. 16. Network latency and EDP of mesh, HetNoC, and WiHetNoC.

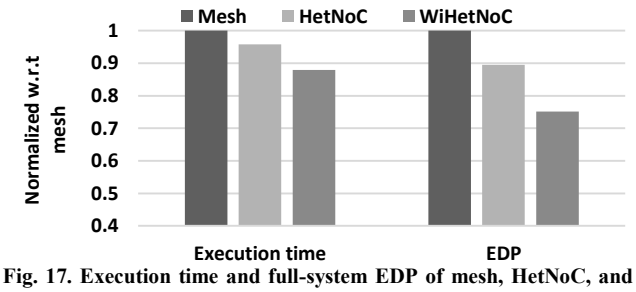

Fig. 17. Execution time and full-system EDP of mesh, HetNoC, and WiHetNoC architectures.

The full system EDP will improve even further when the WiHetNoC design is complemented with suitable core-level task and power management strategies, which is the focus of our future investigation. In addition, we plan to evaluate the characteristics of the proposed architecture for other deep learning workloads.

7. Acknowledgments

This work was supported in part by the US national Science Foundation (NSF) grants CCF-1514269, CCF-1162202, CCF-1314876, CCF-1514206 and CCF-1331804.

8. References

[1] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". *Nature* 521: 436–444. 2015.

[2] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". *Nature* 529, 484–489. 2016.

[3] D. Rumelhard, G. Hinton, and R. Willians. "Learning representations by back-propagating errors". *Nature* 323 (6088): 533–536.

[4] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and Scalability of GPU-Based Convolutional Neural Networks," *Proc. Euromicro Int'l Conf. Parallel, Distributed and Network-Based Processing*, IEEE, 317-324, 2010.

[5] S. Che et al, "Rodinia: A benchmark suite for heterogeneous computing," *in Proc. IEEE Int. Symp. Workload Characterization*, 44–54, 2009.

[6] J. Power et al. "Heterogeneous system coherence for integrated CPU-GPU systems." *In Proc. of the 46th Int'l Symp. on Microarchitecture*, 2013. 457–467.

[7] M.J. Schulte et al, "Achieving Exascale Capabilities through Heterogeneous Computing", *IEEE Micro, vol. 35*, no.4, 26-36, Aug, 2015.

[8] J. Hestness, S.W. Keckler, D.A. Wood. "GPU Computing Pipeline Inefficiencies and Optimization Opportunities in Heterogeneous CPU-GPU Processors". *IISWC*: 87-97, 2015.

[9] U. Y. Ogras and R. Marculescu, ' "It's a small world after all": NoC Performance Optimization via Long-range Link Insertion, ' *in IEEE Trans. on Very Large Scale Integration Systems, Vol.14*, No.7, 2006.

[10] P. Wettin et al., "Design Space Exploration for wireless NoCs Incorporating Irregular Network Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 33, Issue 11,* 1732-1745, 2014.

[11] S. Deb et al., "Wireless NoC as Interconnection Backbone for Multicore Chip: Promises and Challenges", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems, Vol. 2, No. 2*, 228-239, 2012.

[12] S. Deb et al., (2013, December). "Design of an energy efficient CMOS-compatible NoC architecture with millimeter-wave wireless interconnects," IEEE Transactions on Computers, 62(12), pp.2382-2396.

[13] E. Painkras et al., "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits, vol. 48, no. 8*, 1943–1953

[14] V. Dmitri and R. Ginosar. "Network-on-chip architectures for neural networks.". *Proc of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, 135-144.

[15] A. Firuzan, M. Modarressi, and M. Daneshtalab, M. "Reconfigurable communication fabric for efficient implementation of neural networks". *in Proc., of IEEE ReCoSoC*, 1-8. 2015.

[16] Y. Chen et al., "DaDianNao: A Machine Learning Supercomputer," *Proc. 47th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 609–622, 2014.

[17] A. Coates et al., "Deep learning with COTS HPC systems", *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, Georgia, USA, 2013.

[18] A. Bakhoda, J. Kim, and T.M. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerators," *Proc. of 46th Int'l Symp. Microarchitecture*, 457–467, 2013.

[19] H. Jang et al., "Bandwidth-efficient on-chip interconnect designs for GPGPUs" *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE,San Francisco*,CA.1-6.

[20] A. Ziabari et al., "Asymmetric NoC Architectures for GPU Systems" *Proc. Of the 9th International Symposium on Network-on-Chip. Article No. 25*, 2015.

[21] J. Lee, S. Li, H. Kim, and S.Yalamanchilli, "Design Space Exploration of On-chip Ring Interconnection for a CPU-GPU Heterogeneous Architecture," *JPDC*, 2013.

[22] O. Kayiran et al., "Managing GPU concurrency in heterogeneous architectures". *Proc. 47th Int'l Symp. Microarchitecture,* 1–13, 2014.

[23] J. Lee, et al. "Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures." *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18.4 (2013): 48.

[24] J-J. Lin et al., (2007, August). "Communication Using Antennas Fabricated in Silicon Integrated Circuits," *IEEE Journal of Solid-State Circuits*, 42(8), pp.1678-1687.

[25] Y. P. Zhang, Z. M. Chen, and M. Sun, (2007, October). "Propagation Mechanisms of Radio Waves Over Intra-Chip Channels with Integrated Antennas: Frequency-Domain Measurements and Time-Domain Analysis," *Transactions on Antennas and Propagation*, 55(10), pp.2900-2906.

[26] J. Branch, et al., (2005, April). "Wireless communication in a flip-chip package using integrated antennas on silicon substrates," *Electron Device Letters*, 26(2), pp.115-117.

[27] W. Bogaerts, M. Fiers, P. Dumon, "Design Challenges in Silicon Photonics," *IEEE Journal of Selected Topics in Quantum Electronics, vol*.20, no.4, 1-8, 2014.

[28] A. Karkar, T. Mak, K. F. Tong, and A. Yakovlev, "A Survey of Emerging Interconnects for On-Chip Efficient Multicast and Broadcast in Many-Cores". *IEEE Circuits and Systems Magazine*, vol. 16, no. 1, 58-72, 2016.

[29] A. Baroon. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, Vol. 39, no.3, 930–945, 1993.

[30] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, "A simulated annealing-based multi-objective optimization algorithm: AMOSA," *IEEE Transactions on Evolutionary Computation, vol. 12, no. 3*, 269–283, 2008.

[31] O. Lysne, T. Skeie, S.-A. Reinemo and I. Theiss, "Layered routing in irregular networks", *IEEE Trans. On Parallel Distributed Systems*, 2006, 17(1), 1 -65.

[32] K. Duraisamy, R. Kim, P. Pande, "Enhancing Performance of Wireless NoCs with Distributed MAC Protocols", *in Proc., of ISQED*, 2015, 406 – 411.

[33] J. Power, J. Hestness, M. Orr, M. Hill, and D. Wood, "gem5-gpu: A Heterogeneous CPU-GPU Simulator," *Computer Architecture Letters, vol. 13*, no. 1, 2014.

[34] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A Detailed On-chip Network Model inside a Full-system Simulator", *In Proceedings of International Symposium on Performance Analysis of Systems and Software*, Apr. 2009

[35] J. Leng et al., "GPUWattch: enabling energy optimizations in GPGPUs," *in International Symposium on Computer Architecture*, 487–498, 2013.