ALPHA: A Learning-Enabled High-Performance Network-on-Chip Router Design for Heterogeneous Manycore Architectures

Yuan Li, *Student Member, IEEE* and Ahmed Louri, *Fellow, IEEE*

Abstract—Heterogeneous manycores comprised of CPUs, GPUs and accelerators are putting stringent demands on network-on-chips (NoCs). The NoCs need to support the combined traffic, including both latency-sensitive CPU traffic and throughput-sensitive GPU and accelerator traffic. We study the characteristics of the combined traffic, and observe that (1) the limited injection bandwidth is the main obstacle to throughput improvement, and (2) the latency due to local and global contention accounts for a significant portion of the network latency. We propose a router architecture named ALPHA for heterogeneous manycores. ALPHA introduces two new optimizations: (1) increasing injection bandwidth to improve throughput, and (2) resolving local and global contention to reduce network latency. Specifically, ALPHA increases the injection bandwidth through modifications to injection link, crossbar switch and buffer organization in the injection port of the router; ALPHA identifies the upcoming local contention and resolves it by optimally selecting traffic routes; ALPHA detects and alleviates the global contention by utilizing a supervised learning engine for traffic analysis, prediction, and adjustment. Simulation results using Rodinia benchmark show that ALPHA provides 28 percent throughput increase, 24 percent latency reduction, 22 percent execution time speedup, and 19 percent energy efficiency improvement, compared to the baseline router.

Index Terms—Heterogeneous manycore, network-on-chip, router, supervised learning

1 Introduction & Motivation

Heterogeneous manycores have been shown to be able to achieve better energy efficiency and higher performance as compared to homogeneous manycores, due to the fact that they leverage on-chip specialization [1], [2], [3], [4], [5], [6], [7], [8]. Because of the differences in microarchitectures and programming models, different types of cores in a heterogeneous manycore system can have distinct traffic patterns and sensitivities to network throughput and latency. For example, GPU cores involve more point-to-point streaming traffic and are sensitive to throughput [9], [10], [11], [12], [13], [14], while CPU cores involve more coherence traffic and are sensitive to latency [15], [16], [17], [18], [19]. In order to simultaneously meet the communication demands of the different cores, high-throughput and low-latency network-on-chips (NoCs) are critical.

Conventional NoC and router designs for heterogeneous manycores focus on alleviating interference between the throughput-sensitive GPU traffic and the latency-sensitive CPU traffic [4], [20], [21], [22]. These designs either isolate different types of traffic through network partitioning [4], [20], or combine network partitioning with prioritization techniques which enable fast propagation of a certain type of traffic in the network [21], [22]. Although prior designs greatly alleviate the interference problem, they introduce some negative impacts on NoC performance. For example, network partitioning may result in poor network utilization and hence low throughput [20], while prioritization techniques introduce extra latency due to their complicated allocation schemes [23], [24], [25]. The negative impacts of the conventional designs on NoC performance motivate us to explore other throughput and latency optimization opportunities beyond traffic interference alleviation techniques.

Throughput Improvement. Previous works mainly employ two approaches to improve NoC throughput: (1) developing better crossbar switch allocation schemes [26], [27], [28], [29], and (2) increasing injection bandwidth [10], [11]. To explore the opportunities of improving NoC throughput, we run the Rodinia benchmark suite [30], [31] on a heterogeneous manycore system connected by an 8×8 2D mesh NoC with baseline two-stage wormhole routers, each of which includes 2 virtual channels (VCs), 4 flit buffers per VC, and an ideal crossbar switch allocation scheme. The ideal crossbar switch allocation scheme, which guarantees that an output port of a router is utilized as long as there is at least one request for it [32], represents the optimization limit of the first approach. We evaluate the average buffer utilization which indicates the NoC throughput as in Fig. 1, and average injection queue length as in Fig. 2.

We make two observations: (1) throughput improvement delivered by allocation scheme optimization is minimal in heterogeneous manycores, as the network is under-utilized (buffer utilization $\ll 1$) in Fig. 1, even with the ideal crossbar

• The authors are with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052 USA. E-mail: {liyuan5859, louri}@gwu.edu.

Manuscript received 4 June 2019; revised 28 Feb. 2020; accepted 5 Mar. 2020. Date of publication 17 Mar. 2020; date of current version 8 June 2021. (Corresponding author: Yuan Li.)
Recommended for acceptance by D. Atienza.
Digital Object Identifier no. 10.1109/TSUSC.2020.2981340

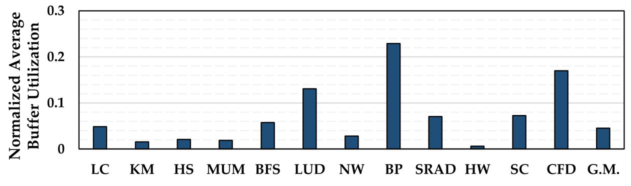


Fig. 1. Average buffer utilization when running the Rodinia benchmark suite, normalized to maximal buffer utilization. 1 in Y axis represents that the NoC buffers are fully utilized.

switch allocation scheme; (2) the limited injection bandwidth is the throughput bottleneck, as we infer low network throughput from Fig. 1 on one hand, and observe long injection queue in Fig. 2 on the other hand. These two observations motivate us to efficiently increase injection bandwidth while keeping the cost minimal, since previous works incur excessive overheads [10], [11].

Latency Reduction. The overall latency consists of two parts [15]: transfer latency and contention latency. The transfer latency refers to the latency incurred when traversing the links and the router pipelines, assuming there is no contention over shared network resources. A large number of techniques have been proposed to reduce the transfer latency. Some techniques reduce the number of hops with high-radix routers [33], [34] or bypass channels [35], while others reduce the latency of traversing router pipeline by overlapping some pipeline stages [36], [37], [38]. These techniques are sufficient when traffic load is relatively low and contention is less intensive. However, the traffic load is usually high in heterogeneous manycores because of the throughput-sensitive GPUs and accelerators. We run the Rodinia benchmark suite on a heterogeneous manycore system connected by an 8×8 2D mesh NoC with baseline two-stage wormhole routers, and observe that the contention latency takes up on average 55 percent of the network latency as shown in Fig. 3. Therefore, in order to further reduce the network latency, the transfer latency reduction techniques need to be augmented with efficient contention identification and resolving schemes.

Contention can be classified into two types [39]: local contention and global contention. An example of local contention is shown in Fig. 4a where two input ports send traffic to the same output port simultaneously. The local contention is expected to occur more frequently when the injection bandwidth increases, as the injection port alone can produce a contention as shown in Fig. 4b. An example of global contention is shown in Fig. 4c where contention occurs at a remote router. While identifying local contention is relatively easy, identifying global contention is challenging and often expensive [16], [39], [40], [41].

Proposed Router Architecture. To this end, we propose ALPHA, a NoC router design that is simultaneously

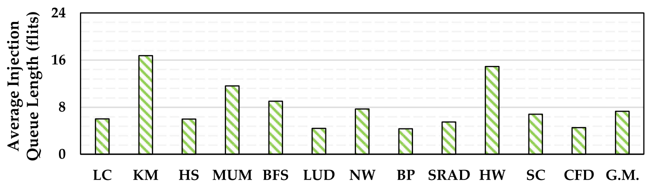


Fig. 2. Average injection queue length in network interface when running the Rodinia benchmark suite.

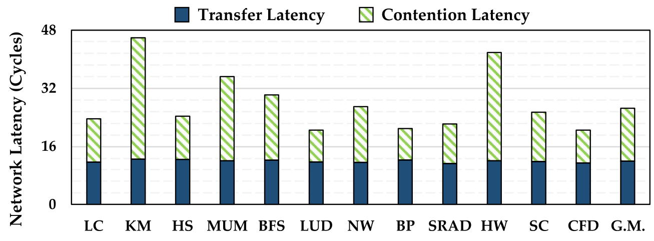


Fig. 3. Breakdown of overall network latency when running the Rodinia benchmark suite.

optimized for throughput and latency performance in heterogeneous manycores. ALPHA achieves high throughput by increasing the injection bandwidth, and low latency by resolving both local and global contention. The main contributions of this paper are:

- We extensively study the characteristics of the combined traffic in heterogeneous manycores, and observe that (1) the limited injection bandwidth is the major obstacle to throughput improvement, and (2) the latency due to local and global contention over the shared network resources takes up a significant portion of the network latency.
- Based on these observations, we propose ALPHA, a router architecture specifically designed for heterogeneous manycores. ALPHA introduces two new optimizations: (1) increasing injection bandwidth to improve throughput, and (2) resolving both local and global contention to reduce network latency. Specifically, ALPHA increases the injection bandwidth through architectural modifications to the injection link, the crossbar switch, and the buffer organization in the injection port of the router; ALPHA identifies the upcoming local contention at the crossbar switch and resolves it by optimally selecting traffic routes; ALPHA also utilizes a supervised learning engine to detect the global contention through traffic analysis and prediction, and alleviate the global contention by adaptively adjusting the traffic injection process.
- We evaluate the proposed router architecture using both synthetic traffic and the Rodinia benchmark suite. The simulation results using synthetic traffic show that ALPHA delivers 51 percent throughput increase and 38 percent latency reduction. The simulation results using the Rodinia benchmark suite show that ALPHA provides 28 percent throughput increase, 24 percent latency reduction, 22 percent execution time speedup, and 19 percent energy efficiency improvement, as compared to the baseline router architecture.

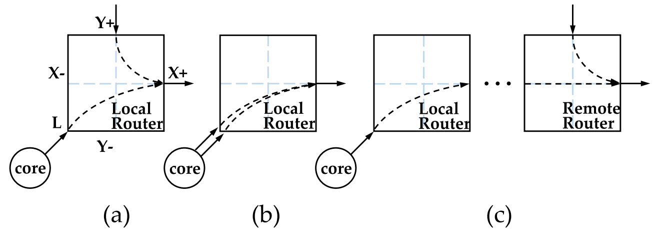


Fig. 4. Examples of (a) local contention (b) local contention due to increased injection bandwidth (c) global contention.

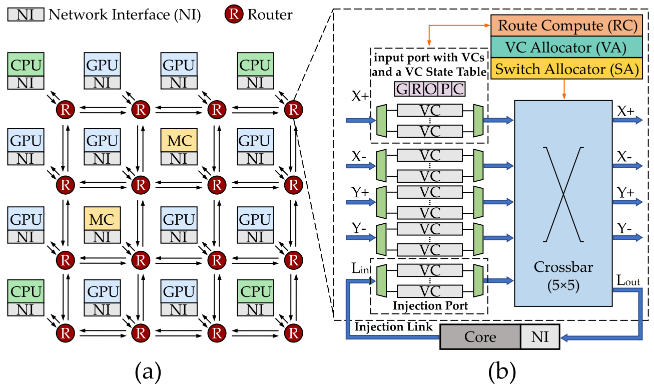


Fig. 5. (a) A typical heterogeneous manycore system connected by a NoC. (b) Baseline router architecture.

2 Baseline Architecture

Heterogeneous Manycore Architecture. A typical heterogeneous manycore architecture is shown in Fig. 5a. This architecture consists of latency-sensitive CPU cores, throughput-sensitive GPU cores, and memory controller (MC) nodes. MC nodes include memory controllers, shared L2 cache banks and directory controllers. All the cores and MC nodes are connected through network interfaces (NIs) by a NoC. The mixed placement of CPU and GPU cores in this heterogeneous manycore architecture is similar to [4], [21], [42], while placing MC nodes in network interior is similar to prior architectures [10], [42], [43].

Baseline Router. Two-stage wormhole routers, as in Fig. 5b, are implemented to connect all the cores and MC nodes in the heterogeneous manycore. As we employ wormhole routing, each traffic packet is either segmented into a head flit, several body flits and a tail flit, or packaged in a single head/tail flit, depending on the packet length. The route information within the head flit is read and processed by Route Compute (RC) logic. The Virtual Channel Allocator (VA) is responsible for allocating available virtual channels (VCs) of downstream routers to requests on a per-packet basis. The Switch Allocator (SA) assigns available paths in the crossbar switch to requests on a per-flit basis. Each input port holds a VC State Table, whose number of entries equals the number of VCs in that input port. Each entry in the VC State Table holds the information about global state (G), output port (R), output VC number (O), current head and tail pointer (P), and the number of credits (C).

3 ALPHA Router Architecture

The proposed ALPHA router introduces two new optimizations: (1) increasing the injection bandwidth to improve throughput, and (2) resolving both local and global contention to reduce network latency. In this section, we explain the architectural modifications and corresponding control scheme designs involved in these two optimizations in a step-by-step manner. First, we describe the necessary modifications to the injection link, the crossbar switch, and the buffer organization in the injection port of the router, in order to increase injection bandwidth. Second, we explain the optimizations involved in ALPHA router to resolve local contention, including modifications to VC State Table and RC/VA/SA units, and control scheme designs of route selection and switch allocation. Third, the architecture and functioning mechanism of the supervised learning engine designed for alleviating global contention are detailed in the last part of this section.

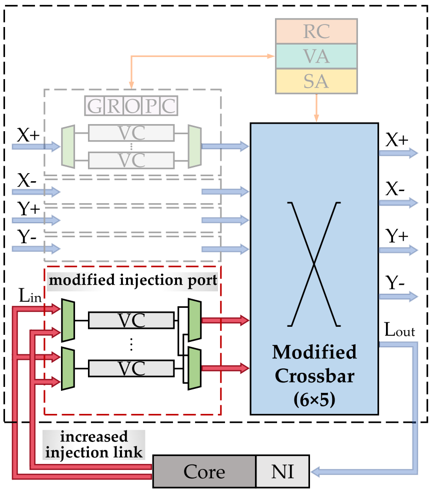


Fig. 6. Modifications in ALPHA to increase injection bandwidth, including increased injection link, modified crossbar switch, and modified buffer organization in the injection port. The blurry units remain the same as in baseline router.

3.1 Increasing Injection Bandwidth

ALPHA increases injection bandwidth by (1) increasing the injection link width, (2) increasing the crossbar switch dimension, and (3) modifying the buffer organization in the injection port of the router. These three modifications are highlighted in Fig. 6 and explained in detail below.

3.1.1 *Modification to Injection Link*

In order to increase the injection bandwidth, the injection link in ALPHA is increased to twice as wide as other links. We do not further increase the injection link width as other works [10], [11] do, because (1) the successive injected packets are very likely to share the same destination node due to data spatial locality, and (2) there are only a few output port choices for an injected packet if minimal routing is guaranteed. For example, there are at most 2 output port choices from a source node to a destination node in a 2D mesh network under minimal routing constraint. In this case, increasing the injection link width to over twice the link width in the baseline router results in trivial performance improvement while suffering significant overheads.

3.1.2 *Modification to Crossbar Switch*

In order to fully utilize the extended injection link, we also extend the number of crossbar inputs from P to $P+1$, where P is the number of input ports in the router. The additional crossbar input is connected to the injection port. Hence, up to two flits in the injection port can traverse the crossbar switch in one clock cycle. This design corresponds to the previous modification of increasing the injection link width. Specifically, the crossbar switch is extended from 5×5 to 6×5, as a normal router in 2D mesh topology has 5 input ports.

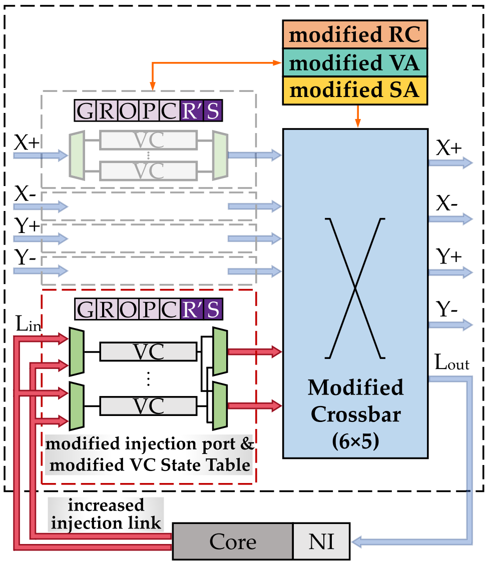


Fig. 7. Modifications in ALPHA to resolve local contention, on top of the router design of increasing injection bandwidth. New modifications include extended VC State Table, modified RC, modified VA, and modified SA.

3.1.3 Modification to Injection Port Buffer Organization

As the bandwidth at the injection link as well as at the crossbar switch has been increased, the injection port itself is now the bottleneck to increasing injection bandwidth. We modify the buffer organization and add additional MUXs to solve this issue. At the input side, the DEMUX is replaced by two MUXs. At the output side, an additional MUX is added and connected to the new input in the crossbar switch. By doing so, up to two flits can be written to and read from the buffers in the injection port in one clock cycle. The ALPHA router's capability of resolving local contention (Section 3.2) enables the injected packets to be timely transmitted without staying in buffers of the injection port for a long time, diminishing the benefits of implementing a large number of VCs in the injection port. In order to reduce overhead, we do not increase the number of VCs in the injection port.

3.2 Resolving Local Contention

The ALPHA router architecture is able to identify the upcoming local contention at the crossbar switch, and resolve it by optimally selecting traffic routes. This local contention resolving function is achieved in three steps. (1) Calculate all possible output ports corresponding to available routes of an incoming packet. We adopt O1TURN [44] multipath routing algorithm because it provides near-optimal worst-case throughput and does not incur long latency due to either increased packet hops or hardware complexity. (2) Collect the output port information of all the VCs which are competing for crossbar switch resources. (3) Make the routing decision for the incoming packet based on the information from the previous two steps, so that the chance of local contention is minimized.

To complete these three steps, we further modify the router architecture from Fig. 6 to include modifications to the VC State Table, the RC unit, the VA unit, and the SA unit as shown in Fig. 7. Additionally, new route selection and switch allocation control schemes are introduced.

3.2.1 Modification to VC State Table

As shown in Fig. 7, since we are employing multipath routing, the output port state field R in the VC State Table is extended to R and R' to accommodate multiple output ports. Besides, an additional state field S is required for route identification purpose. Specifically, in a 2D mesh topology, O1TURN multipath routing employs two minimal routes: the XY route and the YX route. The state fields R and R' store the output ports for the XY route and the YX route respectively. The state field S identifies whether XY route or YX route is being utilized. For a newly injection packet, the state fields R and R' are updated after RC pipeline stage, while the state field S is updated after the route selection process is finished in VA pipeline stage. For a packet passing by, the state field S determines whether the output port stored in R or the output port stored in R' is utilized.

3.2.2 Modification to Routing Computation

As opposed to the baseline router, ALPHA router calculates the output ports of multiple routes for each packet in the RC pipeline stage. Specifically, in a 2D mesh topology, ALPHA router calculates the output ports of both the XY and YX routes, and updates them to state fields R and R' in the VC State Table. The output ports of the XY and YX routes can be calculated simultaneously with symmetric logic. Hence, modification to RC does not incur extra latency in the RC pipeline stage.

Algorithm 1. Route Selection at the Injection Port

```
1  $v$: the number of virtual channels per input port
2  $d$: the array recording out port occupation mask
3  procedure RouteSelection( $v, d$ )
4    for $i \leftarrow 1$ to $v$ do
5      if $stage\ of\ ith\ VC = SA$
6        $d + out\ port\ of\ i\ th\ VC \leftarrow 1$
7    for $i \leftarrow 1$ to $v$ do
8      if $stage\ of\ ith\ VC = VA$
9        if $d + xy\ out\ port\ of\ ith\ VC = 1$
10         if $d + yx\ out\ port\ of\ ith\ VC = 0$
11           $out\ port\ of\ i\ th\ VC \leftarrow yx\ out\ port\ of\ i\ th\ VC$
12       else $out\ port\ of\ i\ th\ VC \leftarrow xy\ out\ port\ of\ i\ th\ VC$
13 end procedure
```

3.2.3 Modification to Virtual Channel Allocator

Architecture and Algorithm Design. In the ALPHA router, we add an XY/YX route selection unit to the VC allocator in the baseline router. This unit performs a two-step operation to select the right output port for each injected packet, with the target of avoiding local contention. First, it collects the output port information of all the VCs, which have been granted next hop VCs and are competing for crossbar switch resources. Second, it compares the collected information from the first step with output port candidates generated by the modified RC unit, and determines if switching from the primary XY route to the YX route can avoid the local contention at the crossbar switch. If so, the injected packet is routed through the YX route. Otherwise, the injected packet is still routed through the primary XY route. The route selection algorithm is described in detail in Algorithm 1.

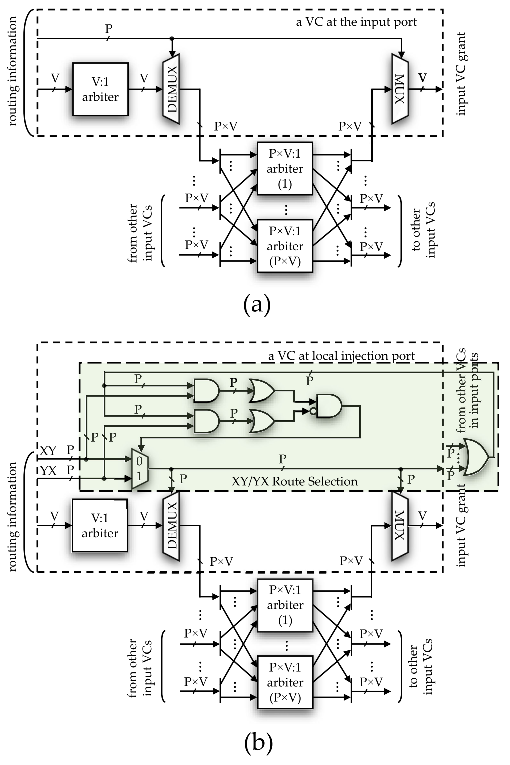


Fig. 8. Circuit-level implementation of VC Allocator in (a) baseline router, and (b) ALPHA router with the route selection unit.

Circuit-Level Implementation. The VC allocator performs matching between requests from the $P \times V$ input VCs and $P \times V$ resources, subject to the constraint that any VCs in the downstream routers requested by a given input VC at any given time share the same output port. P is the number of input ports in a router, while V is the number of VCs in each input port. Fig. 8a shows the circuit-level implementation of the input-first two-stage VC allocator in the baseline router. In the first stage, each input VC first determines which VC at the downstream router to bid on through the $V : 1$ arbiter. In the second stage, all the requests are forwarded through DEMUXs to $P \times V : 1$ arbiters to determine the requests that have won VCs in the downstream routers. Grants for each input VC are grouped and reduced to V-wide vector that indicates the granted VC.

Fig. 8b shows how the route selection unit fits into the original VC allocator in the baseline router. The route selection unit only applies to VCs in the injection port. The VCs in other input ports remain the same as in the baseline router in Fig. 8a. In the route selection unit, the output port information of each VC is first collected by a bitwise-OR gate, then compared with the output port information out of both the XY route and the YX route of current VC. Once a decision of whether utilizing the XY route or the YX route is made, it serves as the control signal of the MUX to forward the output port information of the utilized route to the control end of the DEMUX.

The critical path of the added route selection unit is from the VC allocation output of previous clock cycle to the control end of the DEMUX. Synthesis result from Synopsys Design Compiler shows that the propagation delay of the route selection unit is 3 percent longer than the delay of the 2:1 arbiter at the first stage. Hence, the propagation delay of the added route selection unit has been mostly overlapped by the original first-stage arbiter.

3.2.4 Modification to Switch Allocator

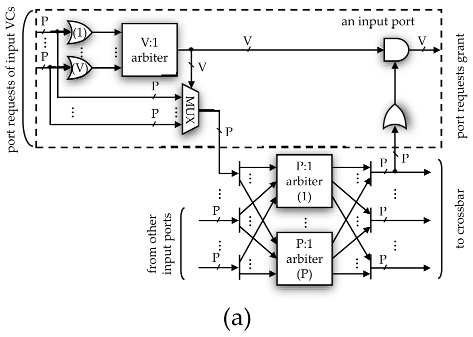
Architecture and Algorithm Design. As the dimension of the crossbar switch has increased, the number of requests at SA increases. To process the possible extra request from the injection port, the SA keeps searching for the second valid request after finding the first valid request. In this way, at most two requests from the injection port can be granted, enabling more than one VC in the injection port to access the crossbar switch in one clock cycle.

However, the arbitration could be unfair and lead to performance degradation when there is only one valid request from VCs in the injection port. In this case, the only request in the injection port will be treated as both the primary request and the secondary request, and sent for allocation twice, giving it a better chance to win the allocation than requests from other input ports. To avoid this unfairness, ALPHA introduces a Single Request Detection (SRD) unit, which monitors the number of valid requests in the injection port, and disables the function of searching for the second valid request when only one valid request exists. The switch allocation algorithm is described in Algorithm 2.

Algorithm 2. Select SA Requests at the Injection Port

```
v: the number of virtual channels per input port
round_robin: arbiter's round robin pointer
procedure SwitchReqSelection( v, round_robin)
  $r\_round\_robin \leftarrow round\_robin$
  for $i \leftarrow 1$ to $v$ do
    if stage of round_robin$^{th}$ VC = SA
      $one\_req \leftarrow outport of round\_robin^{th} VC$
      $round\_robin \leftarrow i$
      break
    $round\_robin \leftarrow (round\_robin + 1)\%v$
  $request\_cnt \leftarrow 0$
  for $i \leftarrow 1$ to $v$ do
    if stage of ith VC = SA
      $request\_cnt \leftarrow reqeust\_cnt + 1$
  if request_cnt $> 1$
    for $i \leftarrow 1$ to $v$ do
      $r\_round\_robin \leftarrow (r\_round\_robin - 1)\%v$
      if stage of r_round_robin$^{th}$ VC = SA
        $two\_req \leftarrow outport of r\_round\_robin^{th} VC$
        break
end procedure
```

Circuit-Level Implementation. The switch allocator performs matching between requests from the V input VCs at each of the router's P input ports, and available crossbar slots, subject to the constraint that at most one VC per input port can receive a grant. Fig. 9a shows the circuit-level implementation of the input-first two-stage switch allocator in the baseline router. In the first stage, the $V : 1$ arbiter determines a winner among all active VCs at each input port. In the second stage, the winning VCs' requests are then forwarded to the appropriate output ports, where $P : 1$ arbitrations take place among requests from different input ports. Grants for each input port are grouped and reduced to V-wide vector that indicates the winning input VC.



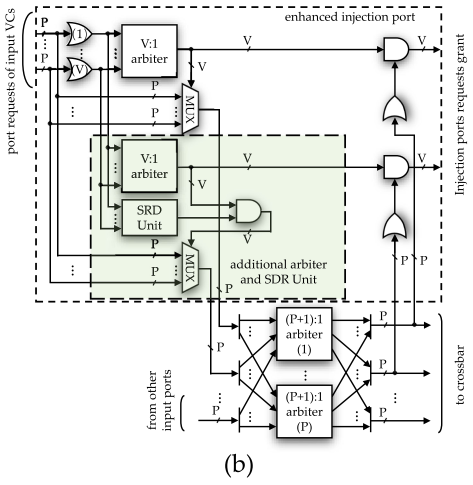
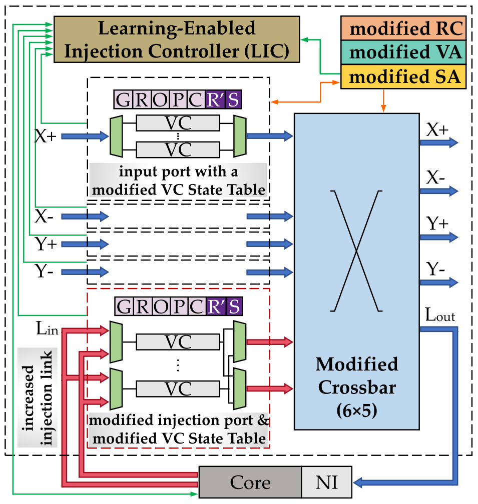


Fig. 9. Circuit-level implementation of switch allocator in (a) baseline router, and (b) ALPHA router with the additional $V:1$ arbiter, the SRD unit, and $(P+1):1$ arbiters.



Learning-Enabled Injection Controller (LIC)
modified RC
modified VA
modified SA
G R O P C R' S
VC
input port with a modified VC State Table
X+
X-
Y+
Y-
Lin
increased injection link
modified injection port & modified VC State Table
Modified Crossbar (6×5)
Lout
Core
NI

Fig. 10. The complete ALPHA router architecture with the Learning-Enabled Injection Controller (LIC) designed to resolve global contention.

Furthermore, the outputs of the second-stage arbiters directly drive the control signals to the crossbar switch.

Fig. 9b shows the three modifications to switch allocator in the ALPHA router architecture. (1) ALPHA router implements an additional $V:1$ arbiter in the first arbitration stage to process the possible extra request from the injection port. This additional arbiter shares the round robin pointer with the original arbiter, however, has opposite priority logic. Hence, if the original arbiter searches for the valid request in the clockwise order, the additional arbiter will search for the valid request in the counter-clockwise order. In this way, at most two valid requests from the injection port can be forwarded to the second arbitration stage. (2) An SRD unit is added to monitor the number of valid requests from VCs in the injection port. This unit outputs logic 0 and masks the outputs of the additional $V:1$ arbiter through a bitwise-AND gate, when there is only one valid request. (3) All the $P:1$ arbiters in the second arbitration stage are replaced by $(P+1):1$ arbiters due to increased crossbar switch dimension. The first two modifications only apply to the injection port, while the third modification applies to the entire SA.

Since the additional $V:1$ arbiter has symmetric circuit design as the original $V:1$ arbiter, it does not incur extra propagation delay. We synthesize the SRD unit and the $(P+1):1$ arbiter with Synopsys Design Compiler, assuming that P and V equal 5 and 2 respectively. Synthesis result shows that the propagation delay of the SRD unit is 53 percent shorter than that of the $V:1$ arbiter at the first stage, and the delay of the $(P+1):1$ arbiter is 7 percent longer than that of the original $P:1$ arbiter at the second stage. The extra delay of the $(P+1):1$ arbiter does not affect the maximal achievable operating frequency because switch allocator has shorter propagation delay than VC allocator [36].

3.2.5 *Deadlock-Free Design*

Employing O1TURN multipath routing is prone to deadlocks because all four turns are permitted, leading to potential cycles on the link acquisition graph [45]. We utilize escape VC [46] technique to resolve potential deadlocks. Other deadlock-free techniques such as VC partitioning [47] and bubble flow control [48] are also compatible with the ALPHA router architecture.

3.3 Learning-Enabled Traffic Injection Control to Alleviate Global Contention

The global contention problem in NoCs has been explored in many previous works. Some works perform adaptive routing to route traffic around the contention regions [37], [41], [49], [50], [51]. Adaptive routing is less effective when network load is high, as it does not throttle intensive traffic injection at sources but rather works with the traffic that has been injected into the network. Other works employ source throttling when contention is detected [16], [40], [52], [53]. The challenges of source throttling lie in (1) identifying global contention, and (2) adjusting the network load to achieve optimal system performance.

To meet these two challenges, we implement a Learning-Enabled Injection Controller (LIC) in each router as shown in Fig. 10. LIC acquires knowledge of system status by extracting information from core, router input ports, and SA unit. The collected information is then fed into a supervised learning engine to (1) identify the global contention through traffic analysis and prediction, (2) investigate the impact of network load on system performance, and (3) optimally adjust the traffic injection process. In the remaining part of this section, we first describe a new metric named SA Grant Rate, which is used to indicate global contention. We then explain the LIC architecture and the corresponding training and inference

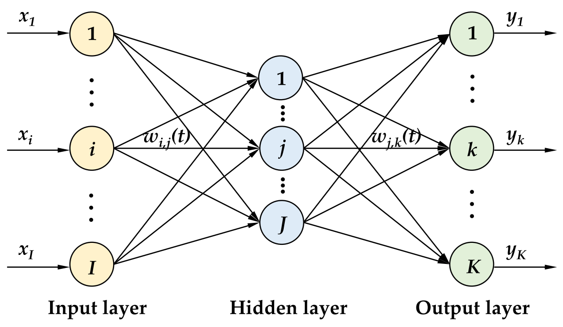


Fig. 11. The 3-layer feed-forward ANN architecture in LIC.

processes. Area overhead and compute latency are calculated in detail in the end.

3.3.1 Global Contention Indication

Previous works utilize multiple metrics to indicate contention, including available buffers [37], available VCs [49], output queue size [51], and starvation rate [16], [40]. We introduce a new metric named SA Grant Rate, which is the ratio of the number of granted SA requests ($N_{granted\_SA\_request}$) to the number of total SA requests ($N_{total\_SA\_request}$) as in Equation (1). We utilize this metric because the crossbar switch is the key shared network resource in the NoC. The SA Grant Rate below threshold indicates contention at the crossbar switch. The threshold is empirically set to 0.9. We analyze the impact of threshold value on system performance in Section 6.

$$SAGrantRate = \frac{N_{granted\_SA\_request}}{N_{total\_SA\_request}} \tag{1}$$

3.3.2 LIC Architecture

LIC utilizes a 3-layer feed-forward artificial neural network (ANN), as shown in Fig. 11, to realize supervised learning based injection process control. The three layers are the input layer, the hidden layer, and the output layer. I, J, and K represent the number of neurons in the input layer, the hidden layer, and the output layer respectively. $w_{i,j}$ represents the synapse weight between neuron i and j.

Input Layer. There are 10 neurons ($I = 10$) in the input layer because we feed 10 different features to the ANN. The features are normalized to be within the range of (0, 1).

Hidden Layer. We implement 8 neurons ($J = 8$) in the hidden layer based on the trade-off between ANN accuracy and implementation cost. Neurons in the hidden layer use the Sigmoid function as the activation function.

Output Layer. The output layer consists of 3 neurons ($K = 3$), each of which corresponds to a traffic injection mode. Neurons in the output layer use the Relu function as the activation function.

3.3.3 LIC Input Features

All the features that we have selected are listed in Table 1, and categorized into five categories: (1) prior mode, (2) global traffic, (3) local traffic, (4) CPU status, and (5) GPU status. By exploiting features from these five categories, LIC can adjust the traffic injection process with an awareness of core microarchitecture, application execution status, and network traffic load.

TABLE 1
Supervised Learning Features

#	Category	Feature
1	prior mode	maximal injection rate in previous epoch
2	global traffic	X+ incoming tagged packet rate in YX route X - incoming tagged packet rate in YX route Y+ incoming tagged packet rate in XY route Y - incoming tagged packet rate in XY route
3	local traffic	local injection packet count local SA Grant Rate
4	CPU status	L1D cache miss count L1I cache miss count instruction count
5	GPU status	L1 cache miss count active warp count coalesced data access count

Microarchitecture Awareness. We implement differentiated LIC designs in routers attached to different types of cores. Specifically, features from Category 1, 2, 3 and 4 are fed into LICs in routers attached to CPU cores, while features from Category 1, 2, 3 and 5 are fed into LICs in routers attached to GPU cores. We choose different features to represent the run-time status of CPU and GPU cores because of their distinct microarchitectures and programming models. Additionally, we do not implement LICs in the routers attached to MC nodes. This is because the MC nodes only contain passive modules (memory controllers, cache banks, etc.) and do not initiate communication.

Application Awareness. LIC takes run-time application execution status into consideration when adjusting the traffic injection process. Previous works [16], [40], [53] utilize features in Category 4 or their combinations to represent the run-time status of CPU cores. As shown in Category 5, in addition to L1 cache miss count, we introduce two new features, namely active warp count and coalesced data access count, to represent the run-time status of GPU cores. Previous work [4] has employed the active warp count to regulate GPU traffic. Furthermore, coalesced data access is a unique and important feature to characterize GPU execution [10], [54].

Traffic Awareness. The features in Category 2 and 3 represent the global and local traffic information respectively. In order to extract more global traffic information, we propose a light-weight piggy-back approach inspired by Internet-based contention detection schemes [55], [56]. This approach spreads contention information by selectively adding tags to traffic packets. A packet is tagged when it traverses a router whose SA Grant Rate is below a certain threshold. A router acquires more knowledge of the global network traffic by monitoring the ratio of tagged packets to total incoming packets.

Precise Contention Quadrant Location. Prior works [16], [40], [53] often apply source throttling without the knowledge of precise contention location. By combining multipath routing with the piggy-back approach, our design is able to

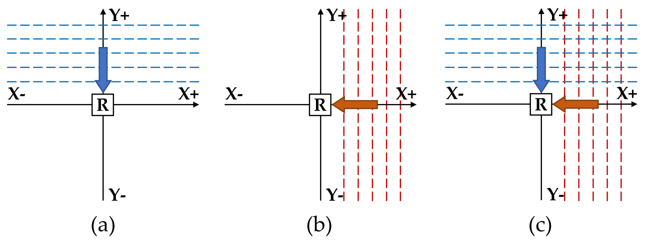


Fig. 12. Precise contention quadrant location enabled by the multipath routing and the piggy-back approach in our LIC design. R represents the router under examination.

determine the contention in each quadrant separately and only throttle traffic going to quadrants with intensive contention. Fig. 12 demonstrates an example of how our design determines the contention in the upper-right quadrant. The tagged packets in XY route in the $Y+$ input port come from the upper-half plane with blue dashed lines as in Fig. 12a. The tagged packets in YX route in the $X+$ input port come from the right-half plane with red dashed lines as in Fig. 12b. Contention in the upper-right quadrant then can be determined by monitoring both the incoming tagged packets rate in XY route in $Y+$ input port, and the incoming tagged packets rate in YX route in $X+$ input port, as shown in Fig. 12c.

3.3.4 LIC Output Modes

The three neurons in the output layer of the ANN in LIC correspond to three control modes the LIC can choose from: (1) turbo injection mode, (2) normal injection mode, and (3) throttled injection mode. Each of the three neurons outputs the probability of choosing the corresponding mode, and LIC always chooses the mode with the highest probability value.

Algorithm 3. Create Label for a Training Set

```
1  $\vec{M}$: mode in prior epoch
2  $n$: number of routers in the heterogeneous manycore
3  $GR[n]$: SA Grant Rate of each router in current epoch
4  $p\_GR[n]$: SA Grant Rate of each router in prior epoch
5  $C_{th}$: threshold value for contention identification
6  procedure CreateLabel( $\vec{M}, n, GR, p\_GR, C_{th}$)
7    for $i \leftarrow 1$ to $n$ do
8      if $GR[i] > C_{th}$
9        return $\vec{L} \leftarrow downgrade from \vec{M}$
10   for $i \leftarrow 1$ to $n$ do
11     $temp\_1 \leftarrow temp\_1 + GR[i]$
12     $temp\_2 \leftarrow temp\_2 + p\_GR[i]$
13   $avg\_GR \leftarrow temp\_1/n$
14   $avg\_p\_GR \leftarrow temp\_2/n$
15   if $avg\_GR > avg\_p\_GR$
16     return $\vec{L} \leftarrow \vec{M}$
17   return $\vec{L} \leftarrow upgrade from \vec{M}$
18 end procedure
```

Turbo Injection Mode. This mode takes full advantage of the increased injection bandwidth. In this mode, each router transmits up to two injected flits per clock cycle.

Normal Injection Mode. In this mode, a router processes the injected traffic in the same manner as the baseline router. Only one injected flit can be transmitted per clock cycle.

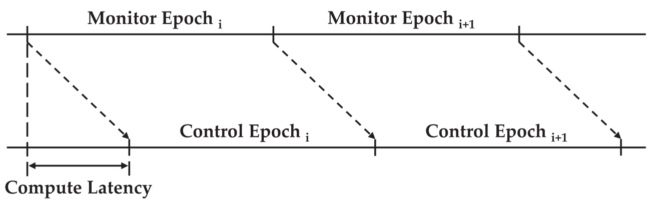


Fig. 13. The monitor epochs and control epochs are offset by ANN compute latency.

Throttled Injection Mode. We apply an 85 percent throttling rate in this mode, which means a node is allowed to inject traffic only in 15 percent of total clock cycles. We choose this throttling rate in order to achieve optimal system performance. The analysis of the impact of throttling rate on system performance is presented in Section 6.

3.3.5 LIC Training Process

LIC performs the training process off-line with an epoch step of $10K$ clock cycles. We randomly choose $200M$ execution clock cycles out of each application in the Rodinia benchmark suite. Since we select 12 applications from this benchmark suite, there are in total $240K$ training samples. Each training sample $(\vec{X}, \vec{L})$ includes an input feature vector $\vec{X}$ and a label vector $\vec{L}$. The algorithm to create a label in a training set is described in Algorithm 3. There are three possible labels (1, 0, 0), (0, 1, 0) and (0, 0, 1), representing the turbo injection mode, the normal injection mode, and the throttled injection mode respectively. The synapse weights are updated at each epoch step using back propagation algorithm [57]. The learning gain factor η is set to be 0.01 because accuracy is more important than the time cost in the off-line training process. After the training process, the synapse weights are stored within each router for inference purpose.

3.3.6 LIC Inference Process

We set the epoch step to $10K$, which is smaller than in other works [16], [40], [53], because (1) a smaller epoch step helps to capture transient traffic patterns and apply more precise traffic injection control, and (2) switching from one injection mode to another does not incur any penalties. Since the ANN compute latency accounts for a significant portion of an epoch step, we utilize two separate sets of epochs for monitoring features and applying traffic injection control. LIC collects feature information at the end of each monitor epoch, and applies a new injection mode at the beginning of each control epoch. The offset between these two sets of epochs, as shown in Fig. 13, may lead to performance degradation because the control decisions are made on outdated feature information. A small epoch step leads to more significant offset between two sets of epochs, and hence more possible performance degradation. We set the epoch step to $10K$ clock cycles. We analyze the impact of epoch step size on system performance in Section 6.

3.3.7 Implementation Cost & Compute Latency

Implementation Cost. The implementation cost of LIC consists of three parts: integer counters for feature monitoring purpose, SRAM cells to store synapse weights, and an arithmetic logic unit (ALU). We implement 14 16-bit integer

TABLE 2
Baseline Architecture Configurations

GPU core	40 shader cores, 0.7 GHz, SIMD pipeline width = 8 48 warps/core, 32 threads/warp, 8 CTAs/core 48kB scratch memory/core, 32k registers/core
GPU cache	L1 64kB/core, 4-way, 64B line size, LRU
CPU core	16 X86 cores, 2GHz
CPU cache	L1 D-cache 64kB/core, 4-way, 64B line size, LRU L1 I-cache 32kB/core, 4-way, 64B line size, LRU
Shared LLC	L2 cache 1MB/MC, 8-way, 64B line size, LRU
DRAM	DDR3 timing, 8 channels
NoC	8x8 2D mesh topology, dimension-order routing wormhole flow control, 2-stage pipeline router channel width = 128 bits, 2 VCs/virtual network 4 flit buffers/data VC, 1 flit buffer/control VC

counters to monitor features. There are 104 synapse weights in the proposed ANN architecture, each of which is stored in a 16-bit SRAM cell in IEEE 754 half-precision binary floating-point format. The ALU is designed to only perform multiplication, addition and division operations as well as activation functions.

Compute Latency. The ANN compute latency is calculated based on latency data of different arithmetic operations provided by [58]. The floating-point multiplication, addition and division operations consume 5, 3 and 44 clock cycles respectively. The total ANN compute latency is less than 1.5K clock cycles.

4 Methodology

We implement a heterogeneous manycore system with ALPHA routers in the Gem5-GPU simulator [59]. DSENT2.0 simulator [60] is used to calculate the power consumption based on the data extracted from Gem5-GPU. We use Synopsys Design Compiler and a 45 nm open-source library to evaluate the area overhead and calibrate the static power consumption result acquired from DSENT2.0. Table 2 shows the configuration of our baseline heterogeneous manycore system.

Applications. In addition to utilizing multiple synthetic traffic patterns, we evaluate the ALPHA router architecture with 12 applications selected from the Rodinia benchmark suite in Gem5-GPU simulator in full-system mode. The Rodinia benchmark suite is specifically designed for heterogeneous systems, exhibiting various types of parallelism, data-access patterns, and data-sharing characteristics.

Different Configurations for Evaluation. We compare the ALPHA router design with five other router architectures or NoC configurations. Baseline configuration, which leverages multiple latency reduction techniques [36], [37], [38], includes regular 2-stage wormhole routers. VIX [32] configuration includes routers with virtual input crossbar design, which attempts to improve switch allocation. DIP [10] configuration includes routers with dual injection ports, which is a straightforward approach to increase the injection bandwidth. O1TURN [44] configuration evenly distributes injected traffic to XY route and YX route. OSCAR [21] specifically targets heterogeneous manycores. It dynamically allocates network bandwidth to CPU and GPU traffic through asynchronous batch scheduling. In our evaluation, we use the same setup as in the original design to adjust the batch composition ratio every 200K cycles.

5 Experiment Results

We present and analyze the results acquired from both synthetic traffic simulation and full-system simulation using the Rodinia benchmark suite in this section. We also analyze the area overhead of ALPHA router design as compared to other five configurations.

5.1 Results From Synthetic Traffic Simulation

We compare the ALPHA router design with four other configurations under different network load conditions using multiple synthetic traffic patterns. The LIC in ALPHA router is disabled to keep the comparison fair, as it may affect the injection rate. OSCAR is not included in this simulation as it is specifically designed for systems that simultaneously involve multiple traffic patterns. The bit-reverse and transpose traffic patterns are used to simulate the intense point-to-point streaming traffic generated by GPU cores, while the uniform-random traffic pattern is used to simulate the relatively uniform-distributed traffic generated by CPU cores. For each synthetic traffic pattern, we examine both network latency and throughput, as shown in Fig. 14.

We make four observations when examining the network latency results from three different synthetic traffic patterns. (1) ALPHA router design does not reduce network latency when the network load is low, as compared to other configurations. (2) ALPHA router reduces network latency by 38 and 34 percent in bit-reverse traffic and transpose traffic respectively, as traffic becomes more intensive and network load reaches a similar level as in real heterogeneous applications. The significant reduction in network latency mainly results from the contention resolving function of the ALPHA router design. (3) VIX delivers low network latency when network load is high in uniform-random traffic because it improves crossbar switch allocation. (4) DIP suffers from high network latency in all three traffic patterns. This is because it fails to timely transmit the injected traffic out of the source router.

ALPHA router architecture achieves higher throughput in all three traffic patterns compared to other configurations. The throughput increases by 51 percent in bit-reverse traffic, 47 percent in transpose traffic, and 10 percent in uniform-random traffic. In bit-reverse and transpose traffic patterns, O1TURN provides higher throughput than Baseline, VIX, and DIP, because it exploits route diversity and hence postpones network saturation.

5.2 Results From Full-System Simulation

We evaluate the ALPHA router design in full-system mode with 12 applications selected from the Rodinia benchmark suite. We compare the ALPHA router design with all five other configurations by examining network latency, throughput, execution time speedup, and energy efficiency.

Network Latency. We make four observations from the network latency comparison between ALPHA router design and four other configurations, which is shown in Fig. 15. (1)

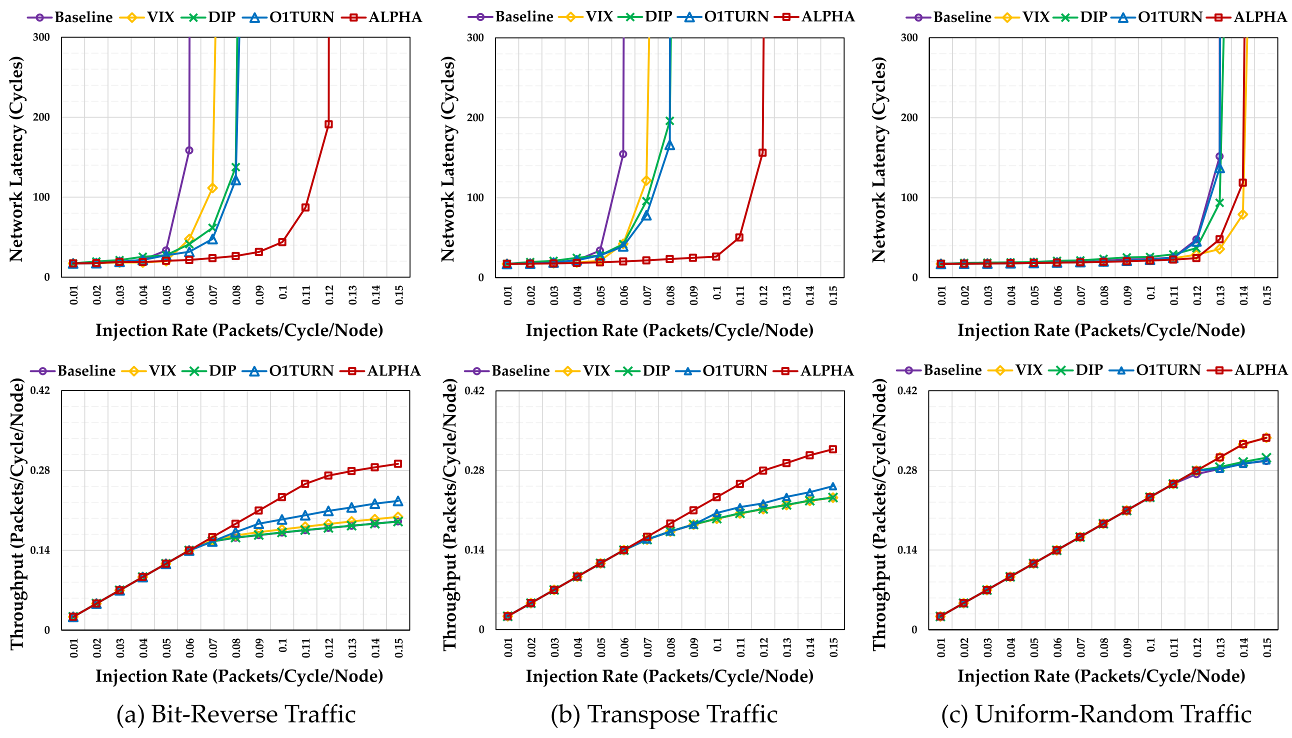


Fig. 14. Network latency and throughput results from synthetic traffic (a)bit-reverse (b)transpose, and (c) uniform-random.

ALPHA router design reduces network latency by 24, 18, 20, 21, and 24 percent, as compared to Baseline, VIX, DIP, O1TURN, and OSCAR respectively. This is because the ALPHA router effectively resolves both local and global contention. (2) ALPHA performs extremely well in applications with high network load and hence high probability to incur contention. These applications include SC, NW, MUM, and BFS which suffer from high cache miss rate, as well as BP and SRAD which involve a large volume of CPU-GPU communication between the serial CPU phases and the parallel GPU kernels. (3) O1TURN delivers the least latency reduction, as evenly distributing traffic to multiple routes does not necessarily help resolve contention and hence reduce latency. (4) The reserved VC for CPU traffic in OSCAR, though mitigates starvation of CPU traffic, leads to long average latency.

Throughput. Throughput is defined as the average number of flits ejected from the network in a unit time period, which is shown in

$$Throughput = \left(\sum_{i=1}^{n} N_i\right) \times T_{exec}{}^{-1}, \tag{2}$$

n is the number of routers in the network, N_i is the number of flits ejected from router i, and T_{exec} is the application execution time. Fig. 16 shows the normalized throughput results for all configurations. ALPHA router design achieves 28, 17, 18 percent, higher throughput than Baseline, VIX, and O1TURN configurations respectively. DIP outperforms VIX and O1TURN, which conforms to our prior observation that the limited injection bandwidth is the major obstacle to throughput improvement. The performance difference between DIP and ALPHA results from the fact that ALPHA utilizes the increased injection bandwidth more effectively by resolving local contention.

Execution Time Speedup. The execution time speedup is obtained by calculating the ratio of the application execution time of the Baseline configuration to the application execution time of other configurations (VIX, DIP, O1TURN, OSCAR, and ALPHA). The execution time speedup of the Baseline configuration itself is always 1. As shown in Fig. 17, the ALPHA router design achieves up to 22 percent execution time speedup as compared to other configurations. In applications including intensive CPU execution phases such as KM and NW, the speedup mainly comes

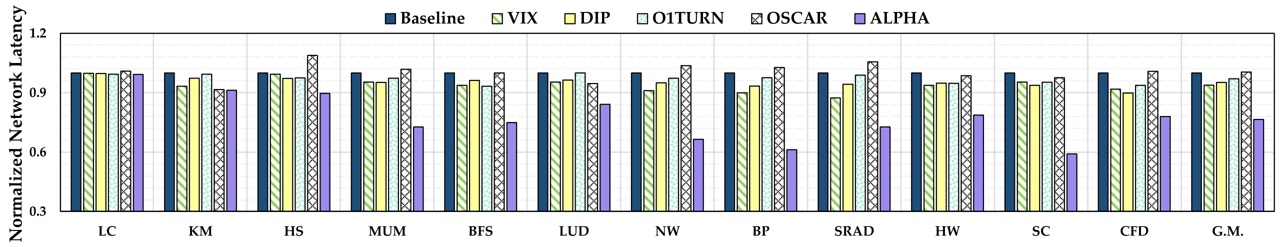


Fig. 15. Normalized network latency comparison between the ALPHA router design and other configurations. All network latency values are normalized to network latency in the Baseline configuration.

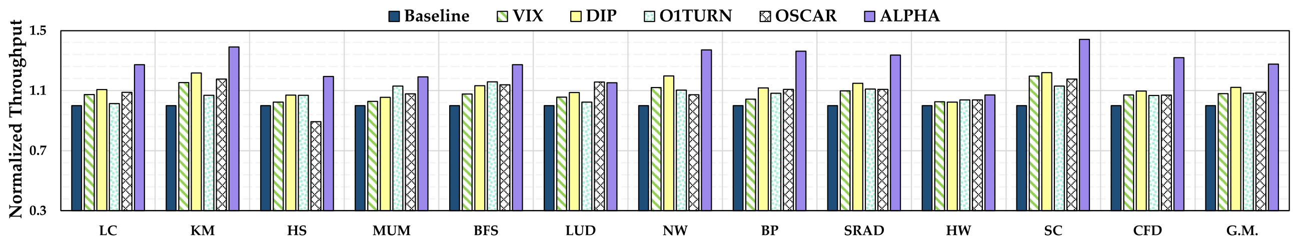


Fig. 16. Normalized throughput comparison between the ALPHA router design and other configurations. All throughput values are normalized to throughput in the Baseline configuration.

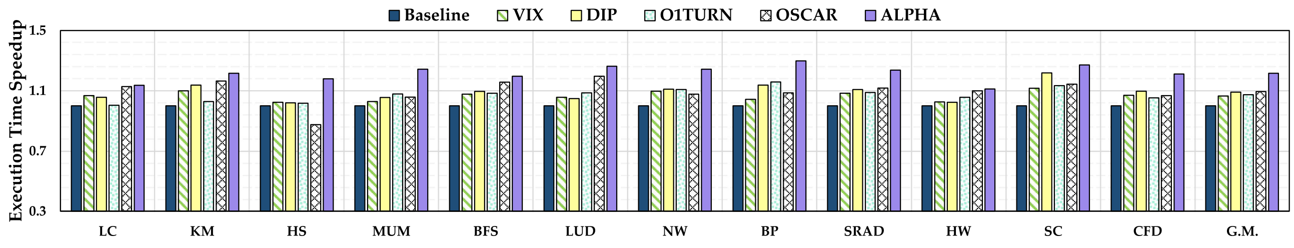


Fig. 17. Execution time speedup of VIC, DIP, O1TURN, and ALPHA configurations over the Baseline configuration. Speedup of the Baseline configuration itself is always 1.

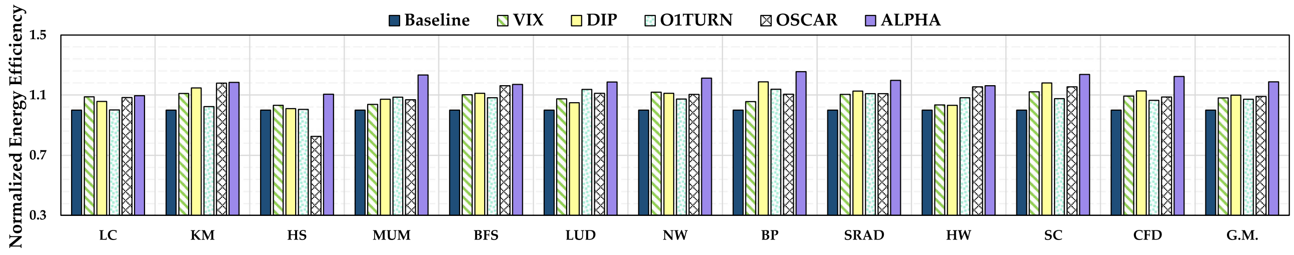


Fig. 18. Normalized energy efficiency comparison between the ALPHA router design and other configurations. All energy efficiency values are normalized to energy efficiency in the Baseline configuration.

from network latency reduction. In some other applications that offload execution to GPU cores such as HS and SC, the speedup mainly comes from throughput improvement. There are also applications which involve both serial CPU phases and parallel GPU kernels (BP, SRAD, etc.). Both latency reduction and throughput improvement contribute to execution time speedup in these applications in the ALPHA router design. OSCAR achieves execution time speedup by allowing critical packets such as CPU packets to be timely transmitted to their destinations.

Energy Efficiency. We define energy efficiency as the reciprocal of energy consumption of running an application, which is shown in

$$Energy\ Efficiency = [(P_{static} + P_{dynamic}) \times T_{exec}]^{-1}. \tag{3}$$

P_{static} and $P_{dynamic}$ are static and dynamic power consumption respectively, and T_{exec} is the application execution time. Fig. 18 shows the energy efficiency measurements for all configurations studied and normalized to the Baseline configuration. The ALPHA router design improves energy efficiency by 19 percent, compared to the Baseline configuration, while the maximal energy efficiency improvement of other configurations is 10 percent (DIP). The energy efficiency improvement of the ALPHA design results from both reducing execution time and alleviating local and global contention.

5.3 Area Estimation

We estimate the router area in different configurations using Synopsys Design Compiler and a 45 nm open-source library. Table 3 lists the area of each router module and the overall router area for different configurations. There is modest mismatch between the sum of the area of all router modules and the overall router area, depending on the employed place and route strategy. ALPHA router design incurs 5.4 percent area overhead compared to the Baseline configuration, due to modifications to the RC unit, the VA unit, the SA unit, and the crossbar switch, as well as the additional LIC. However, the area of ALPHA router is still 5 percent and 10 percent smaller compared to VIX and DIP configurations respectively. This is because we keep our designs of increasing injection bandwidth and resolving local contention light-weight. O1TURN configuration incurs the least area overhead as it only involves limited modifications to the RC unit and the VA unit.

6 Design Analysis

In this section, we examine the performance of the ALPHA router in detail, and explain several critical design choices.

Contention Resolving Function. Both DIP and ALPHA increase the injection bandwidth as compared to the Baseline configuration. We examine the network contention in DIP and ALPHA by analyzing the SA Grant Rate shown in

TABLE 3
Area Estimation

Configuration	Area [mm^2]									
	Flit Buffer/Logic	RC Unit	VA Unit	SA Unit	Crossbar	VC Table	LIC	Batch Scheduling	Total	Increase by %
Baseline	0.1860	0.0004	0.0091	0.0046	0.0178	0.0012	-	-	0.2303	-
VIX	0.1860	0.0004	0.0091	0.0092	0.0356	0.0012	-	-	0.2556	11.0%
DIP	0.2232	0.0004	0.0091	0.0055	0.0178	0.0012	-	-	0.2710	17.7%
O1TURN	0.1860	0.0009	0.0093	0.0046	0.0178	0.0012	-	-	0.2321	0.78%
OSCAR	0.1860	0.0004	0.0091	0.0058	0.0178	0.0012	-	0.0064	0.2367	2.78%
ALPHA	0.1860	0.0009	0.0094	0.0055	0.0214	0.0012	0.0173	-	0.2427	5.38%

Fig. 19, and make two observations. (1) Increasing the injection bandwidth leads to intensive network contention, as the geometric mean of SA Grant Rate in DIP drops to around 87 percent. The SA Grant Rate is relatively high in applications with low network load such as LC and KM, while it is low in applications with high network load such as BP and HW. (2) The contention resolving design in ALPHA is effective, because the SA Grant Rate in ALPHA is 7 percent higher than in DIP. ALPHA performs well in applications with high network load. For example, ALPHA achieves 15 and 11 percent higher SA Grant Rate than DIP in BP and HW applications respectively.

Injection Mode Breakdown. Fig. 20 shows the breakdown of injection modes in ALPHA design in different applications. We capture the ratio of the number of clock cycles utilized by each injection mode to the total application execution clock cycles. The turbo injection mode is frequently used in most applications, indicating the necessity of increasing injection bandwidth. The throttled injection mode is barely used in applications such as LC and LUD. These applications either involve a high fraction of CPU execution or have low cache miss rate. Some other applications (HS, MUM, and BFS, etc.) frequently use the throttled injection mode to reduce network load and hence avoid intensive network contention.

SA Grant Rate Threshold. A router is considered congested when its SA Grant Rate is below a threshold C_{th} in the ALPHA design. We evaluate the impact of this threshold C_{th} on system performance, and observe that the optimal system performance is achieved when the threshold C_{th} is set to 0.9, as shown in Fig. 21. A small C_{th} may not be sensitive enough to identify contention, while a large C_{th} limits the benefit of utilizing the increased injection bandwidth.

Throttling Rate in Throttled Injection Mode. We also examine the impact of throttling rate in the throttled injection mode on system performance. The optimal system performance is obtained when the throttling rate is set to 0.85 as shown in Fig. 22. This throttling rate effectively reduces the network load and avoids contention while incurring minimal negative impact on system performance in the throttled injection mode.

LIC Epoch Size & Hidden Layer Neuron Count. We evaluate the impact of the LIC epoch step size on system performance and present the results in Fig. 23a (normalized to the $10K$ epoch step size case). The optimal system performance is obtained when the LIC epoch step size is $10K$. Implementing more neurons in LIC hidden layer leads to system

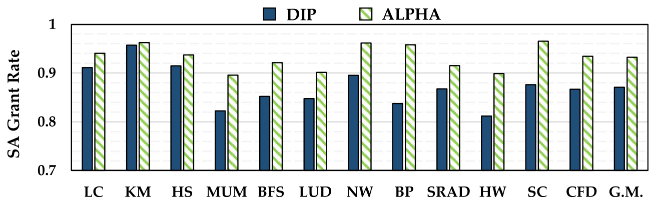


Fig. 19. The SA Grant Rate comparison in DIP and ALPHA.

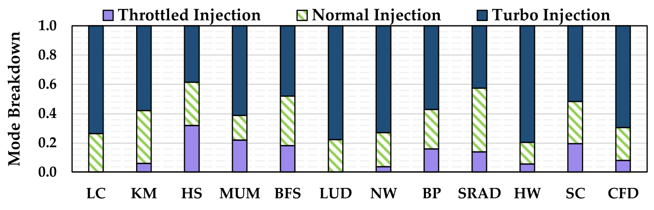


Fig. 20. The injection mode breakdown in ALPHA design.

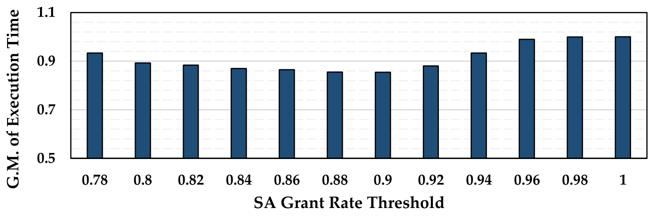


Fig. 21. Geometric mean of execution time with different SA Grant Rate threshold C_{th}, normalized to $C_{th} = 1$.

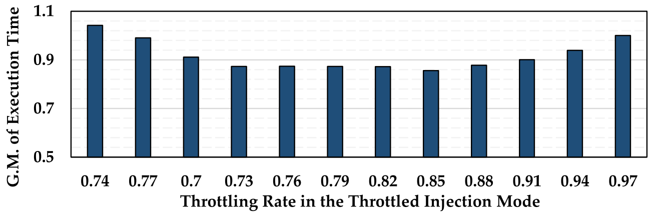


Fig. 22. Geometric mean of execution time with different throttling rate, normalized to throttling rate of 0.97.

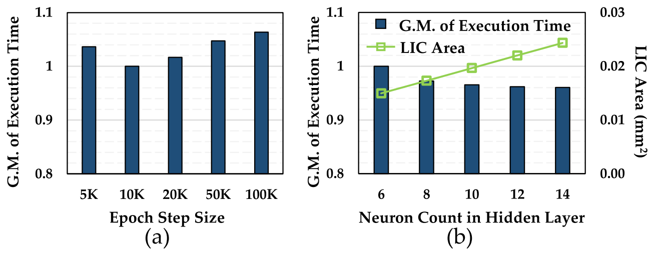


Fig. 23. The impact of (a) LIC epoch step size and (b) hidden layer neuron count on system performance.

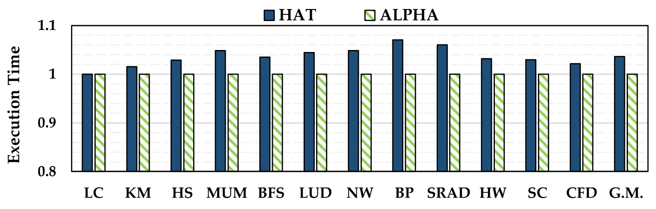


Fig. 24. Execution time comparison of HAT and ALPHA.

performance improvement at the cost of large LIC area. From Fig. 23b (normalized to the 6 neurons case), we observe that implementing 8 neurons has a good balance between system performance and area overhead.

LIC Effectiveness. We compare the proposed LIC with another global contention identification mechanism named HAT [53]. To keep the comparison fair, HAT is implemented on top of the design described in Section 3.2 and Fig. 7. We extend the range of throttling rate to adapt to the increased injection bandwidth while maintaining the adjustment epoch size of $100K$ cycles as in [53]. Fig. 24 shows that ALPHA achieves 3.6 percent execution time reduction as compared to HAT. ALPHA outperforms HAT for several reasons: (1) ALPHA adjusts injection rate at a finer time interval because it includes neither centralized control nor complicated computations; (2) ALPHA's injection rate control is on per-core basis while HAT applies a universal injection rate to all throttled cores; and (3) ALPHA can identify the contention in each quadrant separately and only throttle traffic going to quadrants with intensive contention.

7 Related Work

Increase Throughput. Prior works mainly focus on optimizing the crossbar switch allocation scheme to improve throughput [26], [28], [61]. These works usually involve complicated allocator design such as the iterative allocator [27] or wavefront allocator [29]. Rao *et al.* [32] propose to improve the crossbar switch allocation by feeding more SA requests to the allocator. Other works improve throughput by enhancing the injection process [10], [11]. Xue *et al.* [62] improve throughput by exploiting user-cooperated network coding. We observe that the limited injection bandwidth is the major obstacle to throughput improvement, and propose a light-weight design to tackle this obstacle.

Reduce Network Latency. Conventional works have concentrated on reducing the transfer latency by reducing the number of hops [33], [34], [35], [63] or overlapping router pipeline stages [36], [37], [38], [64]. Monemi *et al.* [65] apply some latency reduction techniques to their NoC prototype development. Prior latency reduction techniques are effective when network load is relatively low. We observe that the network load is usually high in heterogeneous manycores and the contention latency takes up a significant portion of the overall network latency. We propose multiple contention resolving techniques to reduce the contention latency, hence reduce the overall network latency.

Multipath Routing. Multipath routing has been widely explored in the networking community [44], [66]. Tang *et al.* [67] propose the repetitive turn model, which combines multiple routing algorithms with low routing pressures, to improve network performance. Murali *et al.* [68] propose a multipath routing technique with the in-order guarantee by implementing a lookup table at the switch of reconvergent nodes. Yang *et al.* [69] propose a simultaneous dual-path routing algorithm, which is beneficial when transmitting large-size packets. The multipath routing scheme we have employed in this work differs from previous schemes because (1) we select a route based on the local contention information instead of evenly or randomly distributing traffic to all available routes, and (2) the employed multipath routing scheme yields maximal benefit only when combined with the increased injection bandwidth.

Learning-Enabled NoC Design. Multiple machine learning techniques have been introduced to perform design tradeoff or predict traffic in NoCs. These works target to achieve high power efficiency [70], [71], [72], [73], [74], [75], [76], and enable fault-tolerant design [74], [77], [78]. Xiao *et al.* [79] exploit neural network and reinforcement learning to optimize task mapping in a heterogeneous system. Qian *et al.* [80] use kernel-based support vector regression method to predict the NoC performance. We apply the supervised learning technique to detect the global contention through traffic analysis and prediction, and adaptively select injection mode to optimize system performance.

8 Conclusion

The NoCs for heterogeneous manycores are expected to be optimized for the combined traffic from latency-sensitive CPUs and throughput-sensitive GPUs and accelerators. We extensively analyze the characteristics of the combined traffic, and observe that (1) the limited injection bandwidth is the major obstacle to throughput improvement, and (2) the latency due to local and global contention over the shared network resources takes up a significant portion of the overall network latency. Based on our observations, we propose ALPHA, a NoC router design that is simultaneously optimized for throughput and latency performance in heterogeneous manycores. ALPHA increases the injection bandwidth through architectural modifications to the injection link, the crossbar switch and the buffer organization in the injection port of the router. ALPHA resolves the local contention by combining a contention detection design with multipath routing. Furthermore, ALPHA implements a supervised learning engine to detect the global contention through traffic analysis and prediction, and alleviate it by adaptively adjusting the traffic injection process. Our simulation results show that the ALPHA router outperforms the baseline router and other state-of-art router designs in terms of latency, throughput, execution time speed and energy efficiency. We conclude that ALPHA is an effective router design for heterogeneous manycores.

Acknowledgments

This research was partially supported by National Science Foundation Grants CCF-1547035, CCF-1547036, and CCF-1702980. We sincerely thank the anonymous reviewers for their excellent feedback.

References

[1] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?" in *Proc. 43rd IEEE/ACM Int. Symp. Microarchitecture*, 2010, pp. 225–236.

[2] M. Arora, S. Nath, S. Mazumdar, S. B. Baden, and D. M. Tullsen, "Redefining the role of the cpu in the era of CPU-GPU integration," *IEEE Micro*, vol. 32, no. 6, pp. 4–16, Nov. 2012.

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

[4] O. Kayiran *et al.*, "Managing GPU concurrency in heterogeneous architectures," in *Proc. 47th IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 114–126.

[5] G. Venkatesh *et al.*, "Conservation cores: Reducing the energy of mature computations," in *Proc. 15th Architect. Support Program. Languages Operating Syst.*, 2010, pp. 205–218.

[6] S. Yehia, S. Girbal, H. Berry, and O. Temam, "Reconciling specialization and flexibility through compound circuits," in *Proc. 15th IEEE Int. Symp. High-Perform. Comput. Architecture*, 2009, pp. 277–288.

[7] J. Power *et al.*, "Heterogeneous system coherence for integrated CPU-GPU systems," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2013, pp. 457–467.

[8] H. Zheng and A. Louri, "A versatile and flexible chiplet-based system design for heterogeneous manycore architectures," in *Proc. 57th ACM/IEEE Des. Autom. Conf.*, pp. 1–6, 2020.

[9] J. Cong, M. Gill, Y. Hao, G. Reinman, and B. Yuan, "On-chip interconnection network for accelerator-rich architectures," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2015, pp. 8:1–8:6.

[10] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," in *Proc. 43rd IEEE/ACM Int. Symp. Microarchitecture*, 2010, pp. 421–432.

[11] X. Cheng, Y. Zhao, H. Zhao, and Y. Xie, "Packet pump: Overcoming network bottleneck in on-chip interconnects for GPGPUs," in *Proc. 55th ACM/ESDA/IEEE Des. Autom. Conf.*, 2018, pp. 84:1–84:6.

[12] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-efficient on-chip interconnect designs for GPGPUs," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2015, pp. 9:1–9:6.

[13] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing cost-effective on-chip network bandwidth in GPGPUs," in *Proc. 30th IEEE Int. Conf. Comput. Des.*, 2012, pp. 407–412.

[14] K. H. Kim, R. Boyapati, J. Huang, Y. Jin, K. H. Yum, and E. J. Kim, "Packet coalescing exploiting data redundancy in GPGPU architectures," in *Proc. 31st ACM Int. Conf. Supercomputing*, 2017, pp. 6:1–6:10.

[15] L. S. Peh and N. E. Jerger, *On-Chip Networks*. San Rafael, CA, USA: Morgan & Claypool, 2009.

[16] G. Nychis, C. Fallin, T. Moscibroda, and O. Mutlu, "Next generation on-chip networks: What kind of congestion control do we need?" in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, pp. 12:1–12:6.

[17] Y. Yao and Z. Lu, "Opportunistic competition overhead reduction for expediting critical section in NoC based CMPs," in *Proc. 43rd ACM/IEEE Int. Symp. Comput. Architecture*, 2016, pp. 279–290.

[18] H. Zheng and A. Louri, "EZ-pass: An energy performance-efficient power-gating router architecture for scalable NoCs," *IEEE Comput. Architecture Lett.*, vol. 17, no. 1, pp. 88–91, Jan. 2018.

[19] Y. Chen and A. Louri, "An approximate communication framework for network-on-chips," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1434–1446, Jun. 2020.

[20] 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.

[21] J. Zhan, O. Kayiran, G. H. Loh, C. R. Das, and Y. Xie, "OSCAR: Orchestrating STT-RAM cache traffic for heterogeneous CPU-GPU architectures," in *Proc. 49th IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 28:1–28:13.

[22] A. K. Mishra, O. Mutlu, and C. R. Das, "A heterogeneous multiple network-on-chip design: An application-aware approach," in *Proc. 50th ACM/EDAC/IEEE Des. Autom. Conf.*, 2013, pp. 36:1–36:10.

[23] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-aware prioritization mechanisms for on-chip networks," in *Proc. 42nd IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 280–291.

[24] Y. Jin, E. J. Kim, and T. M. Pinkston, "Communication-aware globally-coordinated on-chip networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 2, pp. 242–254, Feb. 2012.

[25] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Aérgia: Exploiting packet latency slack in on-chip networks," in *Proc. 37th ACM/IEEE Int. Symp. Comput. Architecture*, 2010, pp. 106–116.

[26] Y. Chang, Y. S. Huang, M. Poremba, V. Narayanan, Y. Xie, and C. King, "TS-router: On maximizing the quality-of-allocation in the on-chip network," in *Proc. 19th IEEE Int. Symp. High-Perform. Comput. Architecture*, 2013, pp. 390–399.

[27] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no., pp. 188–201, Apr. 1999.

[28] G. Michelogiannakis, N. Jiang, D. Becker, and W. J. Dally, "Packet chaining: Efficient single-cycle allocation for on-chip networks," in *Proc. 44th IEEE/ACM Int. Symp. Microarchitecture*, 2011, pp. 83–94.

[29] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for vlsi communication switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 1, pp. 13–27, Jan. 1993.

[30] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary CMP workloads," in *Proc. IEEE Int. Symp. Workload Characterization*, 2010, pp. 1–11.

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

[32] S. Rao, S. Jeloka, R. Das, D. Blaauw, R. Dreslinski, and T. Mudge, "Vix: Virtual input crossbar for efficient switch allocation," in *Proc. 51st ACM/EDAC/IEEE Des. Autom. Conf.*, 2014, pp. 103:1–103:6.

[33] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," in *Proc. 32nd ACM/IEEE Int. Symp. Comput. Architecture*, 2005, pp. 420–431.

[34] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *Proc. 34th ACM/IEEE Int. Symp. Comput. Architecture*, 2007, pp. 126–137.

[35] A. Kumar, L. S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *Proc. 34th ACM/IEEE Int. Symp. Comput. Architecture*, 2007, pp. 150–161.

[36] L. S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proc. 7th IEEE Int. Symp. High-Perform. Comput. Architecture*, 2001, pp. 255–266.

[37] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *Proc. 42nd ACM/IEEE Des. Autom. Conf.*, 2005, pp. 559–564.

[38] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proc. 31st ACM/IEEE Int. Symp. Comput. Architecture*, 2004, pp. 188–197.

[39] B. Fu and J. Kim, "Footprint: Regulating routing adaptiveness in networks-on-chip," in *Proc. 44th ACM/IEEE Int. Symp. Comput. Architecture*, 2017, pp. 691–702.

[40] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan, "On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 407–418, Oct. 2012.

[41] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *Proc. 14th IEEE Int. Symp. High-Perform. Comput. Architecture*, 2008, pp. 203–214.

[42] A. Mirhosseini, M. Sadrosadati, B. Soltani, H. Sarbazi-Azad, and T. F. Wenisch, "Binochs: Bimodal network-on-chip for cpu-gpu heterogeneous systems," in *Proc. 11th IEEE/ACM Int. Symp. Netw.-on-Chip*, 2017, pp. 7:1–7:8.

[43] D. Abts, N. E. Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *Proc. 36th ACM/IEEE Int. Symp. Comput. Architecture*, 2009, pp. 451–461.

[44] D. Seo, A. Ali, W. T. Lim, N. Rafique, and M. Thottethodi, "Near-optimal worst-case throughput routing for two-dimensional mesh networks," in *Proc. 32nd ACM/IEEE Int. Symp. Comput. Architecture*, 2005, pp. 432–443.

[45] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, no. 5, pp. 874–902, Sep. 1994.

[46] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 12, pp. 1320–1331, Dec. 1993.

[47] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, no. 5, pp. 547–553, May 1987.

[48] V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu, "Adaptive bubble router: A design to improve performance in torus networks," in *Proc. Inte. Conf. Parallel Process.*, 1999, pp. 58–67.

[49] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 4, pp. 466–475, Apr. 1993.

[50] E. Baydal, P. Lopez, and J. Duato, "A family of mechanisms for congestion control in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 9, pp. 772–784, Sep. 2005.

[51] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, "Goal: A load-balanced adaptive routing algorithm for torus networks," in *Proc. 30th ACM/IEEE Int. Symp. Comput. Architecture*, 2003, pp. 194–205.

[52] E. Baydal, P. Lopez, and J. Duato, "A congestion control mechanism for wormhole networks," in *Proc. 9th Euromicro Workshop Parallel Distrib. Process.*, 2001, pp. 19–26.

[53] K. K. Chang, R. Ausavarungnirun, C. Fallin, and O. Mutlu, "Hat: Heterogeneous adaptive throttling for on-chip networks," in *Proc. 24th IEEE Int. Symp. Comput. Architecture High Perform. Comput.*, 2012, pp. 9–18.

[54] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Performance Anal. Syst. Softw.*, 2009, pp. 163–174.

[55] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 8–23, Oct. 1994.

[56] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 89–102, Aug. 2002.

[57] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall, 1994.

[58] A. Fog, "Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via CPUs," 2018. [Online]. Available: http://www.agner.org/optimize/instruction_tables.pdf

[59] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous CPU-GPU simulator," *IEEE Comput. Architecture Lett.*, vol. 14, no. 1, pp. 34–36, Jan. 2015.

[60] C. Sun *et al.*, "Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. 6th IEEE/ACM Int. Symp. Netw.-on-Chip*, 2012, pp. 201–210.

[61] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in *Proc. Conf. High Perform. Comput. Netw. Storage Anal.*, 2009, pp. 52:1–52:12.

[62] Y. Xue and P. Bogdan, "User cooperation network coding approach for NoC performance improvement," in *Proc. 9th Int. Symp. Netw.-on-Chip*, 2015, pp. 1–8.

[63] Z. Qian, P. Bogdan, G. Wei, C. Y. Tsui, and R. Marculescu, "A traffic-aware adaptive routing algorithm on a highly reconfigurable network-on-chip architecture," in *Proc. 8th Int. Conf. HW/SW Co-Design Syst. Synthesis*, 2012, Art. no. 161–170.

[64] A. Kumary, P. Kunduz, A. P. Singhx, L. S. Peh, and N. K. Jhay, "A 4.6tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65 nm CMOS," in *Proc. 25th IEEE Int. Conf. Comput. Des.*, 2007, pp. 63–70.

[65] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "Pronoc: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors Microsystems*, vol. 54, pp. 60–74, Oct. 2017.

[66] T. Nesson and S. L. Johnsson, "Romm routing on mesh and torus networks," in *Proc. 7th ACM Symp. Parallel Algorithms Architectures*, 1995, pp. 275–287.

[67] M. Tang, X. Lin, and M. Palesi, "The repetitive turn model for adaptive routing," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 138–146, Jan. 2017.

[68] S. Murali, D. Atienza, L. Benini, and G. De Micheli, "A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip," in *Proc. 43rd ACM/IEEE Des. Autom. Conf.*, 2006, pp. 845–848.

[69] Y. S. Yang, H. Deshpande, G. Choi, and P. V. Gratz, "SDPR: Improving latency and bandwidth in on-chip interconnect through simultaneous dual-path routing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 3, pp. 545–558, Mar. 2018.

[70] J. Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management," in *Proc. 20th IEEE Int. Symp. High-Perform. Comput. Architecture*, 2014, pp. 308–319.

[71] S. Van Winkle, A. K. Kodi, R. Bunescu, and A. Louri, "Extending the power-efficiency and performance of photonic interconnects for heterogeneous multicores with machine learning," in *Proc. 24th IEEE Int. Symp. High-Perform. Comput. Architecture*, 2018, pp. 480–491.

[72] D. DiTomaso, A. Sikder, A. Kodi, and A. Louri, "Machine learning enabled power-aware network-on-chip design," in *Proc. Des. Autom. Test Europe Conf.*, 2017, pp. 1354–1359.

[73] M. Clark, A. Kodi, R. Bunescu, and A. Louri, "Lead: Learning-enabled energy-aware dynamic voltage/frequency scaling in nocs," in *Proc. 55th ACM/ESDA/IEEE Des. Autom. Conf.*, 2018, pp. 82:1–82:6.

[74] K. Wang, A. Louri, A. Karanth, and R. Bunescu, "Intellinoc: A holistic design framework for energy-efficient and reliable on-chip communication for manycores," in *Proc. 46th ACM/IEEE Int. Symp. Comput. Architecture*, 2019, pp. 1–12.

[75] H. Zheng and A. Louri, "An energy-efficient network-on-chip design using reinforcement learning," in *Proc. 56th Des. Autom. Conf.*, 2019, pp. 47:1–47:6.

[76] Q. Fettes, M. Clark, R. Bunescu, A. Karanth, and A. Louri, "Dynamic voltage and frequency scaling in NoCs with supervised and reinforcement learning techniques," *IEEE Trans. Comput.*, vol. 68, no. 3, pp. 375–389, Mar. 2019.

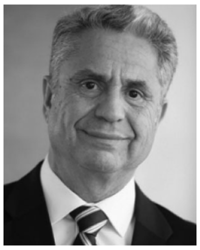
[77] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri, "Dynamic error mitigation in NoCs using intelligent prediction techniques," in *Proc. 49th IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 31:1–31:12.

[78] K. Wang, A. Louri, A. Karanth, and R. Bunescu, "High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learnin," in *Proc. Des. Autom. Test Eur. Conf.*, 2019, pp. 1166–1171.

[79] Y. Xiao, S. Nazarian, and P. Bogdan, "Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 27, no. 6, pp. 1416–1427, Jun. 2019.

[80] Z. Qian, D. Juan, P. Bogdan, C. Tsui, D. Marculescu, and R. Marculescu, "SVR-NoC: A performance analysis tool for network-on-chips using learning-based support vector regression model," in *Proc. Des. Autom. Test Europe Conf.*, 2013, pp. 354–357.

Yuan Li (Student Member, IEEE) received the BS degree in physics from the University of Science and Technology of China, in 2010, and the MS degree in microelectronics from the University of Newcastle upon Tyne, in 2011. He is currently working toward the PhD degree in computer engineering at the George Washington University. His research interests include computer architecture, interconnection networks, accelerator-rich heterogeneous manycore systems, and emerging memory technologies.



Ahmed Louri (Fellow, IEEE) received the PhD degree in computer engineering from the University of Southern California, Los Angeles, California, in 1988. He is the David and Marilyn Karlgaard Endowed chair professor of electrical and computer engineering with the George Washington University, and the director of the High Performance Computing Architectures and Technologies Laboratory. From 2010 to 2013, he served as a program director with the National Science Foundations (NSF) Directorate for computer and information science and engineering. His conducts research in the broad area of computer architecture and parallel computing, with emphasis on interconnection networks, optical interconnects for scalable parallel computing systems, reconfigurable computing systems, and power-efficient and reliable network-on-chips (NoCs) for multicore architectures. He is a fellow of the Institute of Electrical and Electronics Engineers (IEEE), and currently serving as the editor-in-chief of the IEEE Transactions on Computers.