iConn: A Communication Infrastructure for Heterogeneous Computing Architectures

ZHONGQI LI, Qualcomm Inc.
NILANJAN GOSWAMI and TAO LI, University of Florida

Recently, the *graphics processing unit* (GPU) has made significant progress as a general-purpose parallel processor. The CPU and GPU cooperate together to solve data-parallel and control-intensive real-world applications in an optimized fashion. For example, emerging heterogeneous computing architectures such as Intel Sandy Bridge and AMD Fusion integrate the functionality of the CPU and GPU in a single die. However, the single-die CPU-GPU heterogeneous computing architecture faces the challenge of tight budget of die area. The conventional homogenous interconnect fails to provide satisfactory performance by fully exploiting the given area budget in the heterogeneous processing era.

In this article, we aim to implement an interconnect network within an area budget for a CPU-GPU heterogeneous computing architecture. We propose iConn, a 2D mesh-style on-chip heterogeneous communication infrastructure. In iConn, a set of GPU logical units such as the stream processors, the texture units, and the rendering output units form a *computing unit* (CU). Differing from conventional homogenous router design, iConn adopts nonuniform on-chip routers in order to meet the unique communication demands from each single CPU and CU. The routers can also dynamically allocate their buffers across all *virtual channels* (VCs) to meet the latency requirements of CPUs and CUs. Moreover, the memory controller scheduling algorithm is modified from traditional load-over-store scheduling in order to prioritize the traffic. Our simulation results show that iConn improves the performance of CPUs by 23.0% and CUs by 9.4%.

Categories and Subject Descriptors: B.4.3 [**Input/Output and Data-Communications**]: Interconnections; C.5.3 [**Computer System Implementation**]: Microprocessors; C.1.3 [**Processor Architectures**]: Heterogeneous (hybrid) Systems; C.0 [**General**]: Modeling of Computer Architecture

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: CPU, GPU, network-on-chip, heterogeneous computing

ACM Reference Format:
Zhongqi Li, Nilanjan Goswami, and Tao Li. 2015. iConn: A communication infrastructure for heterogeneous computing architectures. ACM J. Emerg. Technol. Comput. Syst. 11, 4, Article 42 (April 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/2700238

1. INTRODUCTION

Contemporary semiconductor technology [Semiconductor Industry Association 2005] is capable of integrating the *central processing units* (CPUs) and *graphic processing units* (GPUs) in a single die as a fused computing architecture. For example, Nvidia's Project Denver [Dally 2011] integrates a 64-bit ARM processor and GPUs in a single processor; Intel's Sandy Bridge [Kanter 2010] has the CPU and GPU on one chip with a shared on-chip L3 cache; and AMD's *accelerated processing unit* (APU) [Winkle 2012; Brookwood 2010] employs off-chip memory [Boudier 2011] to be shared by the CPU and

This work is supported by the National Science Foundation under grants CNS-1423090, CCF-1320100, CCF-0916384, and CCF-0845721 (CAREER).
Authors' addresses: Z. Li (corresponding author), Qualcomm Mobile Computing (QMC), Qualcomm Technologies, Inc., 5775 Morehouse Drive, San Diego, CA 92121; email: zhongqili@ufl.edu; N. Goswami and T. Li, Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2015 ACM 1550-4832/2015/04-ART42 $15.00
DOI: http://dx.doi.org/10.1145/2700238

GPU. Such heterogeneous computing architectures provide the opportunity to leverage both the high computational power from the GPU for regular applications and flexible execution from CPUs for irregular workloads.

In Nvidia's CUDA [Nvidia Corp. 2008], the vertex shaders and pixel shaders are integrated into a unified computing unit called the *streaming processor* (SP). AMD and OpenCL [Khronos OpenCL Working Group 2011] use the term *computing unit* (CU) for a set of GPU logic units, such as the stream processors, the texture units, and the rendering output units in the graphics pipeline. For example, AMD Radeon 7970 [Ryan 2011] has 32 such units, each containing 64 stream processors, 4 texture units, 1 rasterizer operation unit, and 1 rendering output unit. In this study, the CUs are used as the basic processing elements in iConn heterogeneous computing architectures.

These heterogeneous processors pose new challenges to architecture designers. For example, increasing communication demands among CPUs and CUs push more critical performance of the on-chip interconnect network. In order to address this issue, the Intel Sandy Bridge uses a bi-directional ring-style bus, while Nvidia's Project Denver and AMD's Fusion processor adopt a crossbar-style interconnect. These solutions may be suitable for the processor with a small number of cores but, for future heterogeneous computing architectures, are all way too simple.

An on-chip mesh-style network provides a promising alternative to the buses and crossbars in terms of performance and power, however, it faces many new challenges in the heterogeneous computing environment. For instance, the chip area budget is usually tight in a heterogeneous computing architecture [Winkle 2012]. Thus the on-chip network needs to be carefully designed to utilize its area effectively.

The routers in the network [Dally and Towles 2004], which are responsible for receiving packets on its inputs and forwarding packets to the appropriate output, dominate the area of the interconnect network. The buffers in the routers usually occupy more than 75% of the total area of the interconnect [Mishra et al. 2011a]. Most current mesh-style interconnects use uniform routers with the same size of buffer at every router port. However, uniform routers are not the best choice in the heterogeneous computing era, since the CPU and CU possess diametric network demands. CPU cores rely on large caches and a speculative mechanism to achieve high serial execution performance. They usually suffer a large penalty until the memory access is satisfied. By contrast, CUs mainly leverage the single-cycle context switch to remove the latencies incurred by the memory access instructions. Therefore, the traditional homogenous router should be redesigned to match the various communication demands from CPUs and CUs. For example, input buffers connected to those computing cores with less communication demand could be smaller to save some area for communication-intensive cores.

In this article, we propose iConn, a heterogeneous mesh-style communication infrastructure for the heterogeneous computing architecture. The aim of iConn is to alter the conventional uniform routers to better satisfy the various communication demands from different types of cores and varied traffic across the application lifespan. IConn also ensures careful allocation of the resources inside routers and the off-chip memory controllers so that latency-sensitive CPU traffic will not be impeded by massive CU traffic.

In summary, we will attempt to answer the following questions in this article.

(1) How do we design an on-chip network within a limited area to maximize the performance of both CPUs and CUs? To be more specific, how do we allocate buffers to each of the router ports and assign priority levels to each *virtual channel* (VC)?
(2) Since, inherently, CPUs and CUs exhibit different degrees of sensitivity to network latency, how does the aforementioned design affect their performance?
(3) Apart from the interconnect, should the CPU and CU traffic be assigned different priorities in the memory controllers?

The article is organized as follows: Section 2 provides a brief introduction to the heterogeneous computing architecture. Section 3 presents the detailed analysis and implementation of iConn. Section 4 describes the simulation environment and Section 5 presents the simulation results. Section 6 describes related work and Section 7 concludes.

2. CPU-GPU HETEROGENEOUS COMPUTING ARCHITECTURE

GPUs are receiving increasing attention for high-performance parallel computing [Buck et al. 2004] besides conventional desktop and workstation applications. The GPUs' increased popularity has been due in part to a unique amalgamation of performance, power, and energy efficiency [Goswami et al. 2012, 2014].

Traditional CPUs and GPUs communicate through PCI Express (PCIe), which incurs additional overhead costs for CPU-to-GPU data transfers and vice versa. As a consequence, applications which require CPU and GPU co-computing are oftentimes bottlenecked by PCIe data transfers [Vuduc et al. 2010; Bordawekar et al. 2010]. The emergence of heterogeneous computing architectures that aim to "fuse" the CPU and GPU onto the same die, such as AMD Fusion [Dally 2011; Brookwood 2010], Nvidia Project Denver [Dally 2011], and Intel Sandy Bridge [Kanter 2010], brings the issue that the PCIe bottlenecks be addressed. In these architectures, the ×86 or ARM CPU cores and the general-prupose GPU cores share a common path, for example, ring bus or on-chip network, to system memory. Also high-speed block transfer engines assist in data movement between the CPU and GPU cores. Hence, data transfers never hit the external bus, thereby mitigating the adverse effects of slow PCIe.

Figure 1 presents a bird's-eye view and sectional view of a small-scale CPU-CU heterogeneous computing architecture. Each CPU core has its own private L1-I and L1-D caches and an L2 cache shared by all CPU cores. The GPU computing logic units are organized into the CUs consisting of a set of GPU logical units. The organization of memory space in the CPU-GPU heterogeneous computing architecture follows the AMD Fusion architecture [Winkle 2012]. The address space of the main memory is divided into two parts: one part is visible to and managed by the operating system running on the CPUs (host memory) and the other part is managed by the kernel running on the CUs (GPU device memory). IConn follows the current Fusion architecture design where data needs to be moved between the memory managed by the operating system and the portion that is visible to the CUs. The future Fusion architecture has merged memories for the CPUs and CUs in order to avoid the data transfer penalty between the host memory and GPU device memory (zero-copy transfer technique [Gelado et al. 2010; Boudier 2011]).

In iConn, each of the processor cores and memory controllers is connected to the interconnect network through the *network interface* (NI). The data are packetized in the NIs and then transferred hop by hop via the network links based on the decision made by each router. A packet typically encloses a cache line, an invalidation packet, or part of DMA block data. The size of a cache-line packet in iConn is 72 bytes, which contains 64-byte data and 8-byte header. The size of an invalidation packet is 16 byte and the DMA block contains up to 16KB data. In iConn, we assume the NIs break each DMA block into fixed 72-byte length packets.

Two types of traffic exist in iConn. One is between the CPUs and the cache or memory controllers (CPU traffic). The other is between the CUs and the cache or memory controllers (CU traffic).

3. DESIGN OF THE ICONN ARCHITECTURE

The design aim of iConn is to maximize performance of the heterogeneous processor by using a fixed on-chip interconnect area. In order to achieve this goal, iConn embraces

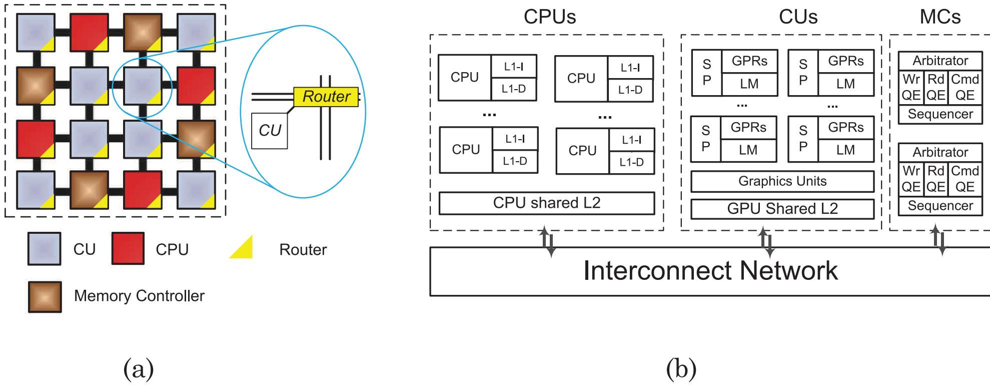


Fig. 1. (a) Bird's-eye view of a typical small-scale CPU-CU computing architecture; (b) sectional view of CPU-CU computing architecture (SP: Shader Processor; LM: Local Memory; GPR: General-Purpose Registers; QE: Queue).

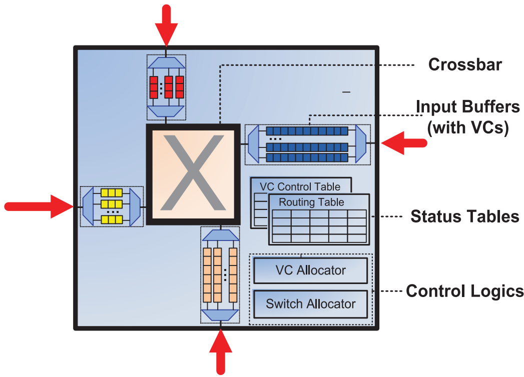


Fig. 2. Block diagram of an asymmetric on-chip router.

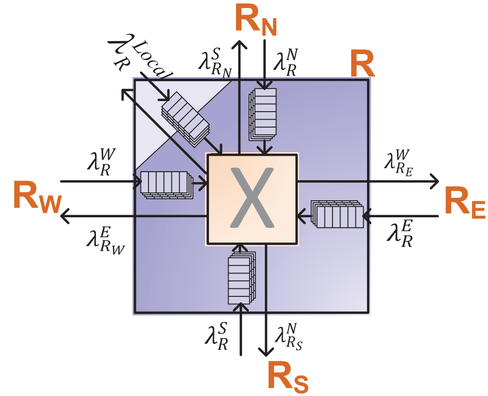


Fig. 3. The router R and rates to/from connected routers.

the design of nonuniform routers which can leverage the buffer resources more efficiently. The appropriate router buffer heavily relies on the amount of through traffic. A router port which connects to the processing core observing heavy traffic load should be assigned a large buffer. This is because larger buffers are able to temporarily store the packets on the path and therefore alleviate the backpressure to the processing core. In this article, we adopt a queuing-theory-based algorithm, aiming to effectively allocate the buffers to different router ports of iConn properly.

This section presents the details of the algorithm. We first introduce the model of routers in NoCs, and then present our buffer allocation algorithm. Afterwards, we introduce the dynamic VC allocation and memory controller scheduling algorithm.

3.1. The Model of a Router

IConn adopts asymmetric routers as depicted in Figure 2. Each input port of the asymmetric routers has its own buffer with unique size. There are no buffers at the output ports; they only employ certain flip-flops to decouple the internal part of the routers with the outside links, which is not the focus of this article.

Each head flit of a packet must proceed through the steps of *bufferwrite* (BW), *routing computation* (RC), *virtual-channel allocation* (VA), *switch allocation* (SA), and *switch*

traversal (ST). A head flit, upon arriving at an input port, is first decoded and buffered in the input buffers according to its input VC in the BW pipeline stage. In the next stage, the routing logic performs RC to determine the output port for the packet. The header then arbitrates for a VC corresponding to its output port in the VA stage. Upon successful allocation of a VC, the header flit proceeds to the SA stage where it arbitrates for the switch input and output ports. On winning the output port, the flit then proceeds to the ST stage where it traverses the crossbar and is stored in the output ports. Finally, the flit is passed to the next node through external links in the *link traversal* (LT) stage. Body and tail flits follow a similar pipeline except that they simply inherit the VC allocated by the head flit. Thus, the time between the header flit of a packet being received by the router and the downstream node starting to receive the packet, without considering contention, can be computed as

$$T = T_{BW} + T_{RC} + T_{VA} + T_{SA} + T_{ST} + T_{LT}. \tag{1}$$

In this article we assume each of the preceding steps to take one clock cycle.

3.2. Buffer Allocation Algorithm

The size of the buffers is of critical importance in the iConn architecture. Sections 3.2 and 3.3 serve the purpose of deriving a heuristic algorithm to solve the size of the buffers based on through traffic.

The area budget of the CPU-GPU heterogeneous computing architecture is usually tight. The interconnect infrastructure takes around 10% to 20% of the whole chip area [Michelogiannakis et al. 2007] and the buffers within the routers occupy over 75% of the total area of the interconnect [Mishra et al. 2011a]. In this study, we assume iConn is manufactured under the future 14nm SOI process. Thus, given the communication amount (obtained from the simulations in Section 4.3) of the processing cores and the total die area, we aim to find the optimized buffer size for all on-chip routers to maximize system performance. The computed buffer arrangement helps in reducing the average end-to-end packet latency in the network and speeds up the execution of applications.

In this article, we perform mathematical analysis on the on-chip routers and traffic to obtain an optimized buffer arrangement. The routers in iConn are modeled using the theory of finite queuing networks [Gross et al. 2008]. For the sake of simplicity, we assume the packets arrive at the routers following the Poisson process. We also assume each router contains only one server and that its job service time (delay in routers without external contention) follows an exponential distribution. Each router then can be modeled as an M/M/1/K finite queue (M: exponential distribution; 1: one server; K: limited buffer size in each router). Although some previous studies [Hu and Marculescu 2004; Coenen et al. 2006; Kodi et al. 2008] applied the queuing model for system performance analysis, most of them focus on homogenous general-purpose computing cores or application-specific *chip multi-processors* (CMPs). Our work extends these studies to the heterogeneous computing architecture.

We summarize the notations of symbols in our analysis in Table I and depict the arrival rates in Figure 3. The router R is connected to four routers at four directions which are named R_S, R_N, R_W, and R_E, respectively. Together the four routers are called $R + 1$. The arrival rates of R from its upstream router are λ_R^{dir}, in which dir is the relative position of its upstream router and can be W, E, S, N, or Local. Similarly, the arrival rate of the downstream routers of router R is λ_{R+1}^{dir}.

Another parameter determining the performance of the interconnect network is the service rates of routers. The service rate of the router R is denoted as μ_R^{dir}. The service rate of the routers is not only determined by the router design, but also by the

Table I. Symbol Notation

T_{RC}	Time delay at RC stage
T_{VA}	Time delay at VA stage
T_{SA}	Time delay at SA stage
T_{ST}	Time delay at ST stage
T_{link}	Time delay at inter-router link
Dir	One of the directions: *E*, *W*, *S*, *N*, or *Local*
R	Router *R*
R_{dir}	Downstream router of the *dir* port of router *R*
λ_R^{dir}	Arrival rate at direction *dir* of router *R*
μ_R^{dir}	Service rate for direction *dir* of router *R*
$s_{R,dir}$	Buffer size of direction *dir* of router *R*
$P_{block(R,dir)}$	The probability that the downstream buffer of direction *dir* of router *R* is full
$\bar{\mu}_{R,dir1}^{dir2}$	The virtual service rate in router R for the data from *dir1* to *dir2*
$\bar{\mu}_{R,dir}$	The virtual service rate in router R for the data to *dir*
$L_{R,dir}$	The queue length of direction dir at router R
$p_{R,dir1}^{dir2}$	The probability that a packet at router R to be delivered from *dir1* to *dir2*

probabilities of whether a packet is routed to the downstream node and the status of the downstream input buffer (whether full or not). As opposed to the arrival rates obtained from real applications, the service rates can only be computed from mathematical analysis. From Gross et al. [2008], it is known that the full probability of the downstream buffer (we use the east port of router R as an example) can be computed by applying the equations for the M/M/1 queuing model (derived from the queuing theory formula which describes the probability that queuing length is greater than the buffer size):

$$\mathrm{P_{block(R,E)}} = \frac{1-\left(\frac{\lambda_R^E}{\mu_R^E}\right)}{1-\left(\frac{\lambda_R^E}{\mu_R^E}\right)^{s_{(R_E),W}}} \cdot \left(\frac{\lambda_R^E}{\mu_R^E}\right)^{s_{R,E}}. \tag{2}$$

The effective service rate of the link can be estimated as $1/\mathrm{P_{block(R,E)}}$, and the total arrival rate of direction W of router $\mathrm{R_E}$ is $\lambda_{R_E}^W$. By using Little's formula, the average waiting time for entering west of router $\mathrm{R_E}$ can be calculated as $\frac{1}{1/\mathrm{P_{block(R,E)}}-\lambda_{R_E}^W}$, in which the arrival rate of the west port of router $\mathrm{R_E}$ can be expressed as $\lambda_{R_E}^W = \sum_{\{directions\}}^{dir} \lambda_R^{dir} p_{R,dir}^W$. So the average waiting time to enter the west port of router $\mathrm{R_E}$ is

$$\frac{1}{1/\mathrm{P_{block(R,E)}} - \sum_{\{directions\}}^{dir} \lambda_R^{dir} p_{R,dir}^W}. \tag{3}$$

The forward probability $p_{R,dir1}^{dir2}$ is predetermined by the routing function. We assume iConn uses fixed X-Y routing in which the value of $p_{R,dir1}^{dir2}$ is either 0 or 1.

Apart from the average waiting time for the west port of router $\mathrm{R_E}$, we also calculate the average waiting time of the router R.

To determine the service rate, we assume there are five non-competition channels at the five ports of router R so that we have five such virtual queues. The arrival rate of each virtual queue can be calculated as $\bar{\lambda}_R^E = \lambda_{R,dir} \times \rho_{R,dir}^E$, where dir is the port of this virtual queue. We assume the service time of this link is $\bar{\mu}_{R,dir}^E$. By using Little's

formula, the average waiting time on this virtual link can be calculated as $\frac{1}{\bar{\mu}^{E}_{R,dir}-\bar{\lambda}^{E}_{R}}$, which can be further derived as

$$\frac{1}{\bar{\mu}^{E}_{R,dir} - \lambda_{R,dir} \times p^{E}_{R,dir}}. \tag{4}$$

Since the average waiting time for the packets to enter the west port of router R_E (Eq. (3)) should be equal to the average waiting time on the virtual queue at the east port of router R (Eq. (4)), the service rate of this virtual queue can be calculated as

$$\bar{\mu}^{E}_{R,N} = {}^{1}\!/P_{block(R,E)} - \lambda^{W}_{R_E} + \lambda_{R,dir} \times p^{E}_{R,dir}. \tag{5}$$

The average service time can be expressed as the sum of all the five downstream channels.

$$\bar{\mu}_{R,N} = p^{E}_{R,N} \times \bar{\mu}^{E}_{R,N} + p^{W}_{R,N} \times \bar{\mu}^{W}_{R,N} + p^{S}_{R,N} \times \bar{\mu}^{S}_{R,N} + p^{N}_{R,N} \times \bar{\mu}^{N}_{R,N} + p^{Local}_{R,N} \times \bar{\mu}^{Local}_{R,N} \tag{6}$$

The service time in a router is a two-part process. The first part of service time is the fixed waiting time, and the second part is the delay caused by contention. For the same reason, the length of the waiting queue also needs to be treated as the sum of two queues which are caused by the fixed waiting time and the contention, respectively. Applying the queuing theory equation to determine the mean queue length, we have

$$L_{R,N} = \frac{\rho}{1-\rho} = \lambda^{N}_{R}\left(\frac{1}{\frac{1}{T} - \lambda^{N}_{R}} + \frac{1}{\bar{\mu}_{R,N} - \lambda^{N}_{R}}\right). \tag{7}$$

After applying Little's formula again, we can obtain the final expression for μ^{E}_{R}, which is

$$\mu^{E}_{R} = \lambda^{N}_{R} + \frac{1}{\lambda^{N}_{R}\left(\frac{1}{\frac{1}{T}-\lambda^{N}_{R}} + \frac{1}{\bar{\mu}_{R,N}-\lambda^{N}_{R}}\right)}. \tag{8}$$

As a result, prior Eqs. (2) through (8) reveal the relationship between the network traffic (arrival rate) and router behavior (blocking probability). This group of linear equations can be solved by using Matlab's nonlinear equation solver.

3.3. Buffer Allocation Schemes

Although the link block probability can be computed based on the traffic rate by leveraging the results from the previous section, it is still unclear how this can be applied to calculate the buffer sizes.

One naïve approach is the exhaustive algorithm which attempts to search all possible combinations of buffer allocations and finds the best solution, however, it leads to *nondeterministic polynomial* (NP) complexity.

We propose to leverage a heuristic algorithm as depicted in Figure 4. The results from our algorithm should be close to the ideal solution. Our algorithm aims to minimize the variation of the average waiting time of each port as described by Eqs. (3) and (4). In our algorithm, we applied a nonlinear equation solver as well as the simulation iterations to determine the size of the buffers.

3.4. Virtual Channel Division

In Section 3.3 we proposed an algorithm to compute all the buffer sizes. The VCs of the ports, which play an important role in modern on-chip networks, should be

ALGORITHM: Buffer Allocation Algorithm

1. Equally allocate the available buffers to every port of all routers.
2. Use the nonlinear equation solver to find the ports with the smallest $\boldsymbol{P_{block(R,dir)}}$ and the biggest $\boldsymbol{P_{block(R,dir)}}$, then assign the buffer of the size of a packet from the least $\boldsymbol{P_{block(R,dir)}}$ port to the highest $\boldsymbol{P_{block(R,dir)}}$ port.
3. Record the current allocations of all buffer sizes and the standard derivation of all $\boldsymbol{P_{block(R,dir)}}$ of all router ports.
4. Run the nonlinear equation solver again.
5. If the new recorded standard derivation of all $\boldsymbol{P_{block(R,dir)}}$ is smaller than before, then repeat step 2 - 4; otherwise use the previous allocation of buffers.

Fig. 4. Buffer allocation algorithm.

utilized effectively as well. In this section we aim to allocate the VCs at each input port efficiently using the size of buffers computed in Section 3.3.

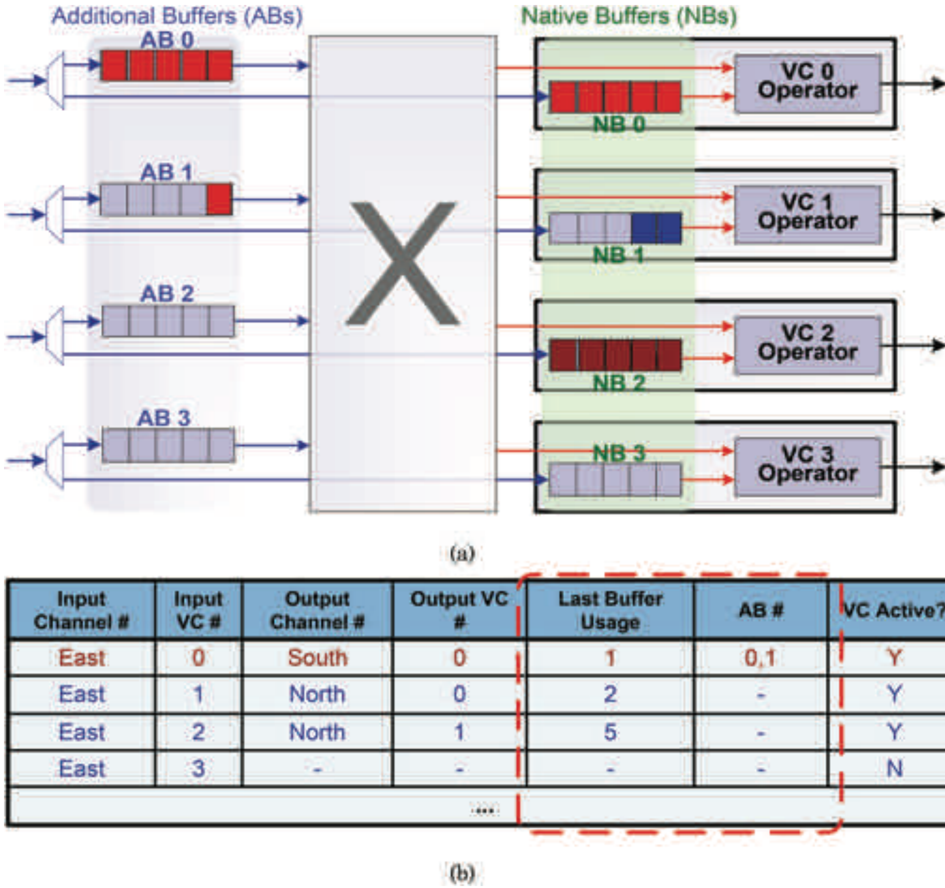
In iConn's VC allocation algorithm, one VC is dedicated to CPU traffic and therefore helps to reserve a certain bandwidth, even in a congested environment with a round-robin VC arbiter. The motivation is based on the fact that CPU workloads are usually more sensitive to latency, while the single-cycle context switch and massive threads can hide the access latency of CU applications to some extent.

The nature of the VC mechanism and fairness of the round-robin arbiter avoid the waste of bandwidth in case of an idle dedicated VC. In conventional design, each VC owns a portion of the buffer at that port. Although the bandwidth can be fully utilized when some VCs are idle, the buffer spaces of these idle VCs are wasted. So we seek to adaptively assign the buffers across all VCs at the same input port. This can be achieved by introducing a centralized buffer shared by all VCs. However, the dedicated VC for CPUs may face starvation since its throughput is usually much smaller than those of the other VCs.

In iConn, we designed a simple architecture suitable for hardware implementation to allocate available VC resources to waiting input requests, as shown in Figure 5. The whole buffer of an input port is equally divided into two halves, namely *native buffers* (NBs) and *additional buffers* (ABs). Each VC has its own private NB and shares the ABs with other VCs.

Each VC may be granted one NB and multiple ABs, depending on the availability of ABs. All buffers granted to one VC are linked together to work as a whole. The last buffer refers to that AB or NB which is most recently assigned to a VC and has nonzero data count. Each VC applies for one extra AB at the cycle when the last buffer usage (packet count) exceeds M. One AB will be granted at the next cycle if any AB is available. The selection of M depends on the actual hardware implementation. Assuming N is the capacity of each buffer, M must be a value smaller than N; in our simulation, M is chosen to be N-1 so that an additional AB will be requested one cycle ahead. An assigned AB will be freed at the cycle when its data count falls to zero and if it is the last buffer.

The usage of ABs and NBs is recorded in the VC control table, as the red dashed square in Figure 5(b) indicates. Figure 5 presents an input port with 4 VCs. Each of the NBs and ABs has a capacity of 5 packets. VC 0 is the VC reserved for CPU traffic, and the other VCs are shared by CUs and CPUs. In this example, there are 11 packets in VC 0, and VC 0 was granted two ABs (0 and 1). VC 1 only has 2 packets stored in NB and therefore has no AB. The last buffer usage in VC 2 is more than 5, so AB 2 is granted. However, since the data count in AB 2 is zero, the last buffer usage section in the VC control table shows the usage of NB 2, as opposed to AB 2.



Input Channel #	Input VC #	Output Channel #	Output VC #	Last Buffer Usage	AB #	VC Active?
East	0	South	0	1	0,1	Y
East	1	North	0	2	-	Y
East	2	North	1	5	-	Y
East	3	-	-	-	-	N
....						

Fig. 5. (a) Dynamic VC structure at an input port; (b) revised VC control table.

3.5. Memory Controller Scheduling Algorithm

Apart from the routers, the memory controllers also deserve careful design to avoid any speed degradation. A memory access may be solved by cache, or it eventually accesses the main memory in the case of a last-level cache miss. Although the dedicated VC improves the latency of CPU traffic, our simulations show that the CPU packets are usually delayed by the overwhelming CU traffic in the memory controllers, provided that only the network routers prioritize the traffic. Thus the memory scheduling algorithm also needs to be modified accordingly.

One widely used memory controller scheduling algorithm, load-over-store scheduling, assigns higher priority to the load memory access than to the store memory access. According to Rixner et al. [2000], load-over-store memory access helps to improve the performance of load-sensitive benchmarks while not hurting the performance of other benchmarks.

We modified the conventional load-over-store scheduling algorithm by assigning the CPU traffic higher priority, namely CPU-over-CU scheduling. As listed in Table II, the traffic is now categorized into four priority levels, improved from the two levels in the original load-over-store scheduling algorithm. This algorithm only requires minor modification and introduces negligible overhead. We will present simulation results showing that the prioritized small amount of CPU traffic will not degrade the performance of CUs much.

Table II. Traffic Priorities in Different Scheduling Algorithms

	Load-over-Store scheduling algorithm	*CPU-Over-CU scheduling algorithm*
CPU Load	0	0
CPU Store	1	1
CU Load	0	2
CU Store	1	3

4. EXPERIMENTAL ENVIRONMENT

To evaluate the performance of iConn, we first collect the traffic trace by running two suites of general-purpose benchmarks on a 16-CPU and 32-CU heterogeneous computing system. Four typical placements of the CPUs, CUs, and on-chip memory controllers are evaluated in this article. The buffer allocation algorithm then uses the traffic trace as the input and computes the buffer sizes. After this, we run the simulation again to compare performance and power consumption under different buffer allocations, VC divisions, and memory scheduling algorithms.

In this section, we will first introduce our CPU-CU integrated simulation framework and then describe the adopted benchmark suites and four different core placements. Finally, we will present the different simulation schemes evaluated in the experiments.

4.1. Simulation Framework

We integrated two simulators, GEM5 [Binkert et al. 2011] and GPGPU-Sim [Bakhoda et al. 2009], into a single framework to evaluate the iConn communication infrastructure. GEM5 is a modular platform simulator for processor microarchitecture, memory, and multi-processor systems. The CPU, memory hierarchy, and interconnect are modeled in GEM5. On the other side, GPGPU-Sim provides a simulation model of contemporary GPUs.

The workflow of our integrated framework simulator is shown in Figure 6. GPGPU-Sim is instantiated as an object within the GEM5 simulator and is slave to GEM5. The two simulators communicate with each other mainly through the subroutines in GPGPU-Sim.

As shown in Figure 6, we hacked the tick() subroutine of the GEM5 simulator by inserting the interface to load the GPGPU-Sim (basically a variation of run_cycle()). The "tick" is the smallest unit of simulation step. It differs in GEM5 and GPGPU-Sim because CPUs and CUs operate at separate frequencies. To simulate the CPU and CU simultaneously, both the cycles of CPU and CU should be the integral multiple of a tick. We choose 0.1ns, which is 1/3 a cycle of the CPU (CPUs work at 3.3 GHz) and 1/5 a cycle of the CU (CUs work at 2 GHz). The tick () function of the GEM5 simulator finishes all CPU operations and then loads GPGPU-Sim. GPGPU-Sim returns to GEM5 after finishing execution, and then GEM5 starts Ruby. The modified Ruby interface allocates the traffic data from GEM5 and GPGPU-Sim to the memory hierarchy and the interconnect. Finally, GPGPU-Sim writes back all the registers.

We use the 16 Alpha-21264 CPUs configured with timing simple mode [Binkert et al. 2011] in GEM5 to simulate the CPU model and timing of memory references. We also simulated 32 CUs using GPGPU-Sim v3.0.0b [Bakhoda et al. 2009]. The detailed configuration of the CPUs, CUs, and interconnect is listed in Table III.

The simulated interconnect network consists of 64 routers in total. Before applying the buffer reallocation algorithm, each input port has a 32-packet buffer. The total buffer size of iConn is therefore 737,280 bytes. We calculated the aggregated area of all buffers from CACTI 6.5 [Muralimanohar and Balasubramonian 2009] and scaled it down to 14nm technology node. Under a 14nm technology node, the buffer size is computed to be 3.01mm^2. We then apply different buffer relocation algorithms to rearrange the buffers to each router port and compare their performance.

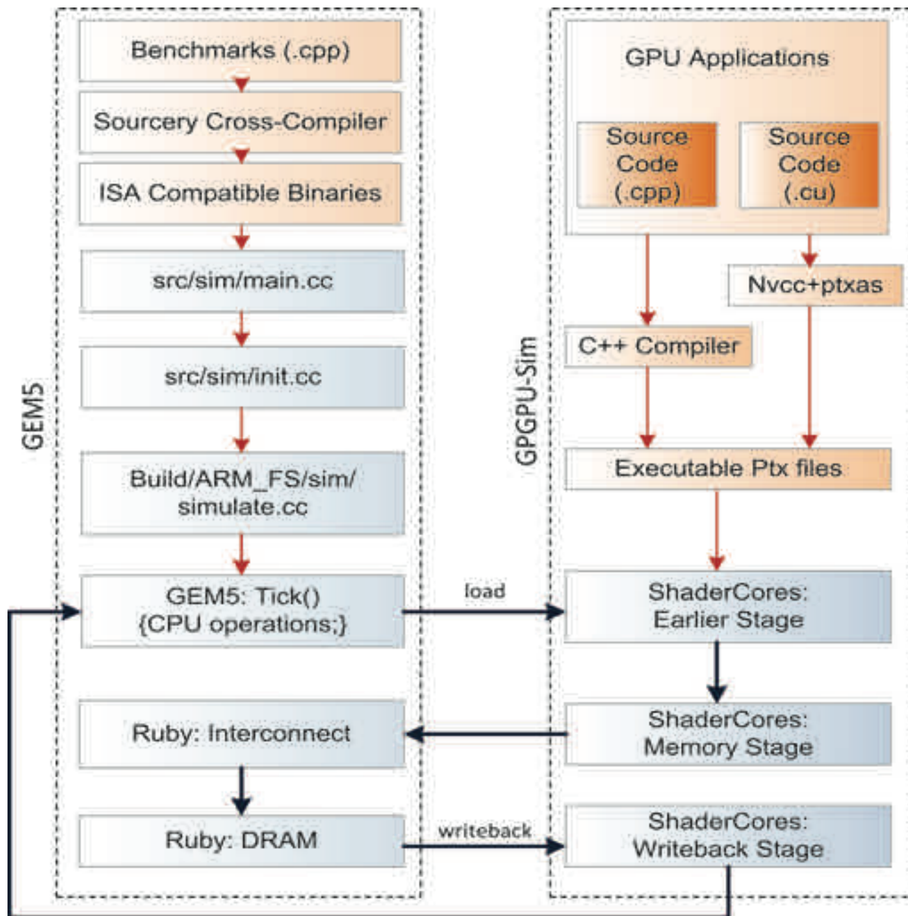


Fig. 6. Simulation flow with GEM5 and GPGPU-Sim.

4.2. Test Benchmarks

In our simulation, we applied CPU- and CU-specific benchmarks to simulate real-world CPU and CU activities. For the CUs, we adopt the CPU-CU benchmarks as listed in Table IV. All benchmarks are chosen from the Nvidia CUDA [Nvidia Corp. 2008] or AMD SDK [AMD Inc. 2014]. Although only general-purpose computing benchmarks are evaluated, the conclusions in this article are also applicable to specific applications such as triangular functions, graphic, and video/audio compression applications.

We run mixed SPEC 2006 [Henning 2006] and PARSEC [Bagrodia et al. 1998] benchmarks on CPUs as listed in Table V. The SPEC 2006 benchmark suite was originally designated to stress a single processor and the memory. In order to test multicore performance, we run SPEC 2006 under "rate" mode, which executes multiple copies of the same benchmark simultaneously. Meanwhile, the multicore benchmark PARSEC stresses both the memory hierarchy and interconnect network between the cores.

In our simulations, each set of the CPU and CU benchmarks is composed of four parts, marked as ❶❷❸❹ for CUs and ①②③④ for CPUs in Table VI. Each part is executed on 4 CPUs and 8 CUs. The operating system performs the mapping for CPU, and the CU runtime system maps CU kernels to different CUs. GPGPU-Sim implements a fake CUDA runtime library and AMD SDK that diverts all driver-level API

Table III. Configuration of Processors and Interconnect

Configuration of the Alpha CPU			
L1 Cache Size	32 KB Instruction + 32 KB Data, 8 way, 64 Byte/line, 2 ports, 1ns latency		
L2 Cache Size	1536 KB Instruction + 1536 KB Instruction (Shared by 24 cores), 8 way, 64 Byte/line, 2 ports, 20 ns latency		
Execution Unit	In-Order (Timing simple model)		
Main Memory	16GB, x8 chip, 1000MHz DDR3 channel, tRCD = 10, tRRD = 4, tRC = 34, tCCD = 4		
Cache coherence	MESI_CMP_directory		
	GPGPU Cores	**Network Configuration**	
Number of CUs	32	Total Buffer Area	3.012 mm^2
Warp Size	**32**	Flit Size	128 bits
SIMD Pipeline Wide	8	Entries per Buffer	32
Number of Threads	512	Routing	X-Y fixed
Number of CTAs / Core	4	Packet Size	72 bytes
Number of Registers	16384		
Texture Cache Size	8KB (2-way set assoc. 64B lines LRU)		
Texture Cache Size	Core 64KB (2-way set assoc. 64B lines LRU)		

Table IV. Testbenches on CUs

Benchmark	Configuration
Breadth First Search (BFS)	Graph size of 64K
Fast Wash Transform (FWT)	Vector of length 2^{15} with 2^7 kernels
Gaussian Elimination (GS)	Number of coefficients is 100 × 100
Hot Spot (HS)	Grid area of 50 × 6 in 60 iterations
3D Laplace Solver (LPS)	Grid size of 100 × 100 × 100
Matrix Multiplication (MM)	Multiplication of 144 × 352 and 352 × 400 matrix
Matrix Transpose (MT)	Matrix size 32 × 512
Path Finder (PF)	32 × 32 grid with weight 10
Ray Trace (RAY)	Image size 256 × 256
Speckle Reducing Anisotropic Diffusion (SRAD)	128 × 128 in 25 iterations
Hybrid Sort (HY)	Dataset size of 2^9
Similarity Score (SS)	256 points with 128 features
Nearest Neighbor (NE)	3 nearest neighbor calculations
Parallel Prefix Sum (SLA)	Number of elements 32
AES encryption (AES)	256KB picture encryption using 128-bit encryption
64 Bin Histogram (64H)	134217728 bytes in 5 time step
Galerkin Solver (DG)	3D version with 6th order polynomial

calls to simulated library implementation of the API. It also implements thread block scheduling across several shader cores [Nvidia Corp. 2008]. We run the applications for 5 billion cycles with 0.5 million warmup cycles and then rotate all the four parts of CU and CPU tasks 90 degrees clockwise to different processors on the chip. After repeating the rotation three times, the four dumped traffic traces are averaged and then used as the input of the buffer allocation algorithm.

4.3. Placement of the Cores and Memory Controllers

The placement of CPUs, CUs, and memory controllers in heterogeneous computing system significantly affects the performance. Modern processor packaging allows sufficient escape paths between the memory controllers and main memories from anywhere

Table V. Testbenches on CPUs

Group	Benchmarks	Configuration
SPEC 2006	401. bzip	train test set
	403. gcc	train test set
	458. sjeng	train test set
	470. lbm	train test set
	433. milc	train test set
	459. GemsFDTD	train test set
	429. mcf	train test set
	436. cacbusADM	train test set
PARSEC	Blackscholes	65,536 options
	Swaption	16 swaptions, 20,000 simulations
	Freqmine	990,000 transactions
	x264	128 frames, 640 × 360 pixels

Table VI. Combination of CPU and CU Testbenches

Test Sets	CU Tasks		CPU Tasks	
	CU Benchmarks		SPEC 2006	PARSEC
1	BFS❶	FWT❷	401. bzip①	Blackscholes②
	GS❸	HS❹	436. cacbusADM③	swaption④
2	LPS	MM	429. mcf	freqmine
	MT	PF	459. GemsFDTD	x264
3	Ray	SRAD	433. milc	Blackscholes
	HY	SS	470. lbm	freqmine
4	NE	SLA	458. sjeng	Blackscholes
	AES	64H	403. gcc	freqmine
5	DG	SS	401. bzip	x264
	BFS	LPS	403. gcc	Freqmine
6	Ray	NE	458. sjeng	Blackscholes
	DG	FWT	470. lbm	x264
7	MM	SRAD	433. milc	Freqmine
	SLA	GS	459. GemsFDTD	Swaption
8	MT	HY	429. mcf	Freqmine
	AES	HS	436. cacbusADM	Blackscholes

on the chip. Thus, an efficient placement should target the minimization of link contention and network latency through reasonable arrangement of the processing cores and memory controllers [Abts et al. 2009]. IConn employs a 2D radix-8 mesh-style on-chip network which contains 16 memory controllers, 16 CPUs, and 32 CUs. The number of processing cores is scaled up from AMD's Kaveri APU [Silcott and Blaszczyk 2013] which contains up to four 28nm steamroller CPU cores, and 8 CUs consisting of 512 stream processors. In iConn, the 16 memory controllers are connected to the off-chip memories though on-chip pins.

In this article, we evaluate our algorithm using four representative placement scenarios as shown in Figure 7. Although only four typical placement scenarios are evaluated in this article, our algorithm is highly scalable and can be easily applied on any number of processing cores and arbitrary placement. The mapping scenarios are chosen based on three observed facts and the conclusions in Mishra et al. [2011b], Abts et al. [2009], and Bakhoda et al. [2010].

(1) CPUs exhibit better performance if located close to each other due to their latency-sensitive nature.

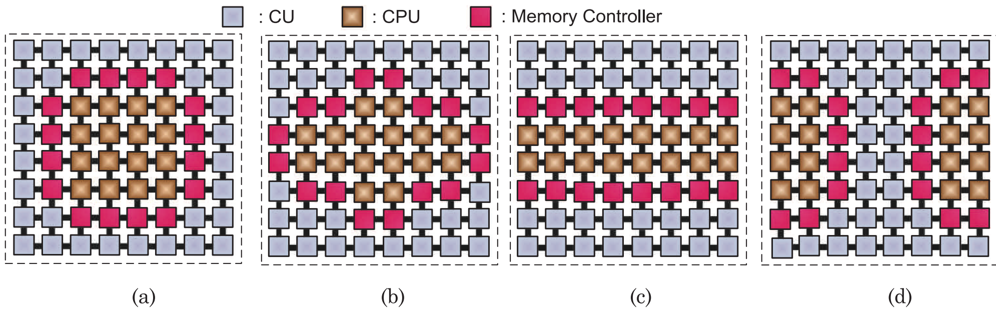


Fig. 7. Placements of CPUs, CUs, and memory controllers.

(2) CUs are not as sensitive as CPUs to the locations. CUs can be set apart but still maintain satisfactory performance. Besides, CU data is more likely to be streamed through main memories rather than caches, that is, the memory controllers on chip.
(3) iConn exhibits better performance by placing the memory controllers at the central part of the chip rather than at the corners.

We assume the directory controllers [Sorin et al. 2011] are attached to CPUs and CUs. We also assume the directory controllers and potential snooping controllers/filters are small and occupy negligible area in our experiments.

4.4. Simulation Schemes

In the simulation part, we compare the results from our buffer allocation algorithm with those of several other buffer allocations. Since only limited studies are available for this topic, we compared our design with a naive approach (LIN) and two approaches proposed in Mishra et al. [2011b] (DUAL and DUAL-L) as listed in Table VII. In our simulation, the total size of the buffers of all router ports is kept the same (more details will be provided in Section 5.1), and we investigate different arrangements of the buffers aiming to maximize performance. The baseline approach uniformly distributes the buffers among all router ports (UNI). In another naïve approach, the buffer size of each router port is linearly proportional to its through traffic amount (LIN). In Mishra et al. [2011], the authors utilize two types of routers: big routers and small routers based on the simulation results. Though Mishra et al. [2011b] is a research merely facing a homogenous CMP environment, we applied its philosophy for comparison. Assuming the number of VCs is x, we chose 16 out of 64 routers with higher traffic throughput and set their VC number as $1.5x$. The remaining 48 routers have $0.5x$ VCs (DUAL). Then, we use the same strategy as Mishra et al. [2011b] by adjusting the width of half of the links with higher traffic from original 128 bit to 160 bit, and the other half of the links to 90 bit (DUAL-L). Doing so has no impact on the bi-sectional bandwidth. In DUAL and DUAL-L, all VCs have the same size of buffers so those channels with more VCs gain more buffers. We then applied our buffer algorithm in BA to compare with the aforementioned schemes. In BA, the VCs at different router ports have different sizes of buffers.

In addition to the buffer allocation schemes, we also evaluate the following VC allocation schemes:

(1) no dedicated channel for CPU and CU (Equal);
(2) always reserve one dedicated VC for CPUs (CPU);
(3) reserve one dedicated VC for CPUs and dynamically allocate the buffers across VCs of a same port (CPU/dyn).

Table VII. Different Approaches Simulation

Buffer Allocation Algorithms	
UNI	Uniform Buffer Size (Baseline)
LIN	Buffer size linearly proportional to traffic
DUAL	Two types of buffer determined by traffic
DUAL-L	Two types of buffer determined by traffic, with link width proportional to buffer size [Mishra et al. 2011]
BA	Our buffer allocation algorithm
VC Allocation Schemes (Default: Equal, n = 4)	
(Equal, n)	CPUs and CUs share all the n VCs; the traffic are treated equally.
(CPU, n)	n VCs, one out of n VCs is reserved for CPU traffic.
(CPU/dyn, n)	n VCs, one out of n VCs is reserved for CPU traffic; dynamically assign buffers across VCs
CPU-over-CU Memory Access Scheduling (Default: load-over-store)	
-MEM	CPU traffic has higher priority than CU traffic based on traditional load-over-store scheduling

Finally, our CPU-over-CU memory scheduling algorithm (-MEM) is examined to compare against a baseline algorithm (load-over-store) as well.

5. SIMULATION RESULTS

In this section, we first present the results from our buffer allocation algorithm in Section 5.1. Then we evaluate the *instructions-per-cycle* (IPC) speedup against other buffer allocation algorithms in Section 5.2. Section 5.3 evaluates system performance by varying the number of VCs and the VC allocation algorithms. Section 5.4 presents the breakdown of memory access latency and the benefits of our memory scheduling algorithms. Section 5.5 exhibits the breakdown of the power consumption.

5.1. Distribution of NoC Buffer Size

In this section, we present the computation results from the buffer allocation algorithm proposed in this article. The buffer allocation algorithm is applied on the four difference placement scenarios described in Section 4.3. Each figure demonstrates the contour plot of the size of the buffers normalized to those routers with the smallest buffer. For the sake of simplicity, we only present the average buffer size of each router. Note that these figures do not represent the actual layout of the nonuniform routers. They only indicate the quantitative results of the buffer allocation algorithm.

As we can observe from Figure 8, those routers connecting to the CUs generally require larger buffers than their counterparts connecting to the CPUs. In each of the four schemes, the largest buffers are 3.6×, 3.2×, 3.1×, and 3.7× larger than the smallest buffers, respectively. However, although the CPUs in the center area usually generate less traffic, the routers in that region still need a certain amount of buffers. This is because the packets tend to be routed through the center area of the chip rather than the edges or the corners in X-Y routing [Dally and Towles 2004]. From the perspective of the chip layout, scenario (c) is the easiest since the buffer distribution is the “flattest” among the four.

5.2. IPC Speedup under Different Buffer Allocation Algorithms

In this section we compare the performance of the system under different buffer reallocation algorithms in Table VII, by executing the eight test sets listed in Table VI. The

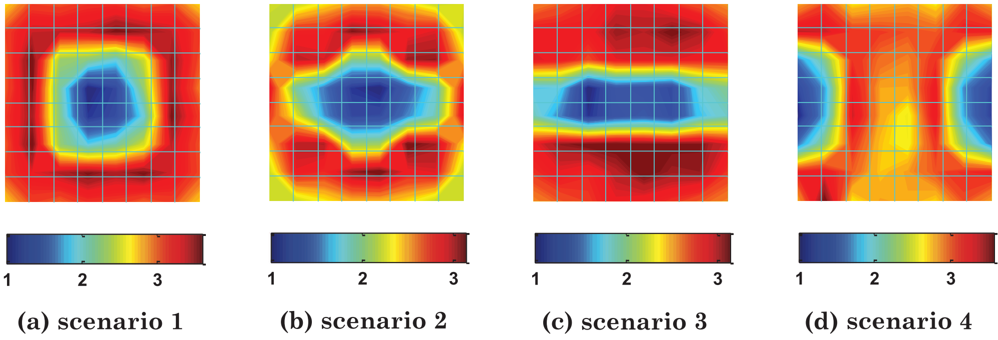


Fig. 8. Contour plot of normalized buffer sizes in NoC under different placement scenarios.

simulation results are shown in Figure 9, where the IPCs of CPUs and CUs are used as the speedup metric.

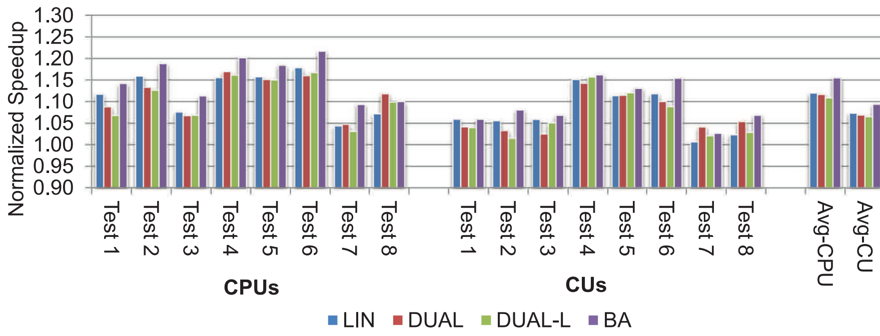
We observe from Figure 9 that the nonuniform buffers in routers can improve the performance of the system, although the CPUs and CUs benefit differently. The naïve linear algorithm (LIN) increases the IPC of CPUs and CUs by an average of 12.9% and 8.4% for the four scenarios, respectively. We then compared the two design philosophies (DUAL and DUAL-L) derived from Mishra et al. [2011b]. The VC number in this simulation is 4, so the wide channels in DUAL and DUAL-L have 6 VCs, and the narrow channels are assigned 2 VCs. The DUAL improves the IPC of CPUs and CUs by 12.7% and 7.4%, respectively. When applying the different link width with DUAL, we can see that DUAL-L improves the IPC by another 0.3% for CPUs but effects nearly no change for CUs. This limited improvement indicates that, in a CPU-GPU heterogeneous computing architecture, the relatively higher traffic compared with conventional CMPs [Mishra et al. 2011b] makes the impact of the link width very limited. Instead, the size of the buffers has a greater effect on the system performance.

We evaluated our buffer relocation algorithm (BA) against the baseline, DUAL, and DUAL-L schemes. The simulation results show that our algorithm achieves 15.9%, 3.0%, and 2.4% improvement for CPUs compared with the baseline, DUAL, and DUAL-L; and gives 8.4%, 3.1%, and 3.2% improvement for CUs compared with the baseline, DUAL, and DUAL-L.

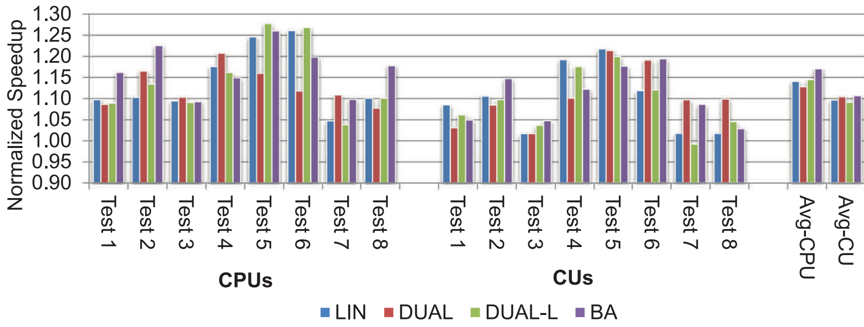
Memory-intensive benchmarks benefit more from the buffer reallocation than memory-nonsensitive applications. Among all the tests, test 4 exhibits the most significant performance improvement. This is because both the CPU and CU benchmarks in test 4 are sensitive to memory activities and therefore it yields heavy network traffic (both cache coherence traffic and memory access traffic). So our buffer allocation algorithm improves the performance more than other tests. We can also see that CUs benefit less than CPUs from the buffer allocation algorithms since the internal wavefronts scheduling algorithm of shader processors is able to hide a significant portion of network latency.

5.3. Performance Improvement with Multiple VCs

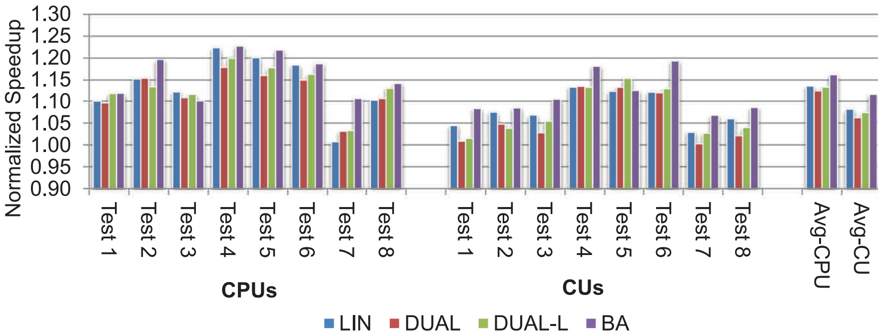
In this simulation, we evaluate the performance by applying different VC allocation schemes and adjusting the number of VCs. The buffer size used in this section is obtained from the BA algorithm. The names of the VC allocation schemes in Table VII are abbreviated to the test cases in Figure 10. In these test cases, the first character "n" is Equal, "c" is CPU, and "d" is CPU/dyn. The digit represents the number of VC(s); for instance, "d4" in Figure 10 stands for (CPU/dyn, 4) in Table VII.



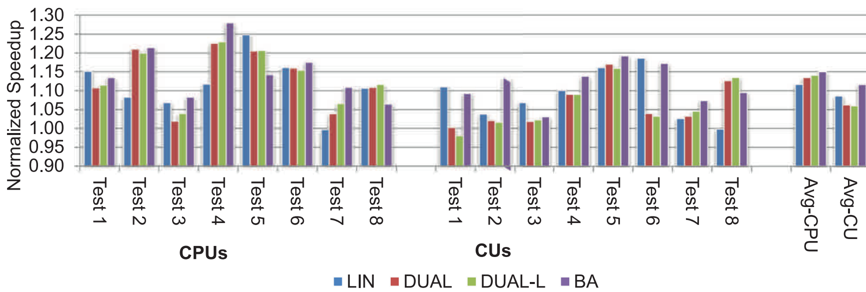
(a) scenario 1



(b) scenario 2



(c) scenario 3



(d) scenario 4

Fig. 9. Normalized IPC improvement under different buffer allocation schemes (baseline: UNI).

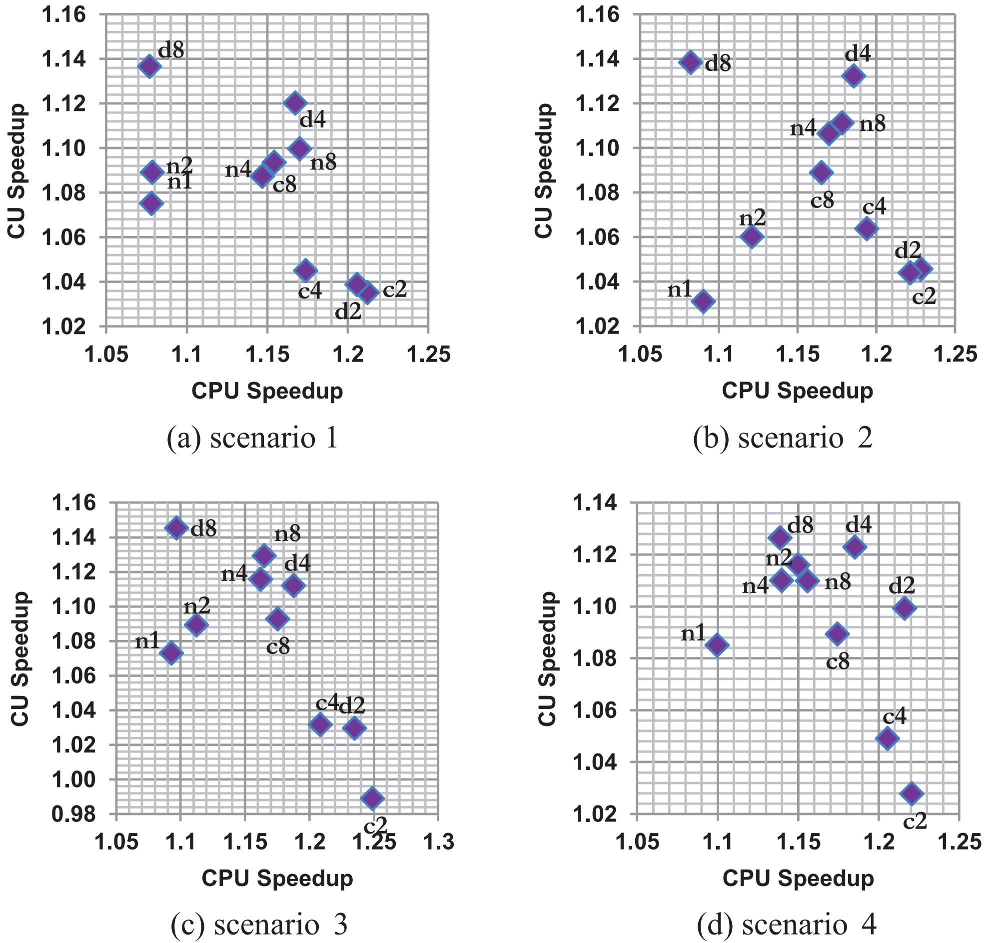


Fig. 10. System performance under different VC numbers and VC allocation schemes.

We first observe from Figure 10 that multiple VCs boost the performance of both CPUs and CUs. Among the Equal (n1 – n8) schemes, the CPUs more greatly benefit from the increased VCs (1.2% for 2 VCs, 5.2% for 4 VCs, and 6.2% for 8 VCs) than the CUs (1.2%, 2.6%, and 3.1%, respectively) by average of the four scenarios. The result is mainly because the CPUs are more sensitive to network latency and multiple VCs help avoid massive CU traffic.

Giving a dedicated VC for the CPUs (CPU), the performance of the CPUs is further improved by 12.7% under 2 VCs, 9.5% under 4 VCs, and 5.9% for 8 VCs compared with Equal by average of the four scenarios. However, the speedup of CPUs comes at a price of slight CU slowdown. As we observed, the 2-VC and 4-VC schemes degrade the performance of CUs by 5.4% and 3.1% compared with Equal, respectively, although the 8-VC network still manages to maintain a 0.8% speedup.

As introduced in Section 3.4, the CPU/dyn scheme is hence used to address the slowdown issue. It shows an average improvement of 11.1% for 2-VC, 7.7% for 4-VC, and 2.2% for 8-VC channels compared with Equal. In the CPU/dyn scheme, the CUs show satisfactory performance, where only the 2-VC scheme is degraded by 2.7% by average of the four scenarios. The 4-VC and 8-VC schemes gain 3.8% and 5.1% improvement,

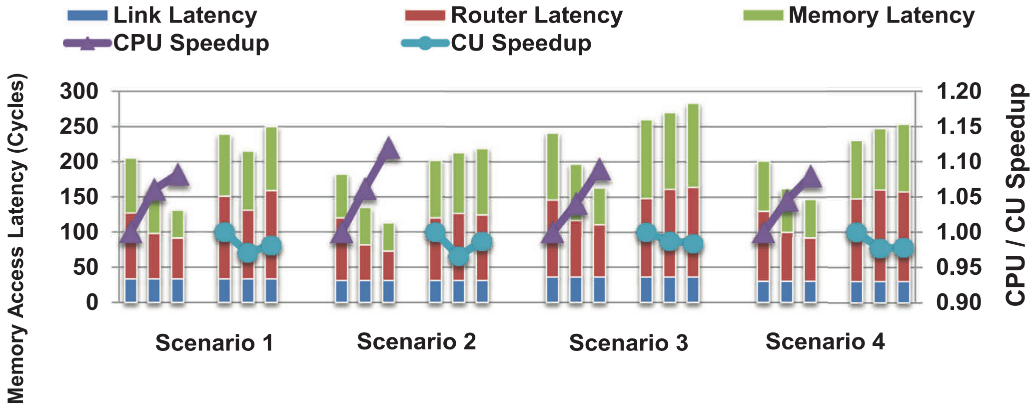


Fig. 11. Memory access latency and speedup under different memory scheduling algorithms (from left to right in each scenario: CPU memory access under Equal, CPU memory access under CPU/dyn, CPU memory access under CPU/dyn-MEM, CU memory access under Equal, CU memory access under CPU/dyn-MEM, CU memory access under CPU/dyn-MEM).

respectively. Among all schemes, (CPU/dyn, 4) exhibits the best performance as both the CPUs and CUs achieve satisfactory performance.

5.4. Performance Improvement under Different Memory Scheduling Algorithms

Sections 5.1 through 5.3 examine system performance by taking various approaches in the interconnect network. In this section, we evaluate our memory scheduling algorithm proposed in Section 3.5 in a 4-VC network with our buffer allocation algorithm (BA). The memory access latency and overall CPU and CU speedup are examined, as presented in Figure 11. The IPC is still used as the metric of CPU/CU speedup.

Generally, the memory access packets experience several different types of delay, among which the *link latency* is the average time consumed on the NoC links. The *router latency* is the time consumed in the network routers, which is mainly due to the temporary storage of packets and downstream resource contention. The *memory latency* includes the latency of the memory controllers and the off-chip memory access delay.

We observe from Figure 11 that router and memory latency dominate the overall memory access latency (45% and 38% by average of the four scenarios, respectively).

Our priority-based dynamic VC division algorithm and memory controller scheduling algorithm address the two respective types of latency. The virtual channel division proposed in Section 3.4 helps in reducing the router latency. For example, by average of the four scenarios, CPU/dyn reduces the router latency of CPU packets by 31.3%, but only insignificantly reduces the memory latency by 14.5% compared with Equal.

The CPU-over-CU memory scheduling algorithm (suffix of “-MEM”) in Section 3.5 is proposed to address this issue. Compared with CPU/dyn and CPU, the CPU-over-CU algorithm reduces 18.5% and 33.4% of the memory access latency by average of the four scenarios. And accordingly, the CPU performance is improved by 3.9% and 9.3%, respectively.

The CU performance degrades to a limited extent when applying CPU/dyn and CPU/dyn-MEM. By average of the four scenarios, the memory access latency of CUs is increased by 1.5% and 7.9% in CPU/dyn and CPU/dyn-MEM compared with Equal, respectively. The moderate memory access latency leads to 2.5% and 1.7% IPC degradation for the CUs. The CPU/dyn-MEM scheme still exhibits 2.2% (not shown in Figure 11) improvement over the (Equal, 1) scheme where no VC mechanism is used.

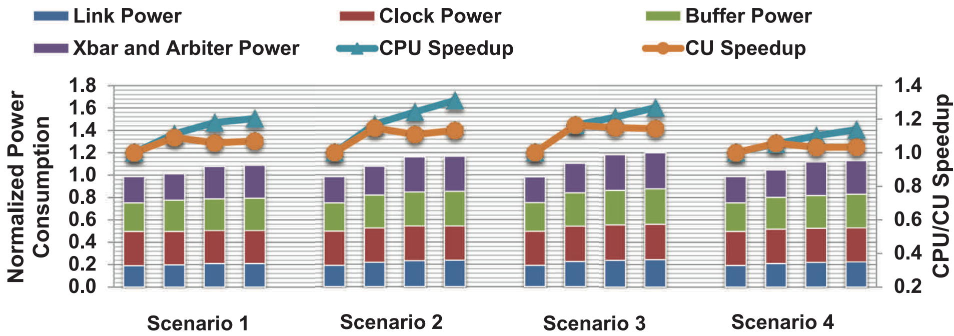


Fig. 12. Power breakdown of different scenarios (from left to right in each scenario: UNI (equal), BA (equal), BA (CPU/dyn), BA (CPU/dyn)-MEM).

5.5. Network Power Breakdown and Overall Performance Improvement

We summarize the overall performance improvement in Figure 12. In a 4-VC network, applying all the techniques (buffer allocation, priority-based dynamic VC allocation, and CPU-over-CU memory scheduling) leads to 23.0% performance improvement in CPUs and 9.4% in CUs, by average of the four scenarios.

We analyze the breakdown of network power consumption in Figure 12. The power consumption is obtained from ORION 6.5 [Kahng et al. 2009]. However, ORION lacks the ability to model imbalanced port buffers in a NoC system. Thus, we elect to use CACTI 6.5 [Muralimanohar and Balasubramonian 2009] to model the buffers and then add the power consumption on top of the other parts of the network, which is calculated from ORION. A major power overhead of CPU/dyn originates from the extra crossbars. This part of power is obtained using the power report from Synopsys Design Compiler [Bhatnagar 1999].

By average of the four scenarios, the link power, clock power, buffer power, and crossbar + arbiter power contributes 19.8%, 29.4%, 26.4%, and 25.2% to the total power consumption in BA (CPU/dyn)-MEM. The overall power consumption of BA (CPU/dyn) and BA (CPU/dyn)-MEM increases by 14.5% and 15.6% compared with UNI (equal). The power consumption is mainly due to the speedup of execution, which leads to faster data transmission.

The power increase of BA (CPU/dyn) from BA (CPU) mainly results from the newly added crossbars at each of the router ports. The newly added crossbar increases the router power by 11.6% and therefore increases the overall power consumption by 7.1% compared with the BA (CPU) scheme.

6. RELATED WORK

Many recent studies focus on the design of a CPU and GPU integrated computing architecture. For example, Yang et al. [2012] proposed a CPU-assisted pre-execution algorithm to accelerate the execution of GPUs. Although some other researchers [Lee et al. 2009; Becker et al. 2009; Brown et al. 2012; Spafford et al. 2012; Power et al. 2013; Cao et al. 2014] studied the heterogeneous computing system, they mainly focus on the applications, simulation, or the mapping algorithm.

Most previous work [Yuffe et al. 2011; Saha et al. 2009; Lee et al. 2012] used a ring-style network to connect CPU and GPU cores. This is only feasible when the number of cores is small. In order to handle future large sets of processing cores and heavy communication burdens, an on-chip network is necessary within the chip.

Only few existing works focus on the interconnect network structure. For example, Mishra et al. [2011b] propose to use heterogeneous routers and links to satisfy

varied communication demands at different areas of a chip multi-processor. Compared with Mishra et al. [2011b], our article provides better granularity. Moreover, Mishra et al. [2011b] only adopt two types of routers: big and small. By contrast, our design assigns the size of buffers at finer granularity. Besides, Mishra et al. [2011b] target only homogenous CMPs with a fixed number of cores (e.g., 64). Our work focuses on the heterogeneous computing system. To our best of our knowledge, our work is the first which explores the characteristics of the on-chip network in the heterogeneous computing environment.

7. CONCLUSIONS

In this article we proposed the iConn communication structure which provides an optimized interconnect design in the CPU-GPU integrated computing architecture. In iConn, we investigated the heterogeneous router architecture, aiming to better utilize the tight-budget on-chip resources. We propose to relocate the buffers across all routers following a finite queuing-network-based mathematical model. We also propose to use priority-based VC allocation to guarantee bandwidth for latency-sensitive CPUs in a congested environment. Also, the CPU-over-CU memory controller scheduling algorithm further helps the priority-based VC allocation. Our simulation shows that the performance of CPUs and CUs improves by 23.0% and 9.4%, respectively, after applying all the techniques.

REFERENCES

Dennis Abts, Natalie D. Enright Jerger, John Kim, Dan Gibson, and Mikko H. Lipasti. 2009. Achieving predictable performance through better memory controller placement in many-core CMPs. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09).*

Amd Inc. 2014. APP SDK – A complete development platform. http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/ (Last accessed 2/2015).

Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu-An Chan, Xiang Zeng, Jay Marting, and Ha Yoon Song. 1998. Parsec: A parallel simulation environment for complex systems. *Comput.* 31, 10, 77–85.

Ali Bakhoda, John Kim, and Tom Aamodt. 2010. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'10).*

Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceeding of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09).*

Aaron Becker, Isaac Dooley, and Laxmikant Kale. 2009. Flexible hardware mapping for finite element simulations on hybrid CPU/GPU clusters. In *Proceedings of the Symposium on Application Accelerators in HPC (SAAHPC'09).*

Himanshu Bhatnagar. 1999. *Advanced ASIC Chip Synthesis: Using Synopsys' Design Compiler and PrimeTime*. Kluwer Academic.

Nathan Binkert, Bradford Beckmann, Steven K. Reinhardt, Gabriel Black, Ali Saidi, et al. 2011. The gem5 simulator. *ACM SIGARCH Comput. Archit. News* 39, 2.

Rajesh Bordawekar, Uday Bondhugula, and Ravi Rao. 2010. Can CPUs match GPUs on performance with productivity? Experiences with optimizing a flop intensive application on CPUs and GPU. Res. rep. RC25033, IBM T. J. Watson Research Center.

Pierre Boudier. 2011. Memory system on fusion APUS - The benefits of zero copy. *AMD Fusion Developer Summit.*

Nathan Brookwood. 2010. AMD fusion family of APUS: Enabling a superior, immersive PC experience. *Insight* 64, 1–8.

Michael Brown, Axel Kohlmeyer, Steven Plimpton, and Arnold Tharrington. 2012. Implementing molecular dynamics on hybrid high performance computers particle-particle particle-mesh. *Comput. Phys. Comm.* 183, 3, 449–459.

Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. 2004. Brook for GPUs: Stream computing on graphics hardware. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'04).* ACM Press, New York, 777–786.

Wei Cao, Chuan-Fu Xu, Zheng-Hua Wang, Lu Yao, and Hua-Yong Liu. 2014. CPU/GPU computing for a multi-block structured grid based high-order flow solver on a large heterogeneous system. *Cluster Comput.* 17, 2, 255–270.

Martijn Coenen, Srinivasan Murali, Andrei Ruadulescu, Kees Goossens, and Giovanni De Micheli. 2006. A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control. In *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS'06).* 130–135.

Bill Dally. 2011. Project Denver processor to usher in new era of computing. http://blogs.nvidia.com/2011/01/project-denver-processor-to-usher-in-new-era-of-computing/ (Last accessed 3/2012).

William Dally and Brian Towles. 2004. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Fransisco.

Isaac Gelado, John E. Stone, Javier Cabezas, Sanjay Patel, Nacho Navarro, and Wen-Mei W. Hwu. 2010. An asymmetric distributed shared memory model for heterogeneous parallel systems. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10).*

Nilanjan Goswami, Zhongqi Li, Ajit Verma, Ramkumar Shankar, and Tao Li. 2012. Integrating anophotonics in GPU microarchitecture. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12).*

Nilanjan Goswami, Zhongqi Li, Ramkumar Shankar, and Tao Li. 2014. Exploring silicon nanophotonics in throughput architecture. *IEEE Des. Test.* 31, 5, 18–27.

Donald Gross, John F. Shortle, James M. Thompson, and Carl M. Harris. 2008. *Fundamentals of Queueing Theory*. 4th Ed. Wiley-Interscience, New York.

John Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Comput. Archit. News* 34, 4, 1–17.

Jingcao Hu and Radu Marculescu. 2004. Application-specific buffer space allocation for networks-on-chip router design. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'04).*

Andrew Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. 2009. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proceeding of the Design, Automation and Test in Europe Conference (DATE'09).*

David Kanter. 2010. Intel's Sandy Bridge microarchitecture. http://www.realworldtech.com/sandy-bridge/1/ (Last accessed 3/2012).

Khronos Opencl Working Group. 2011. The OpenCL specification, version 1.2. https://www.khronos.org/news/press/khronos-releases-opencl-1.2-specification.

Avinash Karanth Kodi, Ashwini Sarathy, and Ahmed Louri. 2008. iDEAL: Inter-router dual-function energy and area-efficient links for network-on-chip (NoC) architectures. In *Proceedings of the International Symposium on Computer Architecture (ISCA'08).*

Jaekyu Lee, Si Li, Hyesoon Kim, and Sudhakar Yalamanchili. 2012. Design space exploration of on-chip ring interconnection for a CPU-GPU architecture. Tech. rep. GIT-CERCS-12-05, Georgia Institute of Technology.

Paul Lee, Jiayuan Meng, Zhenyu Qi, Mircea Stan, and Kevin Skadron. 2009. Design space exploration for integrated CPU-GPU chips. In *Proceedings of the NVIDIA GPU Technology Conference (GTC'09).*

George Michelogiannakis, Dionisios Pnevmatikatos, and Manolis Katevenis. 2007. Approaching ideal NoC latency with pre-configured routes. In *Proceedings of the 1st International Symposium on Networks-on-Chip (NOCS'07).* IEEE Computer Society, 153–162.

Asit Mishra, Narayanan Vijaykrishnan, and Chita R. Das. 2011a. A case for heterogeneous on-chip interconnects for CMPs. In *Proceeding of the International Symposium on Computer Architecture (ISCA'11).*

Asit Mishra, Aditya Yanamandra, Reetuparna Das, Soumya Eachempati, Ravi Iyer, Narayanan Vijaykrishnan, and Chita Das. 2011b. RAFT: A router architecture with frequency tuning for on-chip networks. *J. Parallel Distrib. Comput.* 71, 5, 625–640.

Naveen Muralimanohar and Rajeev Balasubramonian. 2009. Cacti 6.0: A tool to model large caches. Tech. rep., HP Laboratories.

Nvidia Corp. 2008. NVIDIA CUDA compute unified device architecture. Programming guide 2.0.

Jason Power, Arkaprava Basu, Junli Gu, Sooraj Puthoor, Bradford Beckmann, Mark Hill, Steven Reinhardt, and David Wood. 2013. Heterogeneous system coherence for integrated CPU-GPU systems. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'13).*

Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. 2000. Memory access scheduling. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA'00).* 128–138.

Smith Ryan. 2011. AMD Radeon HD 7970 review: 28nm and graphics core next, together as one. http://www.anandtech.com/show/5261/amd-radeon-hd-7970-review (Last accessed 10/2012).

Bratin Saha, Xiaocheng Zhou, Hu Chen, Ying Gao, Shoumeng Yan, Mohan Rajagopalan, Jesse Fang, Peinan Zhang, Ronny Ronen, and Avi Mendelson. 2009. Programming model for a heterogeneous x86 platform. *ACM SIGPLAN Not.* 44, 6, 431–440.

Semiconductor Industry Association. 2005. International technology roadmap for semiconductors (ITRS). http://www.itrs.net/Common/2005ITRS/Home2005.htm (Last accessed 3/2010).

Gary Silcott and Irmina Blaszczyk. 2013. AMD unveils innovative new APUs and SoCs that give consumers a more exciting and immersive experience. http://ir.amd.com/phoenix.zhtml?c=74093&p=irol-newsArticle&ID=1772053 (Last accessed 1/2014).

Daniel J. Sorin, Mark D. Hill, and David A. Wood. 2011. *A Primer on Memory Consistency and Cache Coherence. Synthesis Lectures on Computer Architecture*. Morgan and Claypool.

Kyle Spafford, Jeremy Meredith, Seyong Lee, Dong Li, Philip Roth, and Jeffrey Vetter. 2012. The tradeoffs of fused memory hierarchies in heterogeneous computing architectures. In *Proceedings of the 9th Conference on Computing Frontiers (CF'12).* 103–112.

Richard Vuduc, Aparna Chandramowlishwaran, Jee Choi, Murat Guney, and Aashay Shringarpure. 2010. On the limits of GPU acceleration. In *Proceedings of the USENIX Conference on Hot Topics in Parallelism (HotPar'10).*

William Winkle. 2012. AMD fusion: How it started, where it's going, and what it means. http://www.tomshardware.com/reviews/fusion-hsa-opencl-history,3262-4.html (Last accessed 1/2014).

Yi Yang, Ping Xiang, Mike Mantor, and Huiyang Zhou. 2012. CPU-assisted GPGPU on fused CPU-GPU architectures. In *Proceeding of the Symposium on High Performance Computer Architecture (HPCA'12).*

Marcelo Yuffe, Ernest Knoll, Moty Mehalel, Joseph Shor, and Tsvika Kurts. 2011. A fully integrated multi-CPU, GPU and memory controller 32nm processor. In *Proceeding of the International Solid-State Circuits Conference (ISSCC'11).*

Received March 2014; revised July 2014; accepted September 2014