Throughput-Effective On-Chip Networks for Manycore Accelerators

Ali Bakhoda
ECE Department
University of British Columbia
Vancouver, Canada
Email: bakhoda@ece.ubc.ca

John Kim
CS Department
KAIST
Daejeon, Korea
Email: jjk12@kaist.edu

Tor M. Aamodt
ECE Department
University of British Columbia
Vancouver, Canada
Email: aamodt@ece.ubc.ca

***Abstract*—As the number of cores and threads in manycore compute accelerators such as Graphics Processing Units (GPU) increases, so does the importance of on-chip interconnection network design. This paper explores throughput-effective network-on-chips (NoC) for future manycore accelerators that employ bulk-synchronous parallel (BSP) programming models such as CUDA and OpenCL. A hardware optimization is "throughput-effective" if it improves parallel application level performance per unit chip area. We evaluate performance of future looking workloads using detailed closed-loop simulations modeling compute nodes, NoC and the DRAM memory system. We start from a mesh design with bisection bandwidth balanced with off-chip demand. Accelerator workloads tend to demand high off-chip memory bandwidth which results in a many-to-few traffic pattern when coupled with expected technology constraints of slow growth in pins-per-chip. Leveraging these observations we reduce NoC area by proposing a "checkerboard" NoC which alternates between conventional *full*-routers and *half*-routers with limited connectivity. Checkerboard employs a new oblivious routing algorithm that maintains a minimum hop-count for architectures that place L2 cache banks at the *half*-router nodes. Next, we show that increasing network injection bandwidth for the large amount of read reply traffic at the nodes connected to DRAM controllers alleviates a significant fraction of the remaining imbalance resulting from the many-to-few traffic pattern. The combined effect of the above optimizations with an improved placement of memory controllers in the mesh and channel slicing improves application throughput per unit area by 25.4%.**

***Keywords*-NoC; Compute accelerator; GPGPU**

I. Introduction

The bulk-synchronous parallel (BSP) programming model [44] is attractive for manycore compute accelerators since it provides relatively simple software scalability as the number of cores increases with Moore's Law. Languages such as CUDA [35], [38], OpenCL [19], and recently proposed programming models for future accelerator architectures [16] embody the BSP model. In this paper, we explore the on-chip network design space for compute accelerators. Our goal is to find NoC designs for future manycore accelerator architectures employing BSP-like programming models that provide the best performance per unit area cost—those that are *throughput-effective*.

Highly multi-threaded applications running on multi-core microprocessors have coherence traffic and data sharing

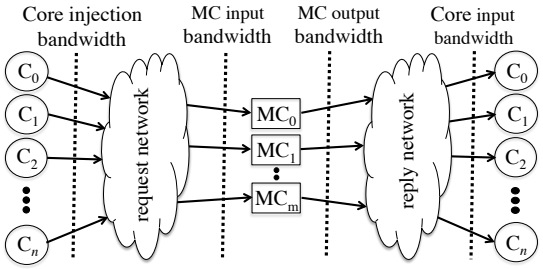


Figure 1. Many-to-Few-to-Many On-Chip Traffic. C nodes are the compute cores and the MC nodes are the memory controllers/memory.

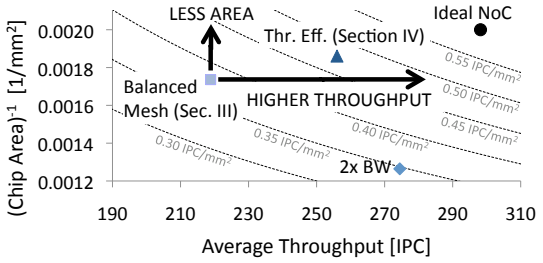


Figure 2. Throughput-Effective Design Space. "Balanced Mesh": bisection bandwidth balanced to off-chip DRAM bandwidth (Section III); "Thr. Eff.": mesh network optimized for many-to-few-to-many traffic (Section IV); "2x BW": mesh with double channel width.

resulting in significant core-to-core communication. In contrast, accelerator applications written in a BSP style [7], [16] tend to organize communication to be local to a group of threads that can execute on hardware units that are located close together and have less communication between threads in different groups even when coherence is supported [16], [17]. Consequently, as the number of pins on a chip is growing only 10% per year [14], the net effect of increases in transistor density on accelerator architectures is an increasingly many-to-few traffic pattern [2] with *many* compute cores sending traffic to a *few* memory controller (MC) nodes. Using detailed closed-loop simulation, we identify how the many-to-few-*to-many* traffic causes another performance bottleneck. A high level diagram of this communication pattern is illustrated in Figure 1.

An implication of this is the following. Starting from a baseline mesh topology with ***bisection*** bandwidth balanced to effective off-chip memory bandwidth (labeled "Balanced

Mesh" in Figure 2) application throughput can be increased while maintaining a regular interconnect topology by naively increasing channel bandwidth. The "2x BW" data point in Figure 2 shows the impact this has on throughput-effectiveness (IPC/mm^2). This figure decomposes throughput per unit chip area as the product of application level throughput (measured in scalar instructions per cycle–IPC) on the x-axis and inverse area (1/mm^2) on the y-axis[1]. Curves in this figure represent constant throughput-effectiveness (IPC/mm^2) and design points closer to the top right near "Ideal NoC" are better where an ideal NoC has infinite bandwidth, zero latency, and zero interconnect area. In contrast, the point "Thr. Eff." results from modifying the baseline NoC to take advantage of the many-to-few-to-many traffic—resulting in a design closer to the throughput-effectiveness of an ideal NoC than alternative designs.

The contributions of this paper are:

- We present a limit study on the impact of on-chip networks across a wide range of compute accelerator applications—identifying the impact of on-chip communication on overall performance. Based on our analysis, we show how conventional network improvements (such as reducing router latency) do not significantly improve overall performance while simply increasing channel width results in significant performance gains but with a large area increase. Consequently, we propose simultaneously considering the effect of the interconnect on parallel application level performance and chip area to find interconnects which are *throughput-effective*.
- We identify that the *many-to-few-to-many* traffic pattern of manycore accelerators (more compute nodes than MCs) creates a traffic imbalance and show how the overall system performance is directly correlated with the injection rate of the *few* MC nodes.
- Based on the above observations, we propose a throughput-effective design that includes a novel *checkerboard* network organization using *half*-routers with limited connectivity to reduce the on-chip network area while having minimal impact on performance. The throughput-effective design also includes a *multi-port* router structure to provide additional terminal bandwidth on the *few* routers connected to the MCs that improves system performance at minimal area cost.

The rest of this paper is organized as follows: Section II summarizes background information, Section III identifies important insights into NoC behavior of manycore accelerator architectures, Section IV describes our proposed NoC, Section V describes experimental results, Section VI summarizes related work and we conclude in Section VII.

[1]Average throughputs are for benchmarks in Table I, described in Section II, using configurations described in Section V. The area estimates are from Section V-F assuming $486mm^2$ is used for compute nodes.

II. Baseline Architecture

In this section we describe our baseline manycore accelerator architecture and on-chip interconnect. Manycore accelerators can be classified along several dimensions: SIMT[2] versus SIMD, degree of multithreading per core, support for caching and coherence, and the granularity at which heterogeneity is introduced. We study a *generic* architecture with some similarities to NVIDIA's Fermi [36] and GeForce GTX 280, but our baseline is *not* meant to be identical to any specific GPU. We believe our conclusions are applicable to other architectures. We employ benchmarks written in CUDA [35], [38], which is similar to the open standard OpenCL [19]. Many of the benchmarks we use (see Table I) are "dwarves" [4] from Rodinia [7].

Our baseline architecture is illustrated in Figures 3, 4, and 5. Figure 3 illustrates the overall chip layout showing the placement of compute nodes and memory controller nodes. In this work, we assume a 2D mesh topology with the memory controllers (MCs) placed on the top and the bottom rows, similar to the topology and layout used in Intel's 80-core design [46] and Tilera TILE64 [47] processors.

Current GPUs often use a crossbar with concentration (to share a single port among several cores) as the number of ports is small. As the number of cores increases, scalability of this approach will be limited. In addition, prior work [5], which included a crossbar comparison, showed that for the workloads we consider performance is relatively insensitive to topology. Thus, we chose a 2D mesh topology since it provides a very regular, simple and scalable network [6].

Each compute node is illustrated in Figure 4. We assume 8-wide SIMD pipelines that execute "warps" (NVIDIA terminology; similar to "wavefronts" in AMD's terminology) consisting of 32 scalar threads executed over four clock cycles. Each compute core maintains a dispatch queue holding up to 32 ready warps (representing up to 1024 scalar threads). In a hardware implementation the large register files would be implemented with banks and bank conflicts might be mitigated using hardware that reorders operand accesses [31] (labeled OC in Figure 4). Memory operations (loads and stores) to global memory (visible to all threads on all cores) go through a memory divergence detection stage (DD) that attempts to "coalesce" memory accesses from different scalar threads within a warp that access a single L1 cache line so that only one request is made per cache block miss. In line with recent manycore architectures such as Sun Niagara [24] we place shared L2 cache banks adjacent to the MCs. The L1 data caches are writeback write-allocate and dirty L1 cache lines are flushed to the L2 under software control (e.g., software managed coherence [16], [36]). Applications also employ a software managed "shared

[2]Single-instruction multiple thread (SIMT): groups of scalar threads execute on a SIMD pipeline using stack-based mechanisms to selectively enable or disable processing elements *without* need for compiler generated predication [8], [30].

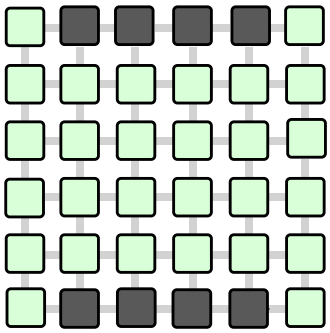


Figure 3. Compute accelerator showing layout of compute node routers and MC node routers in baseline mesh. Shaded routers on top and bottom are connected to MCs.

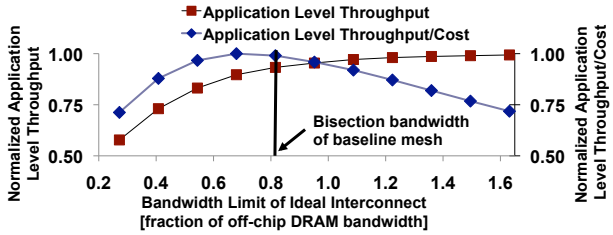


Figure 6. Limit study showing bisection bandwidth of a mesh with 16B channel size can achieve 93% application level throughput (IPC) of a network with infinite bandwidth while maximizing application level throughput per unit estimated area cost.

memory" (*S*). Addresses are low-order interleaved among MCs every 256 bytes [13] to reduce hot-spots [40].

III. Characterization

In this section we analyze characteristics of BSP applications written in CUDA on the baseline architecture described in Section II using *closed-loop* execution driven simulations (see Section V-A for configuration details). We start by identifying the bisection bandwidth required to achieve a *balanced* NoC design when considering the heavy off-chip demands of accelerator workloads. Then, we classify our applications by the intensity of on-chip traffic they generate and their application level throughput sensitivity to interconnect optimizations.

A. Balanced Design

We first size the bisection bandwidth of our network with the aim of finding a balanced design. Bisection bandwidth is a key parameter limiting network throughput. It is defined as the minimum bandwidth over all cuts that partition the network with equal number of nodes in each half [10]. Starting from a system with bisection bandwidth that is "too low" may significantly limit application throughput for memory bound applications (which should instead be limited by off-chip bandwidth) while a system with bisection bandwidth that is "too high" may waste area.

Figure 6 plots two curves: One curve (square markers) is the harmonic mean throughput (IPC) of our benchmarks

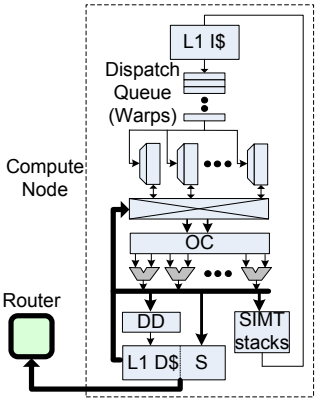


Figure 4. Compute Node

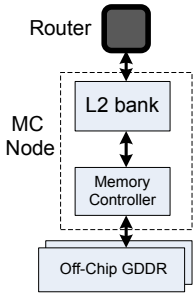


Figure 5. Memory Controller Node

assuming realistic timing models for compute nodes and memory nodes, but a zero latency network with *limited* aggregate bandwidth. This network has zero latency once a flit is accepted, but it limits the number of flits accepted per cycle by enforcing the bandwidth limit specified on the x-axis. Here, bandwidth is total flits transmitted across the network, expressed as a fraction of peak DRAM bandwidth. A packet is accepted provided the bandwidth limit has not been exceeded. Multiple sources can transmit to a destination in one cycle and a source can send multiple flits in one cycle. Application level throughput is normalized to that obtained with an *infinite* bandwidth zero latency network. The slight improvements beyond the point where bisection bandwidth is equal to DRAM bandwidth (1.0 on x-axis) is due to the presence of L2 caches.

The other curve (diamond markers) shows this throughput divided by an estimated chip area. Chip area here includes compute node area and NoC area. NoC area is estimated to be proportional to the square of the channel bandwidth [6]. Although higher network bandwidth continues to increase performance, when normalized to cost, an optimal design from a performance *per* area perspective occurs at around bisection bandwidth ratio of 0.7-0.8. In addition, since performance is generally limited by off-chip bandwidth due to a lack of locality in the workloads and considering the activate/precharge overheads of switching DRAM pages, network bandwidth with 70-80% of peak off-chip DRAM bandwidth also provides a balanced network design. Based on this bisection bandwidth ratio, we determine that this ratio approximately corresponds to a 2D mesh network with 16-byte channels[3].

[3]In Figure 6, the interconnect transfers at most N flits/cycle at interconnect clock frequency ($iclk$). The x-axis in Figure 6 is $x = \frac{N\,[\text{flits/iclk}]\cdot 16\,[\text{B/flit}]\cdot 602\,[\text{MHz (iclk)}]}{1107\,[\text{MHz (mclk)}]\cdot 8\,[\#\text{ MC}]\cdot 16\,[\text{B/mclk}]}$ where $mclk$ is the DRAM clock frequency. At the marked location ($x = 0.816$), N is 12 flits/iclk. Hence, link size is 12 (N) times flit size (16 B) divided by 12 (bisection of a 36-node mesh has 12 links) equals 16B per channel. Clock frequencies are from Table II.

Table I
BENCHMARKS

Name	Abbr.	Name	Abbr.	Name	Abbr.
AES Cryptography [5]	AES	Separable Convolution [37]	CON	MUMmerGPU [5], [7]	MUM
Binomial Option Pricing [37]	BIN	Nearest Neighbor [7]	NNC	LIBOR Monte Carlo [5]	LIB
HotSpot [7]	HSP	Black-Scholes Option Pricing [37]	BLK	Fast Walsh Transform [37]	FWT
Neural Network Digit Recognition [5]	NE	Matrix Multiplication [41]	MM	Scalar Product [37]	SCP
Needleman-Wunsch [7]	NDL	3D Laplace Solver [5]	LPS	Streamcluster [7]	STC
Heart Wall Tracking [7]	HW	Ray Tracing [5]	RAY	Kmeans [7]	KM
Leukocyte [7]	LE	gpuDG [5]	DG	CFD Solver [7]	CFD
64-bin Histogram [37]	HIS	Similarity Score [7]	SS	BFS Graph Traversal [7]	BFS
LU Decomposition [7]	LU	Matrix Transpose [37]	TRA	Parallel Reduction [37]	RD
Scan of Large Arrays [37]	SLA	Speckle Reducing Anisotropic Diffusion [7]	SR		
Back Propagation [7]	BP	Weather Prediction [5]	WP		

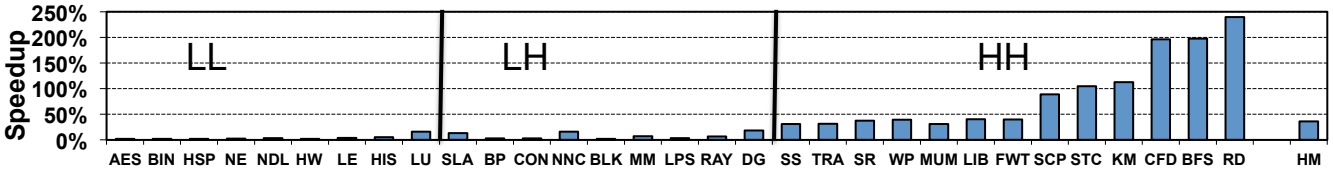


Figure 7. Speedup of a perfect interconnection network over baseline. LL, LH, HH: First character denotes low or high speedup with perfect NoC; second character denotes low or high memory demand.

B. Network Limit Study

Next we perform a limit study to measure the performance benefits of a perfect interconnect (zero latency and *infinite bandwidth*) versus our baseline mesh with 16B channel size. Figure 7 shows the speedup of a perfect network over the mesh with 16B channel bandwidth and a 4-stage router pipeline and a 1-cycle channel delay (5-cycle per hop delay) with the parameters in Table III.

We divide applications into three groups using a two letter classification scheme. The first letter (H or L) denotes high or low (greater or less than 30%) speedup with a perfect network. The second letter (H or L) denotes whether the application sends a heavy or light amount of traffic with a perfect network: accepted traffic, averaged across all nodes, is greater than or less than 1Byte/cycle. All of our applications fall into one of three groups: LL, LH, and HH. Applications in LL place little demand upon the network. Studying the source code of these applications and their detailed simulation statistics we find they have been heavily optimized to group related threads together on a compute node and make good use of the software managed scratchpad memory and/or achieve high L1 hit rates. There is no HL group since applications with low memory access are not likely to get a speedup with a better network. Despite the mesh having sufficient bisection bandwidth (Figure 6) the speedup of a perfect network versus our *realistic* baseline mesh is 36% across all benchmarks, 87% across HH benchmarks and 42% across the Rodinia [7] benchmarks. We explore the reasons for this below.

The LL and HH applications behave as expected: applications that make low use of memory are expected to have low sensitivity to network performance and conversely for those with heavy traffic one would expect to see high speedups. The LH group has a relative high memory usage but its performance does not increase much with a perfect network. Detailed analysis shows these benchmarks achieve close to the peak performance indicating that the interconnect is not the bottleneck with the exception of NNC, which has an insufficient number of threads to fully occupy the fine-grain multithreaded pipeline or saturate the memory system.

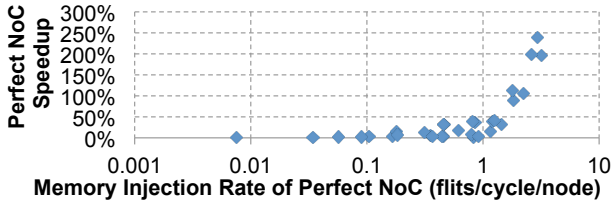


Figure 8. Perfect network speedup versus memory node injection rate

Figure 8 plots perfect network speedup versus average memory controller node injection rate. Speedups are correlated to the memory controller injection rate (or the MC output bandwidth shown in Figure 1) suggesting the presence of a bottleneck on the read response path. We address this bottleneck in Section IV-D.

C. Router Latency and Bisection Bandwidth

In this section we show that aggressive router latency optimizations [26], [28], [33], [39] do not provide significant performance benefits for our workloads. Figure 9 shows that replacing the 4-cycle baseline routers with aggressive 1-cycle routers results in fairly modest speedups ranging from no speedup to at most 7% (harmonic mean speedup is 2.3% for all benchmarks). Figure 10 compares the network latency of these two configurations; y-axis is the network latency reduction of using 1-cycle routers over 4-cycle baseline routers. These figures show that an aggressive router *can* decrease network latency but this improvement in network performance does not necessarily translate into overall performance benefits for these workloads. For example, the

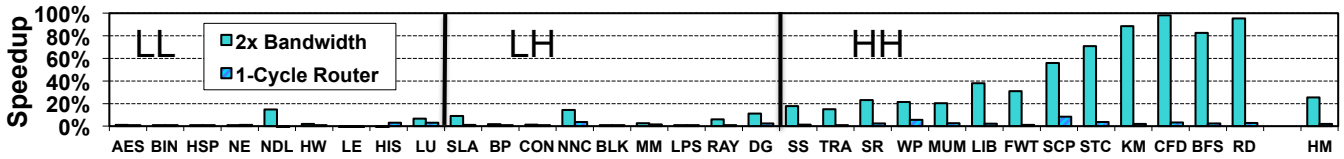


Figure 9. Impact of scaling network bandwidth versus latency. Solid bars: channel size 32 versus 16, Hashed bars: 1-cycle versus 4-cycle router latency.

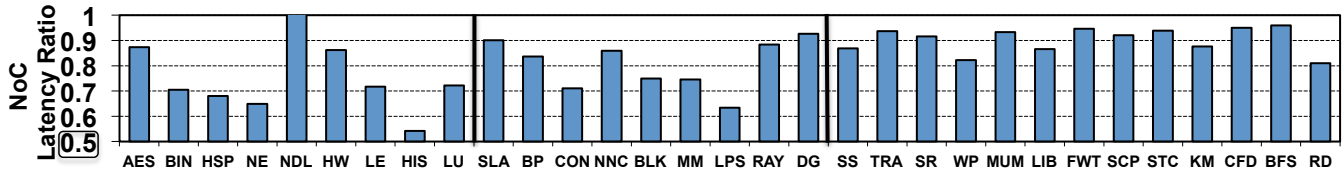


Figure 10. Interconnection latency reduction of using 1-cycle routers over baseline 4-cycle routers

network latency of HIS is reduced by approximately 2× with an aggressive router but this only results in a 3% performance improvement. In contrast, network bandwidth is an important metric as it impacts the overall throughput of the network. By increasing the network channel bandwidth by a factor of 2× (from 16B to 32B), a 27% speedup is achieved over the baseline with 16B channels as shown in Figure 9. However, high-bandwidth NoC designs are very costly in terms of area as we show in Section V-F. Given the baseline mesh was designed to have a bisection bandwidth within 7% of that required to achieve the performance of perfect a network, the data in Figure 9 is again strongly suggestive of an imbalance in the network. Next, we show that one of the reasons is the traffic pattern.

D. Many-to-Few-to-Many Traffic Pattern

The compute accelerator architectures we study presents the network with a *many-to-few-to-many* traffic—with *many* compute nodes communicating with a *few* MCs. As shown earlier in Figure 1, the MC bottleneck is not only caused by the ratio of many cores to few MCs (28/8 in our simulations), but also caused by the difference in packet sizes. As a result, by simulating a closed-loop system with all components modeled, we also identify how the many-to-few-*to-many* traffic pattern causes a bottleneck in addition to the bottleneck caused by the many-to-few pattern. The traffic sent from compute cores to MCs consists of either read requests (small 8-Byte packets) or, less frequently, write requests (large 64-Byte packets) while the traffic from MCs to compute cores only consists of read-replies (large 64-Byte packets). This creates an imbalance in injection rates—on average the injection rate (bytes/cycle) of an MC is 6.9× higher than a compute core. The higher injection rates of memory response data returning from the MCs creates bottlenecks in the reply network that can stall the MCs. This problem is shown in Figure 11 which shows the fraction of the time MCs are stalled (i.e. cannot process requests) because the reply network cannot accept packets from MCs—resulting in MCs being stalled up to 70% of the time for some of the HH benchmarks.

IV. Throughput-Effective Network Design

In this section we leverage the insights from the analysis in Section III to design throughput-effective NoCs for many-core accelerators. We describe the *checkerboard* network organization which uses *half*-routers to reduce network cost while exploiting the many-to-few traffic pattern characteristics. In addition, it also enables a staggered MC placement to avoid creating hotspots. To address the many-to-*few* traffic imbalance, we describe a simple yet effective router microarchitectural extension to the checkerboard network with multi-port routers at the *few* nodes that increases the *terminal* bandwidth of these nodes. We also extend the checkerboard network with channel slicing to create two parallel networks and further reduce cost.

A. Checkerboard Network Organization

Although the many-to-few traffic pattern creates challenges, it also provides opportunities for optimization—for example, there is no all-to-all communication among all nodes in the system. Based on this observation, we propose a *checkerboard* NoC to exploit this traffic pattern and reduce the area of the NoC. Figure 12 shows a 6×6 configuration of the checkerboard network where routers alternate between *full*-routers shown with solid shaded squares and *half*-routers drawn with hatching. A full-router provides full connectivity between all five ports in a 2D mesh while a half-router (shown in detail in Figure 13) limits the connectivity as packets can not change dimensions within the router. The router microarchitecture is similar to a dimension-sliced microarchitecture [18] but in a dimension-sliced router, packets can change dimensions while we limit this capability to further reduce the complexity of the router. While the injection port and the ejection port of a half-router are connected to all ports, the East port only has a connection to the West port and similarly, the North port is connected only to the South port. By taking advantage of half-routers, the router area can be significantly reduced. For example, in a full-router, the crossbar requires a 5×5 crossbar [4] while

[4]Since a packet arriving on a given port can not depart through the same port, the crossbar will actually be a 4×5 crossbar.

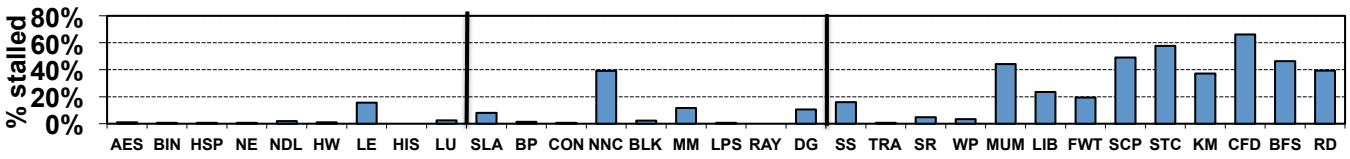
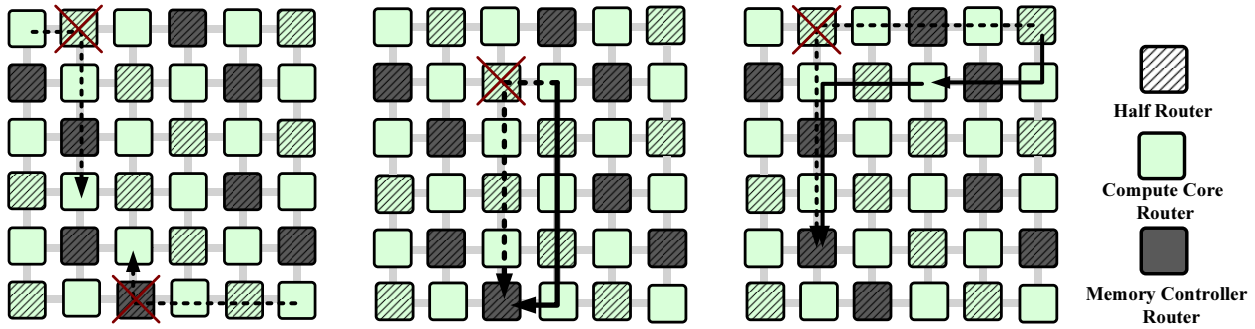


Figure 11. Fraction of time injection port at MCs are blocked preventing data read out of DRAM from returning to compute nodes.



(a) General restrictions (b) Case 1: YX routing (c) Case 2: Checkerboard routing

Figure 12. Checkerboard Mesh On-chip Network routing examples. Dashed lines are examples of XY routes prevented by half-routers (hatched); alternate feasible routes are solid. Dark shaded nodes are MC routers.

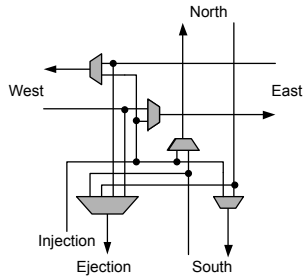


Figure 13. Half-router connectivity

the half-router only requires four 2×1 muxes (two for each dimension) and one 4×1 mux for the ejection port, resulting in approximately 50% reduction in area (detailed analysis shown in Section V-F).

The checkerboard layout does present some limitations in terms of communication (and routing) because of the limited connectivity of the half-routers. Regardless of the routing algorithm (minimal, adaptive, or non-minimal), a packet with a full-router source and a full-router destination that are an odd number of columns or rows away cannot be routed, as illustrated in Figure 12(a), since the packet cannot turn at a half-router. However, by exploiting the many-to-few traffic pattern, the communication between full-routers can be removed by placing the MC nodes at half-routers. Thus, all full-routers represent a compute node and this routing limitation of the checkerboard layout does not become a problem for these manycore accelerator architectures. In addition, as the data in Section III-D suggests, an injection rate imbalance between MCs and compute cores creates hot-spots in the baseline network in which the MCs are placed in neighboring locations on top and bottom of the chip. Thus, the checkerboard network can also exploit a staggered MC placement [2], [5]. Similarly, in architectures with large last level on-chip caches, if the cache banks are restricted to half-routers they can be accessed by all compute nodes. Miss traffic at these banks can reach MC nodes from the cache banks provided both cache banks and MC are also placed at half-router nodes since half routers can route to other half-routers (as described below).

However, if cache banks are placed on the same tiles as the compute cores, the checkerboard organization will restrict cache-to-cache communication as full-routers cannot communicate with all other full-routers. In this case packets would need to be routed to an intermediate half-router (either minimally or nonminimally) and be *ejected* or removed from the network—before being reinjected into the network and being routed to their destination, thus doubling the network load[5]. However, prior work has shown that for accelerator applications written in BSP style languages supporting coherence, cache-to-cache communication is relatively infrequent [16], and hence we expect the impact of this routing on overall performance to be minimal.

B. Checkerboard Routing Algorithm and Flow Control

We assume a baseline dimension-ordered routing (DOR) using XY-routing in the proposed checkerboard network. However, because of the limited connections of the half-routers, XY-routing cannot route a packet for the following two traffic patterns:
Case 1: Routing from a full-router to a half-router which is an odd number of columns away and not in the same row.
Case 2: Routing from a half-router to a half-router which is an even number of columns away and not in the same row.

[5]This is different from randomized routing algorithms such as Valiant [45] routing where packets are routed to an intermediate node but packets do not need to be removed from the network at the intermediate node.

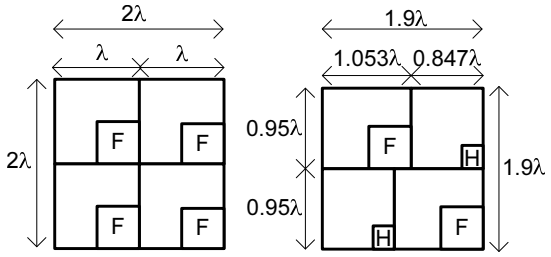


Figure 14. Layout example. Normal (left): F=full-router; Checkerboard (right): H=half-router, F=full-router. Area savings of 10% with two tile layouts assuming (for illustration only) a 75% reduction in half-router and full-routers are 25% of a normal tile.

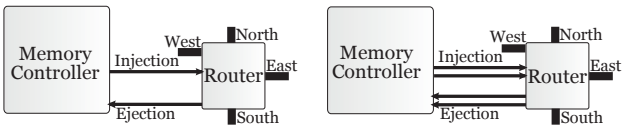
If YX routing is used as the baseline routing algorithm, similar routing restrictions exist as well.

For Case 1, since a packet cannot "turn" or change dimensions at a half-router, YX routing can be used instead of XY routing and thus, the packet turns at a full-router as shown in Figure 12(b). For Case 2, neither XY nor YX routing can be used to route packets because of the limitations of half-routers (Figure 12(c)). As a result, an additional turn is needed to route the packet from the source to its destination by first routing to an intermediate, full-router node and then, routing to the destination. A random, intermediate full-router is selected within the minimum quadrant containing the source and destination that does not share the same row as the source and is not an odd number of columns away from the source. Thus, checkerboard routing (CR), occurs in two phases—in the first phase, YX routing is used to route to the intermediate node and in the second phase, XY routing is used to route minimally to the destination. The CR routing is similar to a 2-phase ROMM routing [34] discussed in Section VI but differs as the random intermediate node is restricted to a full router and each phase needs to be done with a different DOR routing. We implement this routing algorithm with a single extra bit in the header which is set upon injection and tells all the routers on the way that this packets must be YX routed.

To avoid circular dependencies and routing deadlock, two virtual channels are needed in the checkerboard routing, similar to O1Turn routing algorithm [42]. The YX routing is done using one VC while XY routing uses another VC. Additional VCs to avoid protocol deadlock are still needed. Although the checkerboard network requires additional VCs, the reduction in router area is substantial as shown in Section V-F. Reducing overall chip area with this design may require layout modifications like those illustrated in Figure 14. This figure assumes for *illustration and clarity* purposes, a 75% reduction in the area of a half-router and a full-router that is initially 25% of a tile leading to a 10% area reduction in chip area.

C. Double Network—Channel Sliced Network

The area footprint of NoC can be further reduced using channel slicing. For a network with a given bisection bandwidth with each channel having a bandwidth b, our baseline uses a *single* physical network. However, since router area is proportional to $O(b^2)$, it can be reduced by taking advantage of channel slicing [10]: creating a *double* network[6], each with a channel bandwidth of $b/2$. Our channel slicing technique increases the serialization latency of large packets (write requests and read replies) but as we showed earlier these accelerator architectures are not sensitive to a slight increase in latency.



(a) Normal router (b) With 2 injection/ejection ports

Figure 15. Router connections

The traffic in the double network can be load-balanced with a *dedicated* double network where each network is used for a different class of traffic – one network carries request packets and the other network carries reply packets. With a dedicated double network, no extra virtual channel (VC) is needed to avoid protocol deadlock while with a single network, VCs are needed for protocol deadlock avoidance.

D. Multi-port Routers for Memory Controller Nodes

To help reduce the bottleneck at the *few* nodes with many-to-few-to-many traffic pattern (shown in Figure 1), we propose a simple change to the routers attached to the few MC nodes: adding additional injection/ejection ports from/to the MC and creating a *multi-port* router microarchitecture. These additional ports do not increase the network bisection bandwidth or any network channel bandwidth but instead, increase the *terminal* bandwidth by providing more injection/ejection bandwidth from/to the MC nodes. Figure 15(a) shows connection of a conventional router in a 2D mesh network and Figure 15(b) shows the proposed *multi-port* router microarchitecture with additional injection/ejection ports. Selection of the ports at multi-port routers can be done in a simple round-robin fashion.

Note that only the routers connected to the MC nodes change. When adding extra ejection ports, we leverage the fact that an MC is servicing requests from many compute cores; as packets destined to different compute cores get in the MC router, they will start traveling in different directions towards their destination. This technique would not improve performance if the MC had to service a single compute core for a long time since we are not increasing the bandwidth of the links between routers.

V. Experimental Results

In this section we present experimental results for our throughput-effective interconnect optimizations. We start by

[6]Balfour and Dally [6] proposed MeshX2 topology which creates two parallel networks which increases cost. Our approach differs slightly as we *partition* the network – thus, comparing networks with same bisection bandwidth.

Table II
SIMULATION PARAMETERS

Parameter	Value
Number of Compute(Shader) Cores	28
Number of Memory Channels	8
MSHRs / Core	64
Warp Size	32
SIMD Pipeline Width	8
Number of Threads / Core	1024
Number of CTAs / Core	8
Number of Registers / Core	16384
Shared Memory / Core	16KB
Constant Cache Size / Core	8KB
Texture Cache Size / Core	8KB
L1 Cache Size / Core	16KB
L2 Cache Size / MC	128KB
Compute Core Clock	1296 MHz
Interconnect & L2 Clock	602 MHz
Memory Clock	1107 MHz
GDDR3 Memory Timing	t_{CL}=9, t_{RP}=13, t_{RC}=34 t_{RAS}=21, t_{RCD}=12, t_{RRD}=8
DRAM request queue capacity	32
Memory Controller (MC)	out of order (FR-FCFS)
Branch Divergence Method	Immediate Post Dominator [11]
Warp Scheduling Policy	Round Robin among ready warps

Table III
BASELINE INTERCONNECT CONFIGURATION

Topology	Mesh
Routing Mechanism	DOR
Routing Latency (number of router pipeline stages)	4
Channel Latency	1
Flow Control	Virtual Channel based on Wormhole
Virtual Channels	2
Buffers per Virtual Channel	8
Allocator	iSLIP
Input Speedup	1
Channel width (Flit size)	16 bytes

Table IV
ORION 2.0 CONFIGURATION

Technology	65nm
Crossbar type	Matrix
Buffer Type	SRAM
Wire Layer	Intermediate
Wire Spacing	Single

Table V
ABBREVIATIONS

DOR	Dimension Order Routing
CP	Checkerboard Placement
CR	Checkerboard Routing
TB	Baseline Top-Bottom Placement
2P	2 injection port MCs
BW	Bandwidth

describing our simulation setup, then explore the impact of MC placement, the impact of checkerboard router design, the impact of separate physical networks, and finally the impact of multi-port routers at the MC nodes.

A. Methodology

We use a modified version of GPGPU-Sim [5], a detailed cycle level simulator modeling a contemporary GPU running compute accelerator workloads. The modifications we made include adding support for a limited number of MSHRs per core, proper modeling of memory coalescing according to the CUDA manual [38], using Booksim 2.0 [1] instead of Booksim 1.0, and adding support for some undocumented (by NVIDIA) barrier synchronization behavior required by LE and SS benchmarks (barriers synchronize at the level of warps rather than scalar threads in NVIDIA GPUs [48]).

Table II and III show our hardware parameters. Configuration abbreviations are listed in Table V. We modeled half routers with a 3-stage pipeline, though we found the performance impact of one less stage was negligible. While we are interested in future designs, we chose parameters similar to GeForce GTX 280 except for the addition of caches which more closely represent per thread resources on Fermi. We do this to aid in estimating area overheads of compute portions of the overall accelerator. We use ORION 2.0 [15] for network area estimation; Table IV shows the corresponding configuration options. The benchmarks used in simulation are listed in Table I. We simulate all benchmarks to completion to capture distinct phases of the benchmarks.

B. Checkerboard Placement (CP)

Figure 16 shows the performance impact of moving the MC nodes from the top-bottom configuration in Figure 3 to the staggered locations shown in Figure 12, but still maintaining full routers and DOR routing. This placement of the MCs benefits from less contention [2] and by itself results in an average speedup of 13.2% compared to baseline top-bottom placement. We chose this particular placement by picking the best performing placement after simulating several valid checkerboard placements (but did not exhaustively simulate all valid placements).

For applications with low injection rates at the MC nodes (such as LL and LH applications), the MC placement has little or no impact on overall performance since the contention in the return network is not high. Note that WP's loss of performance (6%) is due to global fairness issues that slow down a few of the compute cores. There are studies [29] that tackle the global fairness in NoCs which are orthogonal to the techniques we introduce in this paper.

C. Checkerboard Routing (CR)

Figure 17 shows the relative performance of DOR with 4 VCs (solid bars) and checkerboard routing with 4 VCs (hashed bars) compared to the DOR routing with 2VCs. Simulations show that using checkerboard network, with half of the routers being *half*-routers, results in minimal loss in performance (on average 1.1% reduction), compared to the 2VC DOR network which requires all *full*-routers while significantly reducing the network area. Although a different routing algorithm is required in the checkerboard network, it is still *minimal* routing (minimal hop count between source and destination). Checkerboard network has minimal impact on average network latency as it makes balanced use of the virtual channels in each direction. For example, RD uses the VC dedicated to YX routing for 60.1% of the total packets it injects to the network. Thus, checkerboard network reduces router area with minimal performance loss on average.

D. Double Network—Channel Sliced Network

As described earlier in Section IV, the traffic with a double network is load-balanced with a *dedicated* double network where each network is used for a different class of

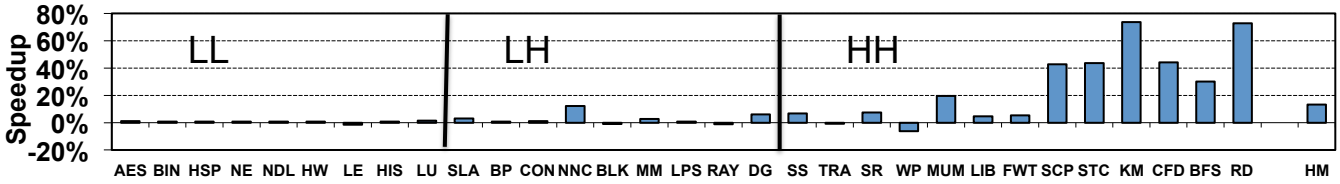


Figure 16. Overall speedup of using checkerboard placement of routers compared to baseline top-bottom placement (both configuration have 2 VCs).

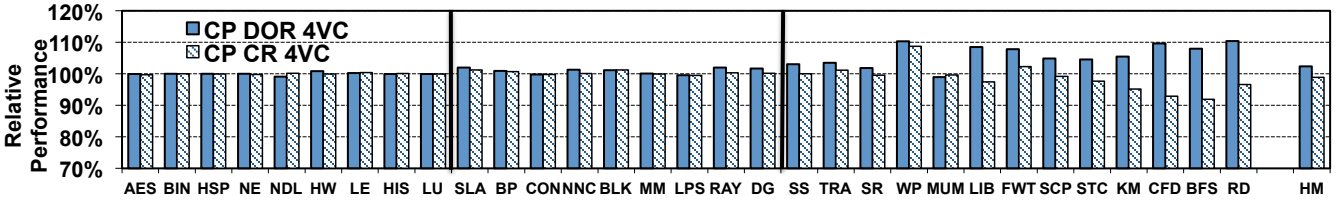


Figure 17. Relative performance (IPC) of DOR with 4 VCs (solid bars) and checkerboard routing with 4 VCs (hashed bars) compared to DOR routing with 2 VCs; all with checkerboard placement(CP). Higher bars mean better performance.

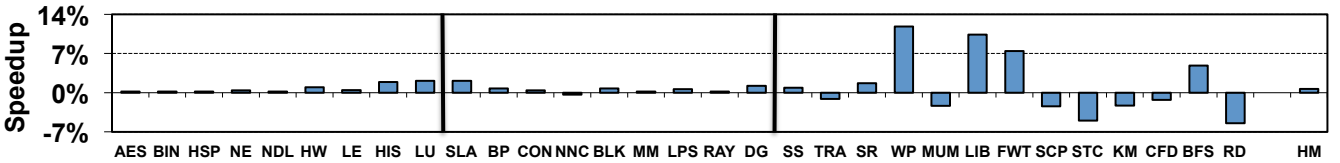


Figure 18. IPC speedup of using two physical networks with channel width 8B (each network has 2VCs) compared to a single network with channel width 16B with 4VCs (both have checkerboard routing and checkerboard placement and 8 buffers per VC).

traffic—one network dedicated to read/write requests and the other network dedicated to replies. Conventionally, channel slicing is beneficial if combined with the reduction of the network diameter [10], [22]; however we utilize channel slicing without reducing network diameter to reduce network area (Section V-F). In addition to the area savings in the router crossbar (taking advantage of quadratic dependency of crossbar area on channel bandwidth) we also save buffer area by keeping the number of VCs constant as we move from single network to double network. The number of VC buffers in the network remains constant but the amount of storage of each VC buffer is reduced to half since the channel width is also halved. Figure 18 shows the speedups of the double over the single network. On average there is no change in performance (around 1% speedup) while providing area savings as we show in Section V-F.

One drawback of channel slicing is increased serialization latency for large packets with narrower channels. This increase in latency only impacts read reply and write request packets since the small read request packets still fit in a single flit. However, as shown earlier in Section III-C, the additional latency has minimal impact on these workloads and are tolerated by the compute cores.

E. Multi-port routers

Figure 19 shows the speedups of increasing terminal bandwidth of MC routers by adding an extra injection port (left bars), an extra ejection port (middle bars) and combination of these changes (right bars) – as described in Section IV and Figure 15(b). It can be seen that the speedups gained by extra injection and ejection ports are orthogonal and add up when combined. The highest speedups are gained by HH benchmarks. The extra injection ports at MC routers reduces the average fraction of execution time the injection ports at MCs are blocked by 38.5% which provides additional performance benefits.

Adding extra ejection ports to MC routers only helps a few benchmarks such as TRA and FWT that are sensitive to the delivery timing of requests to the FR-FCFS input sorting queue in the MC. Their speedup is due to an increase in DRAM row locality for these benchmarks which translates into higher DRAM efficiency[7]—e.g. FWT's DRAM efficiency goes from 57% to 65% with the addition of the extra ejection port. We will not keep the extra ejection port as part of our throughput-effective design since the speedups it provides are limited to a few benchmarks.

Combining the optimizations introduced above (checkerboard placement, checkerboard routing, double network and 2 injection ports at MC routers) results in a 17% speedup versus our baseline introduced in Section II as shown in Figure 20. Compared with 36% speedup of a perfect network, our throughput-effective network achieves roughly half of the performance possible with a perfect network while significantly reducing area.

Figure 21 plots *open-loop* latency versus offered load for the combinations of checkerboard and multiple injection ports evaluated earlier using closed-loop simulation for both uniform many-to-few and hotspot traffic. For hot-spot traffic 20% of requests go to one MC as opposed to of 12.5% $(1/8)$ for uniform random. These open-loop simulations use a single network with two logical networks for request and

[7]Defined as the percentage of time a DRAM chip is transferring data over its data pins divided by the time when pending memory requests exist.

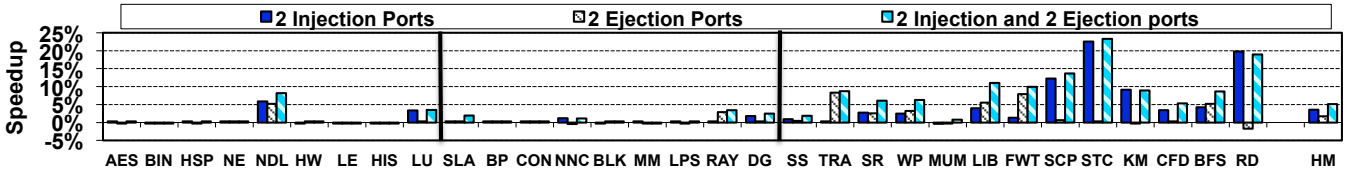


Figure 19. IPC speedup of adding multi-port MC routers versus double network checkerboard.

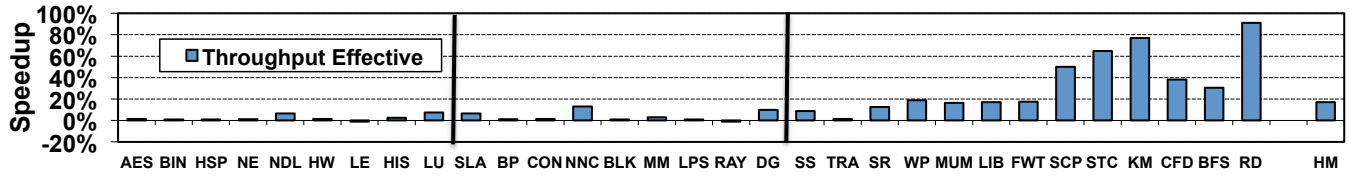
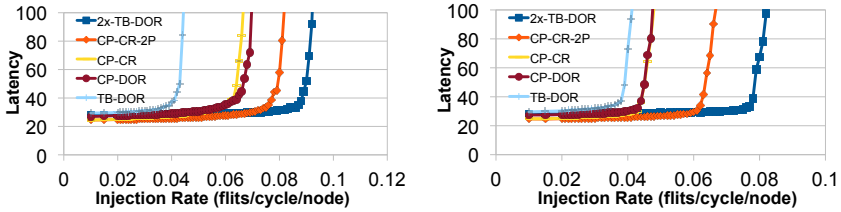


Figure 20. IPC speedup of combining checkerboard placement and routing with double network and two injection port MCs versus baseline top-bottom with DOR.



(a) Uniform Random Many-to-Few-to-Many (b) Hotspot Many-to-Few-to-Many

Figure 21. Latency versus network throughput for different architectures. The *few* nodes (8 MC nodes) inject 4-flit packets while the compute nodes inject 1-flit packets i.e., only read traffic is simulated. The overall network throughput is limited because of the many-to-few-to-many bottleneck.

reply traffic. These figures show that combining checkerboard placement (CP), checkerboard routing (CR) and two injection ports at the MC (2P) improves performance by increasing saturation throughput versus the baseline top-bottom placement (TB). The double bandwidth counterpart of baseline (2x-TB) is also shown for reference. The largest contributors to performance for uniform random traffic are the placement of MCs and increasing injection ports at the MCs (note read response packets are larger than read request packets). For the hot-spot traffic the improvements of MC placement are more moderate while adding the extra injection ports at MCs improves performance significantly by alleviating the bottlenecks created by hot-spot traffic. Although addresses are low-order interleaved among MCs every 256 bytes [13] to reduce hot-spots we have observed that temporary hot-spots happen in closed-loop simulations.

F. Area Analysis

We use ORION 2.0 [15] to estimate area of various router architectures and network topologies. As shown earlier, aggressive investments to reduce router latency do not result in substantial overall performance improvements. Table VI provides the area estimates for the designs we evaluated. We use the GTX280's area, $576mm^2$ in 65nm, as our baseline. Then we estimated the area of "compute" parts by subtracting the total estimated area of our baseline mesh network from the GTX280's area ($486mm^2$). Assuming the compute area does not change, we estimate the total chip area for other network configurations (last column of table). The first row shows the area of baseline mesh with channel width of 16 bytes and the second row a mesh with channel width of 32 bytes. As expected, a quadratic increase in the router area happens by doubling the channel width. The high area overhead of the mesh with channel width 32 bytes, which is 53% of GTX280's area, makes it impractical to build. By exploiting half-routers, which occupy only 56% of the area of a full-router, the checkerboard network results in a significant reduction in total router area of 14.2% (comparing sum of router area numbers which are $59.2mm^2$ in 65nm for checkerboard and $69mm^2$ for baseline router). By further taking advantage of the quadratic dependency, the double network reduces the area further by 37%. Table VI's last row shows the area of the configuration with 2 injection ports at MC nodes; it increases the router area overhead only by 1%. In this design, the eight half-routers connected to MCs have 2 injection ports instead of 1.

Overall, considering both the increase in throughput and reduction in area, we improve throughput-effectiveness (IPC/mm^2) by 25.4% when comparing the checkerboard network with checkerboard placement, 2 injection ports, and double network versus our balanced baseline mesh.

VI. Related Work

A. Accelerator Architectures

Rigel [16] is an accelerator that is fundamentally similar to our architecture but provides a more flexible programming model compared to CUDA and chooses a MIMD model rather than SIMT. The Cell [23] architecture's NoC design is an example of making tradeoffs between network's area and latency. The Cell designers chose a ring over a crossbar

Table VI
AREA ESTIMATIONS (mm^2). OVERHEADS ARE BASED ON GTX280'S AREA. A "/" SEPARATES DIFFERENT ROUTER TYPES FOR CONFIGURATIONS THAT HAVE MORE THAN ONE ROUTER TYPE.

	Area of 1 link	Crossbar Area	Buffer Area	Allocator Area	Area of 1 Router	Link Area Sum	Router Area Sum	% of NoC overhead	Total Chip Area
Baseline	0.175	1.73	0.17	0.004	1.916	21.015	69.00	15.63%	576
2x-BW	0.349	6.95	0.34	0.004	7.305	41.963	263.0	52.95%	790.948
CP-CR	0.175	0.83/ 1.73	0.34/ 0.34	0.004/ 0.015	1.18 / 2.10	21.015	59.20	13.9 %	566.2
Double CP-CR	0.087	0.43/ 0.20	0.087/ 0.087	0.004/ 0.015	0.522/ 0.30	21.015	29.74	8.7 %	536.74
Double CP-CR 2P	0.087	0.43/ 0.20/ 0.28	0.087/ 0.087/ 0.10	0.004/ 0.015/ 0.015	0.522/ 0.30/ 0.38	21.015	30.44	8.93%	537.44

to meet their area and power constraints [25]. The choice of centralized arbiters can limit scalability. UltraSPARC T2 [43] microprocessor is a multithreading, multi-core CPU that uses a crossbar interconnect. GPUs and Cell are related to stream computing [3], [9].

B. Interconnection Networks

Increasing number of cores on a single chip has increased the importance of networks-on-chip (NoC). However, much of the research in NoC have focused on reducing network latency by improving different aspects of NoC such as lower latency router microarchitectures [26], [33], lower-diameter topologies [6], [12], [21], or better flow control [27], [28]. However, as we showed in Section III, reducing latency does not help to improve overall performance in compute accelerator applications but they are more sensitive to bandwidth. Bufferless routing [32] was proposed to reduce network cost by removing buffers but for applications with high traffic, network throughput can be degraded.

On-chip networks for GPUs have been explored by Bakhoda et al. [5] where impact of different network parameters are evaluated. This work builds upon their work, providing more in-depth analysis and proposing a cost-efficient on-chip network architecture for accelerator architectures. Yuan et al. [49] proposed a complexity-effective DRAM access scheduling technique for manycore accelerators that relies on modification to arbitration scheme in request path of NoC. Abts et al. [2] studied alternative MC placements for core-memory traffic; however, they did not show overall performance benefits on applications but focused on latency metrics and synthetic traffic patterns. The MC placement that we use in this work leverages this prior work by staggering the MC placement and shows how overall performance can be significantly improved. Checkerboard routing is similar to ROMM [34]. In 2-phase ROMM, a random node is selected within the minimal quadrant and DOR routing is used to route the packet to a random node in the first phase before routing to the destination in the second phase.

Increasing the radix of the routers in on-chip networks have been proposed [6], [21] to reduce the network diameter and increase network performance, mainly through lower latency. The multi-port approach differs as we only increase radix across a *few* routers to minimize the impact on complexity.

The proposed half-router shares some similarity to the low-cost router microarchitecture [20]. However, unlike the low-cost router microarchitecture which provides full connectivity for XY routing, the routing is restricted in the half-router to further reduce complexity.

VII. CONCLUSION

In this paper, we analyze the impact of communication and on-chip network across a wide range of applications in manycore compute accelerators. We describe how these applications are not sensitive to latency but to bandwidth and how the traffic pattern (mostly many-to-few-to-many) creates a bottleneck in the on-chip network. To improve the performance, we focus on *throughput-effective* on-chip network where we optimize for higher application throughput *per* area. To achieve a throughput-effective on-chip network, we propose a *checkerboard* organization where we exploit *half*-routers to reduce network cost with minimal loss in performance. We further extend the checkerboard network with multi-port routers to address the many-to-few-to-many bottleneck and provide a throughput-effective microarchitectural technique to improve network performance by increasing the terminal bandwidth of the network.

ACKNOWLEDGEMENTS

We thank Wilson W. L. Fung, Andrew Turner, Johnny Kuan, Arun Ramamurthy, Minoo Jalali, and the anonymous reviewers for their valuable feedback on this work. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Booksim interconnection network simulator. http://nocs.stanford.edu/booksim.html.

[2] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti. Achieving predictable performance through better memory controller placement in many-core cmps. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 451–461, 2009.

[3] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das. Evaluating the Imagine Stream Architecture. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 14–25, 2004.

[4] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A view of the parallel computing landscape. *Commun. ACM*, 52(10):56–67, 2009.

[5] 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 Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 163–174, April 2009.

[6] J. D. Balfour and W. J. Dally. Design Tradeoffs for Tiled CMP On-Chip Networks. In *Proc. ACM Conf. on Supercomputing (ICS)*, pages 187–198, 2006.

[7] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proc. IEEE Symp. on Workload Characterization (IISWC)*, pages 44–54, 2009.

[8] B. W. Coon and E. J. Lindholm. US Patent 7,353,369: System and Method for Managing Divergent Threads in a SIMD Architecture, 2008.

[9] W. J. Dally, F. Labonte, A. Das, P. Hanrahan, J.-H. Ahn, J. Gummaraju, M. Erez, N. Jayasena, I. Buck, T. J. Knight, and U. J. Kapasi. Merrimac: Supercomputing with streams. In *ACM/IEEE Conf. on Supercomputing*, page 35, 2003.

[10] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[11] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt. Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow. In *Proc. IEEE/ACM Symp. on Microarchitecture (MICRO)*, 2007.

[12] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Express cube topologies for on-chip interconnects. In *Proc. IEEE Symp. on High-Perf. Computer Architecture (HPCA)*, pages 163–174, 2009.

[13] M. Harris. UNSW CUDA Tutorial Part 4 Optimizing CUDA. http://www.cse.unsw.edu.au/~pls/cuda-workshop09/slides/04_OptimizingCUDA_full.pdf.

[14] Int'l Technology Roadmap for Semiconductors. 2008 Update. http://www.itrs.net/Links/2008ITRS/Home2008.htm.

[15] A. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Proc. IEEE/ACM Conf. on Design Automation and Test in Europe (DATE)*, April 2009.

[16] J. H. Kelm, D. R. Johnson, S. S. Lumetta, M. I. Frank, and S. Patel. A task-centric memory model for scalable accelerator architectures. *IEEE Micro Special Issue: Top Picks 2010*, Jan./Feb. 2010.

[17] J. H. Kelm, D. R. Johnson, W. Touhy, S. S. Lumetta, and S. Patel. Cohesion: A hybrid memory model for accelerator architectures. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, Saint-Malo, France, June 2010.

[18] R. Kessler and J. Schwarzmeier. Cray T3D: A New Dimension for Cray Research. *Compcon Spring '93, Digest of Papers.*, pages 176–182, 22-26 Feb 1993.

[19] Khronos Group. OpenCL - The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl/.

[20] J. Kim. Low-Cost Router Microarchitecture for On-Chip Networks. In *Proc. IEEE/ACM Symp. on Microarchitecture (MICRO)*, pages 255–266, 2009.

[21] J. Kim, J. Balfour, and W. Dally. Flattened Butterfly Topology for On-Chip Networks. In *Proc. IEEE/ACM Symp. on Microarchitecture (MICRO)*, pages 172–182, 2007.

[22] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 420–431, 2005.

[23] M. Kistler, M. Perrone, and F. Petrini. Cell Multiprocessor Communication Network: Built for Speed. *IEEE Micro*, 26:10–23, 2006.

[24] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-Way Multithreaded Sparc Processor. *IEEE Micro*, 25(2):21–29, 2005.

[25] D. Krolak. Cell Broadband Engine EIB bus. http://www.ibm.com/developerworks/power/library/pa-expert9/, Retrieved Sept. 2010.

[26] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. Jha. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *Proc. IEEE Conf. on Computer Design (ICCD)*, October 2007.

[27] A. Kumar, L.-S. Peh, and N. K. Jha. Token flow control. In *Proc. IEEE/ACM Symp. on Microarchitecture (MICRO)*, pages 342–353, Lake Como, Italy, 2008.

[28] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jhay. Express virtual channels: Towards the ideal interconnection fabric. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, San Diego, CA, June 2007.

[29] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 89–100, 2008.

[30] A. Levinthal and T. Porter. Chap - a SIMD graphics processor. In *Proc. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 77–82, 1984.

[31] J. E. Lindholm et al. United States Patent #7,339,592: Simulating Multiported Memories Using Lower Port Count Memories (Assignee NVIDIA Corp.), March 2008.

[32] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 196–207, 2009.

[33] R. D. Mullins, A. West, and S. W. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proc. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 188–197, 2004.

[34] T. Nesson and S. L. Johnsson. ROMM Routing on Mesh and Torus Networks. In *Proc. ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 275–287, 1995.

[35] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable Parallel Programming with CUDA. *ACM Queue*, 6(2):40–53, Mar.-Apr. 2008.

[36] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Fermi, September 2009.

[37] NVIDIA Corporation. NVIDIA CUDA SDK code samples. http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html.

[38] NVIDIA Corporation. *NVIDIA CUDA Programming Guide*, 3.0 edition, 2010.

[39] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. IEEE Symp. on High-Perf. Computer Architecture (HPCA)*, 2001.

[40] G. F. Pfister and V. A. Norton. Hot-Spot Contention and Combining in Multistage Interconnection Networks. *IEEE Trans. on Computers*, c-34(10):943–948, 1985.

[41] S. Ryoo, C. Rodrigues, S. Stone, S. Baghsorkhi, S.-Z. Ueng, J. Stratton, and W. W. Hwu. Program Optimization Space Pruning for a Multithreaded GPU. In *Proc. IEEE/ACM Symp. on Code Generation and Optimization (CGO)*, pages 195–204, April 2008.

[42] 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. IEEE/ACM Symp. on Computer Architecture (ISCA)*, pages 432–443, 2005.

[43] Sun Microsystems, Inc. *OpenSPARC^{TM} T2 Core Microarchitecture Specification*, 2007.

[44] L. G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, 1990.

[45] L. G. Valiant and G. J. Brebner. Universal Schemes for Parallel Communication. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 263–277, 1981.

[46] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, Jan. 2008.

[47] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro*, 27:15–31, 2007.

[48] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos. Demystifying GPU Microarchitecture Through Microbenchmarking. In *Proc. IEEE Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 235–246, 2010.

[49] G. L. Yuan, A. Bakhoda, and T. M. Aamodt. Complexity Effective Memory Access Scheduling for Many-Core Accelerator Architectures. In *Proc. IEEE/ACM Symp. on Microarchitecture (MICRO)*, pages 34–44, Dec. 2009.