# Designing On-Chip Networks for Throughput Accelerators

ALI BAKHODA, University of British Columbia
JOHN KIM, Korea Advanced Institute of Science and Technology
TOR M. AAMODT, University of British Columbia



As the number of cores and threads in throughput accelerators such as Graphics Processing Units (GPU) increases, so does the importance of on-chip interconnection network design. This article explores throughput-effective Network-on-Chips (NoC) for future compute 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 to 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. Next, we show that increasing network terminal bandwidth at the nodes connected to DRAM controllers alleviates a significant fraction of the remaining imbalance resulting from the many-to-few traffic pattern. Furthermore, we propose a "double checkerboard inverted" NoC organization which takes advantage of channel slicing to reduce area while maintaining the performance improvements of the aforementioned techniques. This organization also has a simpler routing mechanism and improves average application throughput per unit area by 24.3%.

Categories and Subject Descriptors: C.1.2 [**Computer Systems Organization**]: Multiprocessors—*Interconnection architectures*

General Terms: Design, Performance

Additional Key Words and Phrases: Bulk-synchronous parallel, throughput accelerator, GPGPU, NoC

**ACM Reference Format:**
Bakhoda, A., Kim, J., and Aamodt, T. M. 2013. Designing on-chip networks for throughput accelerators. ACM Trans. Architec. Code Optim. 10, 3, Article 21 (September 2013), 35 pages.
DOI: http://dx.doi.org/10.1145/2512429

## 1. INTRODUCTION

The Bulk-Synchronous Parallel (BSP) programming model [Valiant 1990] is attractive for throughput accelerators since it provides relatively simple software scalability as the number of cores increases with Moore's Law. Languages such as CUDA [Nickolls

---

A preliminary version of this work appeared in the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) [Bakhoda et al. 2010].
This work was partially supported by the Natural Sciences and Engineering Research Council of Canada. J. Kim was supported in part by MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2013-H0301-13-1011) and in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIP) 2011-0015039.
Authors' addresses: A. Bakhoda (corresponding author), Electrical and Computer Engineering Department, University of British Columbia, Vancouver, BC, Canada; email: bakhoda@ece.ubc.ca; J. Kim, Division of Web Science and Technology and Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea; T. M. Aamodt, Electrical and Computer Engineering Department, University of British Columbia, Vancouver, BC, Canada.

1544-3566/2013/09-ART21 $15.00
DOI: http://dx.doi.org/10.1145/2512429

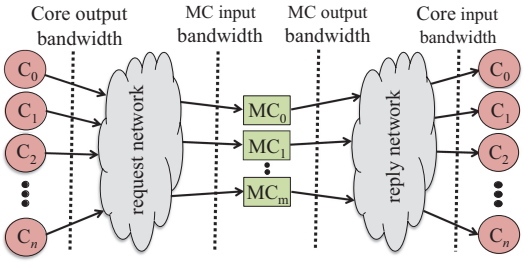



Fig. 1. Many-to-few-to-many on-chip traffic. *C* nodes are the compute cores and the *MC* nodes are the memory controllers/memory.

et al. 2008; NVIDIA 2010], OpenCL [Khronos Group 2010], and recently proposed programming models for future accelerator architectures [Kelm et al. 2010a] embody the BSP model. In this article, we explore the on-chip network design space for compute accelerators. Our goal is to find NoC designs for future compute accelerator architectures employing BSP-like programming models that provide the best performance per unit area cost, that is, those that are *throughput effective*.

Highly multithreaded applications running on multicore microprocessors may have coherence traffic and data sharing resulting in significant core-to-core communication. In contrast, accelerator applications written in a BSP style [Che et al. 2009; Kelm et al. 2010a] 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 [Kelm et al. 2010a, 2010b]. Consequently, as the number of pins on a chip is growing only 10% per year [ITRS 2008], the net effect of increases in transistor density on accelerator architectures is an increasingly many-to-few traffic pattern [Abts et al. 2009] 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. As we will see, MCs become hotspots in the system and have much higher injection rates than the compute cores due to the prevalence of read requests in the system which are small packets but result in large read-reply packets.

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-level throughput can be increased while maintaining a regular NoC 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 more desirable. An ideal NoC has infinite bandwidth, zero latency, and zero NoC area. In contrast, the "Thr. Eff." point 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.

[1]Average throughputs are for benchmarks in Table I, described in Section 2, using configurations described in Section 5. The area estimates are from Section 5.9 assuming $244mm^2$ is used for non-NoC parts of chip.

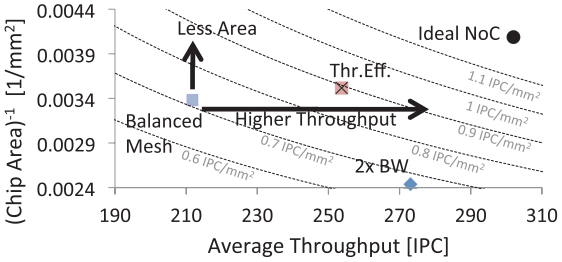



Fig. 2. Throughput-effective design space. "Balanced Mesh": bisection bandwidth balanced to off-chip DRAM bandwidth (Section 3); "Thr. Eff.": mesh network optimized for many-to-few-to-many traffic (Section 4); "2x BW": mesh with double channel width.

The contributions of this article are as follows.

—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 and unacceptable area increase. Consequently, we propose simultaneously considering the effect of the NoC on parallel application-level performance and chip area to find NoCs which are *throughput effective*.
—We identify that the *many-to-few-to-many* traffic pattern of throughput 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 aforesaid 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 multiport router structure to provide additional terminal bandwidth on the few routers connected to the MCs that improves system performance at minimal area cost.
—We propose a "Double Checkerboard Inverted" (DCI) network organization which maintains the benefits of the preceding techniques while having a simpler routing mechanism. This design utilizes a special form of channel slicing where each compute or MC node is connected to both a half and a full router.
—We demonstrate how to extend the DCI network organization to operate with more sophisticated routing mechanisms. By employing two simple injection rules, we show how "Class-based Deterministic Routing" (CDR) can be used in conjunction with DCI without incurring any area overhead.
—We investigate the interactions of various MC placements and our proposed throughput-effective techniques, discuss the implementation trade-offs of peripheral versus scattered placements, and demonstrate how to benefit from our proposed techniques in both cases of peripheral and scattered MC placement.

Figure 3 shows a visual organization of the building blocks for throughput-effective design options we explore in this article. Throughput-effective designs can be achieved by either increasing the application-level performance or decreasing the area footprint of system components: Checkerboard network organization and channel slicing reduce area while multiport MC routers and optimized MC placement provide application-level speedups. We will combine these basic techniques or their enhanced versions to achieve a throughput-effective design.

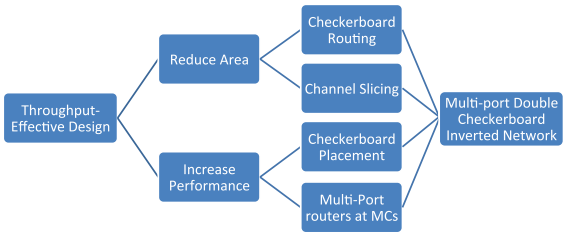



Fig. 3. Overview of building blocks for throughput-effective designs explored in this article.

The rest of this article is organized as follows: Section 2 summarizes background information, Section 3 identifies important insights into NoC behavior of throughput accelerator architectures, Section 4 describes our proposed NoC, Section 5 describes experimental results, Section 6 provides discussions, Section 7 summarizes related work, and we conclude in Section 8.

## 2. BASELINE ARCHITECTURE

In this section we describe our baseline accelerator architecture and on-chip interconnect. Throughput 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 [NVIDIA 2009] and GeForce GTX 280, but our baseline is not meant to be identical to any specific GPU. We believe that our conclusions are applicable to other architectures. We employ benchmarks written in CUDA [Nickolls et al. 2008; NVIDIA 2010], which is similar to the open standard OpenCL [Khronos Group 2010]. Many of the benchmarks we use (see Table I) are "dwarves" [Asanovic et al. 2009] from Rodinia [Che et al. 2009].

Our baseline architecture is illustrated in Figures 4, 5, and 6. Figure 4 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 [Vangal et al. 2008] and Tilera TILE64 [Wentzlaff et al. 2007] processors. We are interested in a general-purpose accelerator architecture, therefore, similar to Intel's Pangaea [Wong et al. 2008], we do not consider fixed function graphics hardware.

### 2.1. Network

Current GPUs often use a crossbar with concentration (to share a single port among several cores). This results in a few number of ports and makes the crossbar a viable option but as the number of cores is bound to increase in the future, the scalability of this approach will be limited. It has been shown that the overheads of high cardinality switches (e.g., a 30 $\times$ 30 crossbar) are unacceptable [Pullini et al. 2007]. A crossbar would be connecting components that are scattered throughout the chip. Routing very long wires to and from a central crossbar is neither easy nor inexpensive. In addition, prior work [Bakhoda et al. 2009], which included a crossbar comparison, showed that for the workloads we consider performance is relatively insensitive to topology. Thus,

[2]Single-instruction multiple thread (SIMT): groups of scalar threads execute on an SIMD pipeline using hardware mechanisms to selectively enable or disable processing elements *without* need for compiler-generated predication [Levinthal and Porter 1984; Coon and Lindholm 2008].

Table I. Benchmark Abbreviations and Their Sources (Rodinia [Che et al. 2009], SDK [CUDA SDK 2009], and GPGPU-Sim [Bakhoda et al. 2009])

| Name | Abbr. | Name | Abbr. |
|---|---|---|---|
| Kmeans (Rodinia) | KM | CFD Solver (Rodinia) | CFD |
| Leukocyte (Rodinia) | LE | Heart Wall Tracking (Rodinia) | HT |
| LU Decomposition (Rodinia) | LU | Similarity Score (Rodinia) | SS |
| Back Propagation (Rodinia) | BP | Streamcluster (Rodinia) | STC |
| HotSpot (Rodinia) | HSP | Needleman-Wunsch (Rodinia) | NDL |
| Speckle Reducing Anisotropic Diffusion (Rodinia) | SR | Neural Network Digit Recognition (GPGPU-Sim) | NE |
| BFS Graph Traversal (Rodinia) | BFS | Nearest Neighbor (Rodinia) | NNC |
| AES Cryptography (GPGPU-Sim) | AES | MUMmerGPU(Rodinia, GPGPU-Sim) | MUM |
| Black-Scholes Option Pricing (SDK) | BLK | LIBOR Monte Carlo (GPGPU-Sim) | LIB |
| Binomial Option Pricing (SDK) | BIN | Matrix Multiplication [Ryoo et al. 2008] | MM |
| Separable Convolution (SDK) | CON | Ray Tracing (GPGPU-Sim) | RAY |
| 3D Laplace Solver (GPGPU-Sim) | LPS | Matrix Transpose (SDK) | TRA |
| Fast Walsh Transform (SDK) | FWT | Parallel Reduction (SDK) | RD |
| gpuDG (GPGPU-Sim) | DG | Scalar Product (SDK) | SCP |
| 64-bin Histogram (SDK) | HIS | Scan of Large Arrays (SDK) | SLA |
| Weather Prediction (GPGPU-Sim) | WP | | |

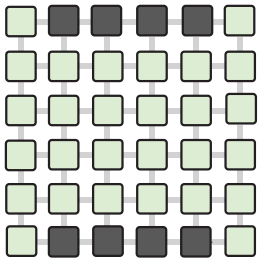


Fig. 4. Compute accelerator showing layout of compute node routers and MC node routers in baseline mesh. Shaded routers on top and bottom are connected to MCs.

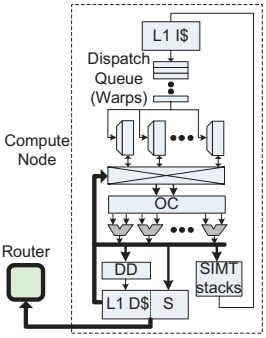



Fig. 5. Compute node.

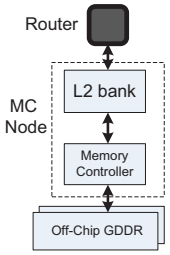



Fig. 6. Memory controller node.

we chose a 2D mesh topology since it provides a very regular, simple, and scalable network [Balfour and Dally 2006]. To avoid protocol deadlock, request and reply traffic have dedicated virtual channels. Comparisons with asymmetric crossbars as well as topologies such as CMesh and flattened butterfly are presented in Sections 6.1 and 6.2, respectively. The techniques provided in this work are orthogonal to concentration as we will discuss in more detail in Sections 6.1 and 6.3.

### 2.2. Compute Nodes and Memory Controllers

Figure 5 illustrates a compute node. 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). 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

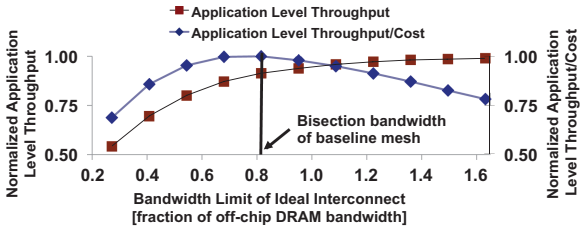



Fig. 7. Limit study showing bisection bandwidth of a mesh with 16B channel size can achieve 91% application-level throughput (IPC) of a network with infinite bandwidth while maximizing application-level throughput per unit estimated area cost.

cache line so that only one request is made per cache block miss. In line with recent multicore architectures such as Sun Niagara [Kongetira et al. 2005] and as suggested by an NVIDIA patent application [Nickolls et al. 2011], we place shared L2 cache banks adjacent to the MCs. Applications also employ a software-managed scratchpad memory or "shared memory" (*S*) in NVIDIA's terminology. Addresses are low-order interleaved among MCs every 256 bytes [Harris 2009] to reduce the likelihood of any single MC becoming a hotspot [Pfister and Norton 1985].

## 3. CHARACTERIZATION

In this section we analyze characteristics of BSP applications written in CUDA on the baseline architecture described in Section 2 using *closed-loop* execution-driven simulations (see Section 5.1 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 NoC optimizations.

### 3.1. 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 [Dally and Towles 2004]. Starting from an on-chip network 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 an on-chip network with bisection bandwidth that is "too high" may waste area.

Figure 7 plots two curves: One curve (square markers) is the harmonic mean throughput (IPC) of our benchmarks 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

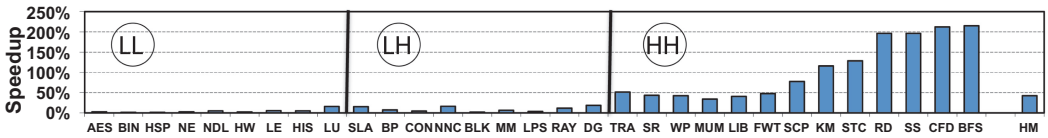



Fig. 8. Speedup of an ideal NoC over baseline. LL, LH, HH: First character denotes low or high speedup with ideal NoC; second character denotes low or high memory demand. HM is harmonic mean of all benchmarks.

estimated to be proportional to the square of the channel bandwidth [Balfour and Dally 2006]. Although higher network bandwidth continues to increase the 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, a network bandwidth with 70–80% of the 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]

Note, the resulting balanced channel bandwidth depends on the ratio of NoC area to the rest of the chip. If the ratio of NoC goes down the maximum point of throughput/cost curve (diamond markers) moves to the right and the curve becomes more flat. For example, if we assume that NoC is 3× smaller than our estimated area, the maximum will occur around the 1.2 mark on $x$-axis which corresponds to a network with 24-byte channels. While starting from a different baseline bandwidth would change the raw numbers in the rest of the article, it would not affect the trends.

### 3.2. Network Limit Study

Next we perform a limit study to measure the performance benefits of an ideal NoC (zero latency and infinite bandwidth) versus our baseline mesh with 16B channel size. This gives us an upper bound for application-level throughput improvements that are possible by optimizing NoC. Figure 8 shows the speedup of an ideal network over the mesh with 16B channel bandwidth, a 4-stage router pipeline, and a 1-cycle channel delay (5-cycle per hop delay) with the parameters shown in Table IV.

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 an ideal network. The second letter (H or L) denotes whether the application sends a heavy or light amount of traffic with an ideal network: accepted traffic, averaged across all nodes, is greater than or less than 1 Byte/cycle. All of our applications fall into one of these three groups: LL, LH, and HH. 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 7) the average speedup of an ideal network versus our realistic baseline mesh is 42.3% across all benchmarks, 102.7% across HH benchmarks, and 44.6% across the Rodinia [Che et al. 2009] benchmarks. We explore the reasons for this next.

Applications in LL place little demand on the network as a result of low utilization of the off-chip DRAM. Studying the source code of these applications and their detailed simulation statistics we find several reasons for this: some benchmarks have

[3]In Figure 7, the network transfers at most $N$ flits/cycle at interconnect clock frequency ($iclk$). The $x$-axis in Figure 7 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 (16B) divided by 12 (bisection of a 36-node mesh has 12 links) equals 16B per channel. Clock frequencies are from Table II.

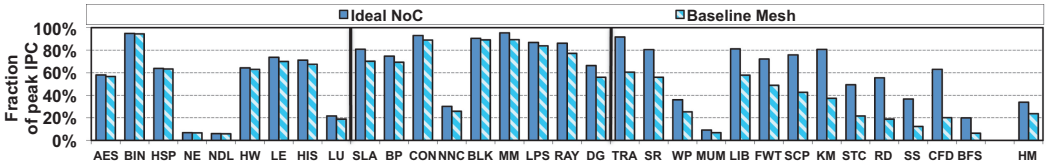



Fig. 9. IPC of ideal NoC and baseline NoC shown as a fraction of the peak theoretical IPC of the chip.

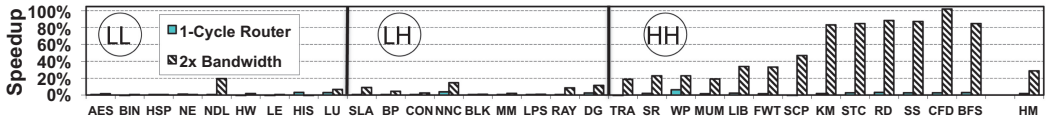



Fig. 10. Impact of scaling network bandwidth versus latency. Solid bars: 1-cycle versus 4-cycle router latency, hashed bars: channel size 32 versus 16.

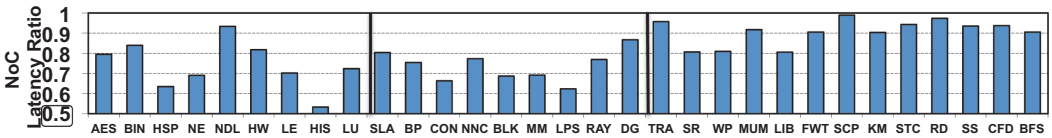



Fig. 11. NoC latency reduction of using 1-cycle routers over baseline 4-cycle routers.

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. On the other hand, another set of benchmarks like NE and LU spend the majority of their runtime running a few threads on each core which cannot fully occupy the multithreaded pipeline. 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 moderate memory usage but its performance does not increase much with an ideal network. Figure 9 shows the IPC of the ideal NoC and baseline mesh configuration as a fraction of the peak theoretical IPC of the chip. Peak theoretical IPC is reached when all the SIMD lanes of the chip are executing one instruction per cycle. All but one of LH benchmarks achieve close to peak performance indicating that the NoC is not the bottleneck. The exception, NNC, has an insufficient number of threads to fully occupy the fine-grain multithreaded pipeline or to saturate the memory system.

### 3.3. Router Latency and Bisection Bandwidth

In this section we show that aggressive router latency optimizations [Peh and Dally 2001; Mullins et al. 2004; Kumar et al. 2007a, 2007b] do not provide significant performance benefits for our workloads. Figure 10 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 6% (harmonic mean speedup is 1.8% for all benchmarks). Figure 11 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 is not enough to translate into noticeable overall performance benefits for these workloads.[4]

[4]Based on our measurements the average hop count is 5.2 in this configuration and average serialization latency is 2.5 cycles. Reducing the router delay from 4 to 1 cycles should result in a 55% NoC latency

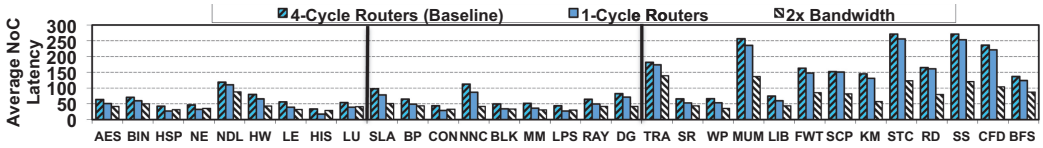



Fig. 12. Impact of scaling network bandwidth and router latency on overall raw NoC latency.

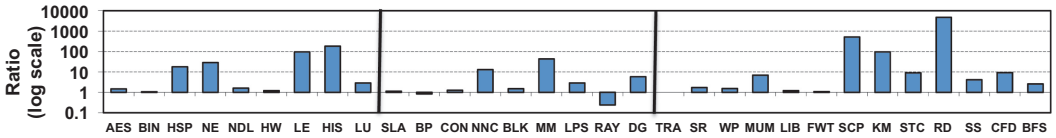



Fig. 13. Ratio of the number of read-request packets to write-request packets sent from compute cores in logarithmic scale.

Contention in the network prevents the aggressive routers from achieving a latency reduction comparable to a zero-load network. Even LL benchmarks that have a low average NoC usage suffer contention because they generate bursty traffic. One example of this behavior is loading the scratchpad memory from global memory in the beginning of a kernel and then leaving the NoC unused for the majority of kernel runtime.

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 28.6% speedup is achieved over the baseline with 16B channels as shown in Figure 10. However, high-bandwidth NoC designs are very costly in terms of area as we show in Section 5.9.

To shed more light on this matter, Figure 12 compares the average NoC latencies of 4-cycle router baseline, 1-cycle router, and 2× bandwidth routers. It can be seen that the latency saved by the 1-cycle routers is only a small fraction of the overall NoC latency, especially for the HH benchmarks. On the other hand, doubling the bandwidth has a large and meaningful impact on the latency (e.g., the NoC latency goes down 55% for STC). The data in Figure 12 is strongly suggestive of an imbalance in the network resulting in considerable contention. Next, we show that the traffic pattern is one of the reasons for this imbalance.

### 3.4. Many-to-Few-to-Many Traffic Pattern

The compute accelerator architectures we study present 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. To better demonstrate the differences in the number of read and write requests, Figure 13 shows the ratio of the number of read-request packets to the number of write-request packets that are sent from compute cores for all the benchmarks.

reduction, that is, 0.45 in Figure 11. Note that the link latency is always 1 and serialization latency is not changing so the minimum possible latency reduction ratio is 0.4.

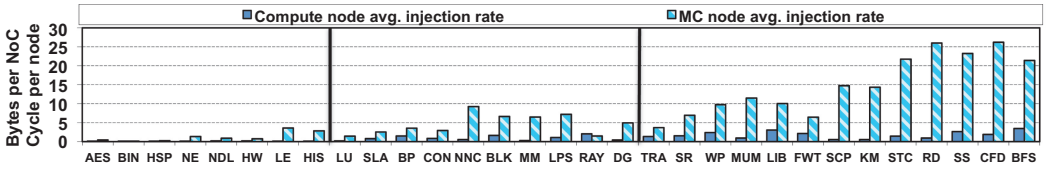



Fig. 14. Average injection rates of compute nodes compared to MC nodes in term of bytes per network cycle per node. NoC is ideal (infinite BW and zero latency).

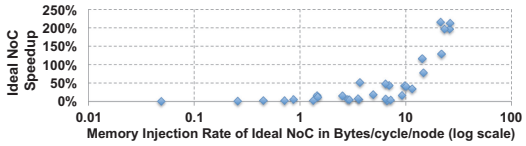



Fig. 15. Ideal NoC speedup versus memory node injection rate.

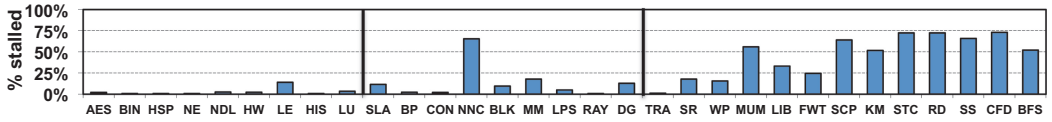



Fig. 16. Fraction of time injection port at MCs are blocked preventing data read out of DRAM from returning to compute nodes.

Figure 14 compares the average injection rates of compute nodes and MC nodes for all the benchmarks. For this figure NoC is assumed to be ideal to better demonstrate the traffic demand that the NoC is expected to handle. This figure demonstrates the bottleneck caused by the few-to-many component of the many-to-few-*to-many* traffic pattern, especially for the HH benchmarks.

Figure 15 plots ideal 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) with a correlation coefficient of 0.926. This suggests the presence of a bottleneck on the read response path.

The higher injection rates of memory response data returning from the MCs create bottlenecks in the reply network that can stall the MCs. This issue is depicted in Figure 16 which shows the fraction of the time MCs are stalled (i.e., cannot process requests) because the network cannot accept packets from MCs, resulting in MCs being stalled up to 73% of the time for some of the HH benchmarks. We address this issue in Section 4.3.

## 4. THROUGHPUT-EFFECTIVE NETWORK DESIGN

In this section we leverage the insights from the analysis in Section 3 to design throughput-effective NoCs for compute accelerators. As shown in Figure 3, throughput-effective designs can be achieved by either increasing the application-level performance or decreasing the area footprint of system components. We introduce techniques that utilize both of these strategies. 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 multiport 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

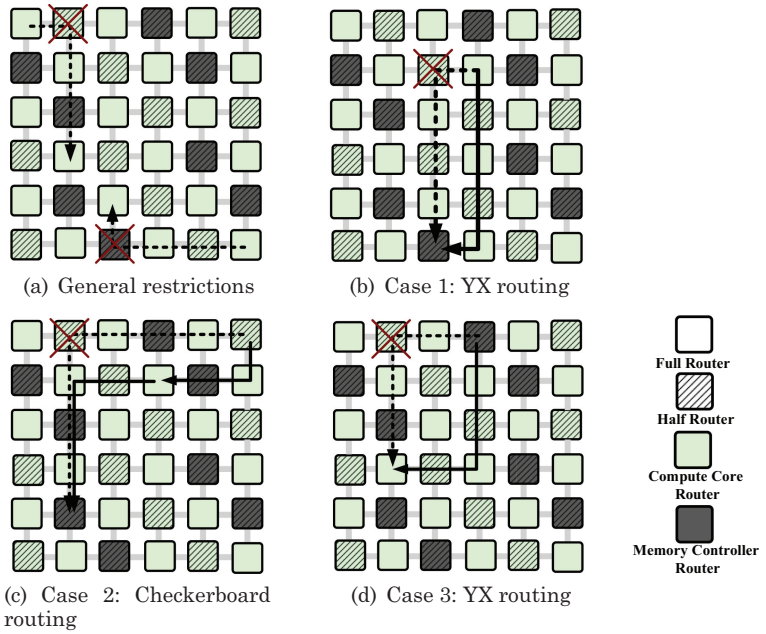



Fig. 17. Checkerboard mesh on-chip network routing examples. Dashed lines are examples of XY routes prevented by half routers (hatched); alternate feasible routes are solid. Dark shaded nodes are MC routers.

networks and further reduce area. Finally we demonstrate how the benefits of these techniques can be realized in a *double checkerboard inverted* network organization.

### 4.1. 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 17 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 18) limits the connectivity as packets cannot change dimensions within the router. The router microarchitecture is similar to a dimension-sliced microarchitecture [Kessler and Schwarzmeier 1993] 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[5] while 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 40% reduction in area (detailed analysis in Section 5.9).

The checkerboard layout does present some limitations in terms of communication (and routing) because of the limited connectivity of the half routers. Regardless of

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

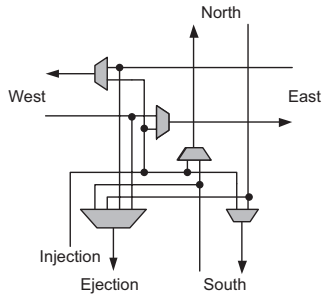



Fig. 18. Half-router connectivity.

the routing algorithm (minimal, adaptive, or nonminimal), 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 17(a), since the packet cannot turn at a half router. However, by exploiting the many-to-few traffic pattern, the communication between fullrouters 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 throughput accelerator architectures. In addition, as the data in Section 3.4 suggests, an injection rate imbalance between MCs and compute cores creates hotspots 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 [Bakhoda et al. 2009; Abts et al. 2009]. Similarly, in architectures with large last-level on-chip caches, if the cache banks are restricted to nodes with 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 MCs are also placed at half-router nodes, since half routers can always route to other half routers (as described shortly).

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[6]. However, prior work has shown that for accelerator applications written in BSP-style languages supporting coherence, cache-to-cache communication is relatively infrequent [Kelm et al. 2010a], and hence we expect the impact of this routing on overall performance to be minimal. In addition, in Section 4.5 we will illustrate a novel extension of checkerboard routing that completely eliminates this limitation.

### 4.2. 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 three traffic patterns.

*Case* 1. It won't route a packet in routing from a full router to a half router which is an odd number of columns away and not in the same row.

[6]This is different from randomized routing algorithms such as Valiant routing [Valiant and Brebner 1981] where packets are routed to an intermediate node but packets do not need to be removed from the network at the intermediate node.

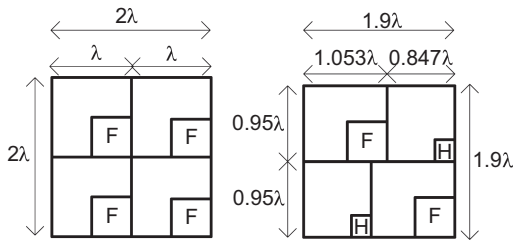



Fig. 19. Layout example. Normal (left): F=full router; Checkerboard (right): H=half router, F=full router. Area savings of 10% with two tile layouts assuming (for illustration only) a 75% reduction in half-router area and full routers occupy 25% of a normal tile.

*Case* 2. It won't route a packet in routing from a half router to a half router which is an even number of columns away and not in the same row.

*Case* 3. It won't route a packet in routing from a half router to a full router which is an even number of columns away and not in the same row.

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 17(b). For Case 2, neither XY nor YX routing can be used to route packets because of the limitations of half routers (Figure 17(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 [Nesson and Johnsson 1995] discussed in Section 7 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.

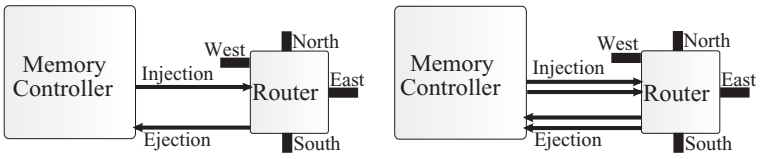
Case 3 can be handled either like Case 1 or like Case 2. In Section 5 we will show that handling Case 3 like Case 1 using YX routing yields higher average application-level throughput. Figure 17(d) shows an example of Case 3 being handled like Case 1.

To avoid circular dependencies and routing deadlock, two virtual channels are needed in the checkerboard routing, similar to O1Turn routing algorithm [Seo et al. 2005]. 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 5.9.

Reducing overall chip area with this design may require layout modifications like those illustrated in Figure 19. 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.

### 4.3. Multiport 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 multiport 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




(a) normal router (b) with 2 injection/ejection ports

Fig. 20. Router connections assuming a single network.

MC nodes. Figure 20(a) shows connections of a conventional router in a 2D mesh network and Figure 20(b) shows the proposed multiport router microarchitecture with additional injection/ejection ports. Selection of the ports at multiport routers can be done in a simple round-robin fashion. The important point is increasing the terminal bandwidth, and multiport routers are one simple way of achieving this effect.

Note that only the routers connected to the MC nodes change. When adding extra injection 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 destinations. 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.

*4.3.1. Smart Injection Port Selection Policy.* As mentioned before, selection of injection ports at multiport routers can be done in a simple round-robin fashion. Round-robin ensures fair usage of port resources and provides reasonable performance as we will show in Section 5.4. Nevertheless it is possible to optimize the overall throughput of the system by designing more intelligent injection mechanisms when there is a choice. The underlying mechanism that improves the performance for multiport injection is that as packets reach the router they start bidding for different outgoing ports and therefore get through the router at the same time. We designed a smart port selection mechanism that tries to boost this effect and therefore increase the number of packets that start traveling in the network. The smart mechanism can be summarized in the following steps.

*Step 1.* Start with a random injection port.
*Step 2.* If port under consideration is empty then inject to it otherwise use step 3.
*Step 3.* If the current packet is going to bid for the same output direction as the last packet injected to this port, then inject to this port.
*Step 4.* Otherwise, try the next port with criteria in steps 2 and 3, if there is no more ports left to test, pick the last port under consideration.

The basic idea is to keep the packets that are going to the same output direction together in the same injection port so that when a packet to a different direction arrives it goes to another port. Therefore, there is a higher chance that the ports are sending packets to different directions in parallel. The aforesaid mechanism is applicable to any number of ports. Implementation for our 2-port setup would require a 2-bit history for each injection port to remember the output direction of last injected packet. Depending on timing requirements of hardware implementation the preceding steps can be done in parallel for all the ports with simple logic.

We also considered other policies such as random port selection or selecting the port with maximum empty space. Our default round-robin policy and the smart policy introduced earlier performed better than the other policies we tried. Another policy that performed particularly poorly was similar to the smart method mentioned before except for step 3 where it would inject to the port under consideration if the current

packet was going to bid for a different output direction as the last packet injected to that port. This is essentially the opposite of the smart policy and would distribute packets going to the same output into different ports.

### 4.4. 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 [Dally and Towles 2004]: creating a double network[7], 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.

The traffic in the double network can be distributed with a dedicated double network where each network is used for a different class of traffic where one network carries request packets and the other network carries reply packets, or with a combined double network where all traffic classes can be sent across either network. With a dedicated double network, no extra Virtual Channel (VC) is needed to avoid protocol deadlock while with either a combined double network or a single network, VCs are needed for protocol deadlock avoidance. On the other hand, a *dedicated* double network essentially cuts the terminal bandwidth of all the nodes to half. A *combined* double network maintains the terminal bandwidth of all the the nodes and can better load-balance the network traffic since a packet can utilize either network. It will also increase the area footprint of routers by doubling the number of VCs required. A combined double network with checkerboard routing requires four VCs in each network while a dedicated double network would require only two VCs per network. Later in Section 4.5 we will demonstrate how we can keep the number of VCs the same as a dedicated double network while load-balancing the traffic like a combined double network. Note that channel slicing is orthogonal to checkerboard routing.

### 4.5. Double Checkerboard Inverted Network Organization (DCI)

The double network organization (Section 4.4) can reduce the NoC area compared with a single network and enables new optimization opportunities to remove the limitations of checkerboard networks. In this section we introduce a double network organization based on the checkerboard network that we refer to as *Double Checkerboard Inverted* (DCI).[8]

The double checkerboard network can be created by two checkerboard subnetworks. However, the DCI differs from the double checkerboard as we exploit an *inverted checkerboard* where the location of the full and half routers is inverted, compared with a normal checkerboard network. Thus, one of the subnetworks in DCI is a checkerboard network while the other subnetwork is an inverted checkerboard network—resulting in all nodes (both compute or MC nodes) to be connected to both a full router and a half router as shown in Figure 21. This form of double network organization removes all the limitations of a checkerboard design as it allows all the nodes to communicate directly with all other nodes. The routing algorithm is also simplified as DOR routing can be used without having to switch routing at an intermediate node (Section 4.2).

[7]Balfour and Dally [2006] proposed MeshX2 topology which doubles the networks and thus, doubles the bisection bandwidth and increases cost. Our approach differs slightly as we *partition* the network, thus comparing networks with same bisection bandwidth.

[8]DCI does not necessarily fit into the description of dedicated or combined terminology that we used earlier in Section 4.4. It has similarities and differences with both of these networks that we will explain in this section.

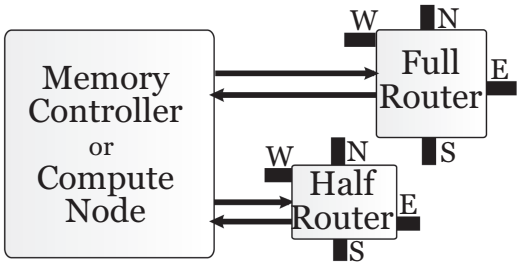



Fig. 21. The connections between a node and routers are shown for one tile in the DCI network organization. Sizes are not to scale.

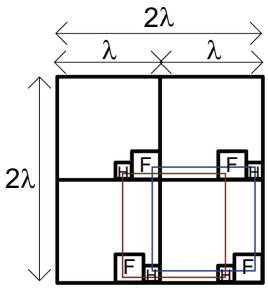
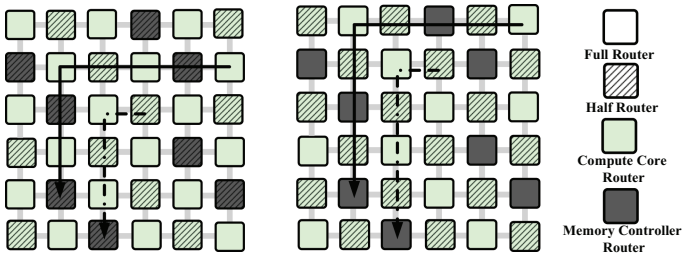



Fig. 22. Layout example showing four tiles for DCI. H=half router, F=full router.




(a) Subnetwork 0 (b) Subnetwork 1

Fig. 23. DCI routing examples. The two physical subnetworks are shown separately for clarity. Solid lines show examples of selecting full routers for injection while dashed lines show examples of selecting half routers. Dark shaded nodes are MC routers.

*Subnetwork selection policy*. To leverage a simple XY DOR routing, the following injection rules must be followed.

*Rule* (1). If the destination node is an *even* number of columns away from the source node then inject the packet to the *full* router attached to the source node.

*Rule* (2). If the destination node is an *odd* number of columns away from the source node then inject the packet to the *half* router attached to the source node.

Following this injection policy, all instances of special Cases 1–3 of checkerboard routing, mentioned in Section 4.2, are removed and there is no need for any packet to turn at a half router. Figure 23 shows some examples of DCI routing. Solid lines in Figure 23 are examples of selecting full routers for injection while dashed lines show examples of selecting half routers for injection.

Each subnetwork in DCI requires a minimum of two VCs for protocol deadlock avoidance—one VC for request packets and one VC for reply packets. DCI utilizes both subnetworks for both memory request and reply packets similar to a combined double network while it can keep the number of VCs the same as a dedicated double checkerboard network (combined and dedicated networks were described in Section 4.4).

As Figure 21 shows DCI provides two injection and ejection ports at each node while the routers themselves only have one injection and one ejection port. Since the subnetworks are formed using channel slicing the terminal bandwidth of DCI is the same as a single network and a combined double network. Achieving a similar effect with a dedicated double checkerboard network would require multiport routers.

Another benefit of DCI is that it allows for a more regular tile design since all the nodes now have both a full and a half router as demonstrated in Figure 22.

### 4.6. Augmenting DCI with Advanced Routing Techniques

A further advantage of the DCI design is that it can be extended to work with routing techniques that are more sophisticated than DOR. For example, O1turn routing [Seo et al. 2005] can be used with DCI by adding an extra set of virtual channels. DCI can also be used with "Class-based Deterministic Routing" (CDR) [Abts et al. 2009]. In CDR routing *request* packets are routed using XY routing and *reply* packets are routed using YX routing. As observed by Abts et al. [2009] CDR routing is useful in the cases with a peripheral row-based MC placement like our baseline in Figure 4. Utilizing YX routing for the reply packets reduces their contention as they leave the MCs when MCs are placed in neighboring locations in a row.

We now describe an extension to the DCI subnetwork selection policy that enables CDR *without* requiring any extra VCs. To use DCI with CDR the compute cores that send request packets and use XY routing employ the injection rules (1) and (2) of Section 4.5 and MC nodes that send reply packets and use YX routing employ the rules (1′) and (2′) described next.

*Rule* (1′). If the destination node is an even number of rows away from the source node then inject the packet to the *full* router attached to the source node.

*Rule* (2′). If the destination node is an odd number of rows away from the source node then inject the packet to the *half* router attached to the source node.

Using this technique, no packet will need to turn at any half router and no change to the CDR technique is necessary once a packet is injected into the network. Note that each subnetwork requires only one VC dedicated to XY routing for the request packets and one VC dedicated to YX routing for the reply packets. Routing deadlock does not happen because each packet only turns once, and protocol deadlock does not happen since request and reply packets have dedicated VCs.

*4.6.1. DCI Enhanced (DCIE)—Increasing Network Utilization for DCI.* If both the source and destination of a packet are on the same row/column then the packet does not need to turn. Such a packet can use either of the subnetworks in DCI as it will never turn and therefore it does not have to comply with the subnetwork selection rules of DCI. We can take advantage of this observation by choosing the injection subnetwork in a manner that improves overall network utilization. When the nonturning packets are being injected, we select the subnetwork that has been least utilized recently. To keep the mechanism simple we limit our selection mechanism to local subnetwork selection history. There are numerous implementation possibilities; we chose to employ an up-down counter that counts up when a packet is injected to subnetwork 1 and counts down when a packet is injected to subnetwork 0. When the injection mechanism faces a nonturning packet it decides where to inject it based on the counter value, for example, if counter value is positive the packet is injected to subnetwork 0. This is just a simple way of approximating the load of each subnetwork, where the goal is to take advantage of nonturning packets to increase the utilization of the DCI network. We will refer to this technique as DCI Enhanced or DCIE.

## 5. EXPERIMENTAL RESULTS

In this section we present experimental results for our throughput-effective NoC optimizations. We start by describing our simulation setup, then explore the impact of checker MC placement, the impact of checkerboard routing, the impact of multiport routers at the MC nodes, and the impact of channel slicing. Next we evaluate the impact of the DCI network and then show the effects of various MC placements on the aforementioned techniques accompanied by a discussion of implementation trade-offs

Table II. Simulation Parameters

| Parameter | Value |
|---|---|
| # of Compute Cores | 28 |
| # 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 |
| NoC & L2 Clock | 602 MHz |
| Memory Clock | 1107 MHz |
| GDDR3 Memory Timing | $t_{CL} = 9, t_{RP} = 13, t_{RC} = 34$ $t_{RAS} = 21, t_{RCD} = 12, t_{RRD} = 8$ |
| Peak DRAM Bandwidth | 141.7 (GB/sec) |
| DRAM request queue size | 32 |
| Memory Scheduling Policy | out of order (FR-FCFS) [Rixner et al. 2000] |
| Branch Divergence Method | Immediate Post Dominator [Fung et al. 2007] |
| Warp Scheduling Policy | Round Robin among ready warps |

Table III. Area Estimation Configuration

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

Table IV. Baseline NoC Configuration

| | |
|---|---|
| Topology | 2D Mesh |
| Routing Algorithm | DOR |
| Routing Latency (number of router pipeline stages) | 4 |
| Channel Latency | 1 |
| Flow Control | Virtual Channel based on Wormhole |
| Virtual Channels | 2 |
| Buffers per VC | 8 |
| Allocator | iSLIP |
| Input Speedup | 1 |
| Channel width (Flit size) | 16 bytes |
| Read request packet size | 8 bytes |
| Read reply packet size | 64 bytes |
| Write packet size | 64+8 bytes |

of MC placements. We also evaluate some of our techniques in an open-loop simulation setup and present the area analysis of our techniques. Finally, we discuss the throughput effectiveness of our techniques.

### 5.1. Methodology

We use a modified version of GPGPU-Sim [Bakhoda et al. 2009], 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 [NVIDIA 2010], using Booksim 2.0 [Jiang et al. 2013] 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 [Wong et al. 2010]). All these changes except the migration from Booksim 1.0 to Booksim 2.0 are currently publicly available as GPGPU-Sim version 2.1.2b.[9]

Table II and IV show our hardware parameters.[10] Configuration abbreviations are listed in Table V. While we are interested in future designs, we chose parameters

[9]The current version of GPGPU-Sim also includes a more realistic model for texture caches that was not present in our previous work [Bakhoda et al. 2010]. This new model slightly changes the simulation results of benchmarks like MUM and DG that use texture caches. We also model a smaller number of entries for the buffers that connect L2 caches to the routers and DRAM controllers (8 entries for the buffers from router towards DRAM controller and 6 entries for the buffers in the other direction). In our previous work all these entries were set to 32. We believe the new settings are more realistic. This change results in an HM slowdown of 0.5% across all benchmarks for the baseline configuration and also slightly reduces the speedup of multiple ejection ports at MC routers.

[10]In our previous work [Bakhoda et al. 2010], we modeled half routers with a 3-stage pipeline though we found the performance impact of one less stage was negligible. In this work half routers have the same number of pipeline stages as full routers (4-stage).

Table V. Abbreviations

| | |
|---|---|
| BW | Bandwidth |
| DOR | Dimension Order Routing |
| CDR | Class-based Deterministic Routing |
| CP | Checkerboard Placement |
| CR | Checkerboard Routing |
| TB | baseline Top-Bottom placement |
| 2P | 2 injection/ejection Port routers at MCs |
| DCI | Double Checkerboard Inverted |
| DCIE | DCI Enhanced |

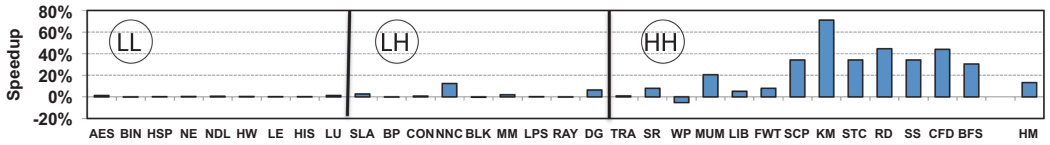



Fig. 24. Overall speedup of using checkerboard placement (CP) of routers compared to baseline top-bottom (TB) placement (both configuration have 2 VCs).

similar to GeForce GTX 280 except for the addition of caches which more closely represent per-thread resources on Fermi.[11] We do this to aid in estimating area overheads of compute portions of the overall accelerator. We use a modified version of Orion 2.0 [Kahng et al. 2009] for network area estimation; Table III shows the corresponding configuration options. Area estimation is explained in more detail in Section 5.9. The benchmarks used in simulation are listed in Table I[12]. We simulate all benchmarks to completion to capture distinct phases of the benchmarks.

### 5.2. Checkerboard Placement (CP)

Figure 24 shows the performance impact of moving the MC nodes from the top-bottom configuration in Figure 4 to the staggered locations shown in Figure 17, while still maintaining full routers and DOR routing. This placement of the MCs benefits from less contention [Abts et al. 2009] and by itself results in an average speedup of 13.2% compared to baseline top-bottom placement.[13] 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). We will revisit the effect of MC placement in more detail in Section 5.7.

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 (5.2%) is due to fairness issues that slow down a few of the compute cores. There are studies [Lee et al. 2008, 2010] that tackle the global fairness in NoCs which are orthogonal to the techniques we introduce in this article.

### 5.3. Checkerboard Routing (CR)

As described in Section 4.2 Checkerboard Routing (CR) requires an extra set of virtual channels to avoid deadlock. Therefore, we also simulate a configuration with 4 VCs

[11]Compute core caches are shared by all threads running on that compute core.
[12]In our previous work [Bakhoda et al. 2010] we used the original CUDA SDK versions of RD and BIN benchmarks. We have upgraded them to their newer 2.3 versions in this work.
[13]Although Abts et al. [2009] showed the network performance improvement of better MC placement, they did not show its impact on overall system performance. Their evaluation was with synthetic traffic patterns and measured network latency.

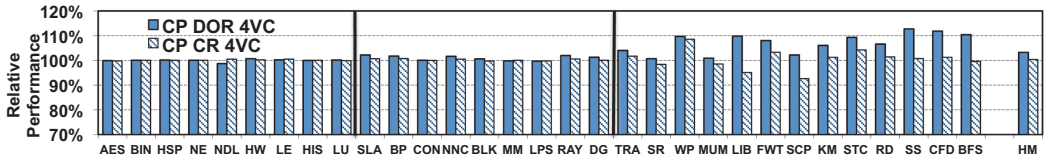



Fig. 25. Relative performance (IPC) of DOR with 4 VCs (solid bars) and Checkerboard Routing (CR) with 4 VCs (hashed bars) compared to DOR routing with 2 VCs; all with Checkerboard Placement (CP). Higher bars mean better performance.

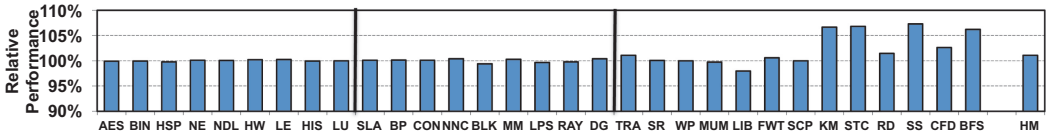



Fig. 26. Relative performance (IPC) of routing from half routers to full routers that are an even number of columns away using YX routing compared to routing using YX to an intermediate node and then XY to destination; both cases are checkerboard routing. Higher bars mean better performance.

and DOR routing to make a fair comparison. Figure 25 shows the relative performance of DOR with 4VCs (solid bars) and checkerboard routing with 4VCs (hashed bars) compared to the DOR routing with 2VCs. These simulations show that using a checkerboard network, with half of the routers being half routers, results in no significant change in performance (on average 0.3% improvement), compared to the 2VC DOR network which requires all full routers. However, checkerboard results in 4.5% and 17.7% reductions in total router area compared to 2VC and 4VC DOR networks, respectively, using area estimations of Section 5.9.[14]

Although a different routing algorithm is required in the checkerboard network, it is still minimal routing (minimal hop count between source and destination) and therefore the checkerboard network has minimal impact on average network latency. Averaged across all benchmarks, 38.3% of packets in the system will require YX routing—one of the three cases explained in Section 4.2.

As explained in Section 4.2, the special Case 3 of CR is routed by always taking the YX route (like Case 1). Figure 26 shows the relative performance of routing Case 3 like Case 1 using YX routing compared to handling Case 3 like Case 2 which involves YX routing to an intermediate node and then XY routing to destination.

Although both alternatives employ minimal routing, handling Case 3 only using YX routing provides a noticeably higher performance for many of HH benchmarks (maximum speedup of 7.3% for SS). Based on our analysis of channel utilizations, using YX routing provides better load-balancing. The other alternative puts more pressure on the routers and links in the center of the chip and increases contention as it involves routing to an intermediate router in the minimal quadrant.

### 5.4. Multiport Routers

Figure 27 shows the speedups of increasing terminal bandwidth of MC routers by adding an extra injection port (3.1%), an extra ejection port (2.2%), and the combination of these changes (5.2%)—as described in Section 4.3 and Figure 20(b). It can be

[14]Area reductions are calculated based on total router area numbers in Table VI which are 37.52mm$^2$, 43.95mm$^2$, and 35.83mm$^2$ for 2VC DOR, 4VC DOR, and Checker Routing (CR) respectively.

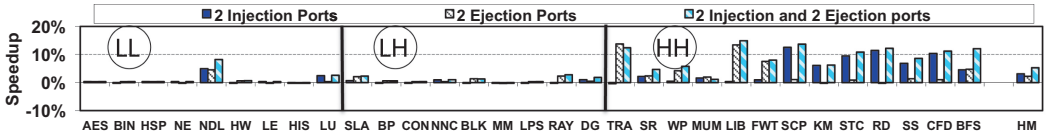



Fig. 27. IPC speedup of adding multiport MC routers versus single CP-CR.

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 (8.1%). The extra ports at MC routers reduce the average fraction of execution time that the injection ports at MCs are blocked (shown in Figure 16) by 58% which provides additional performance benefits. The extra terminal bandwidth provided by the extra injection ports at MCs alleviates the bottleneck at the injection port caused by the many-to-few-*to-many* pattern. Since MCs are servicing many compute cores, the packets will travel in different directions towards their respective destinations after they get in the network. A scattered MC placement like Figure 17 increases the chances of packets going to different directions as they get into the network. On the other hand, in the top-bottom MC placement of Figure 4 a majority of packets that are injected by the MCs will contend for the east/west ports of MC routers. This is caused by the combination of XY routing and presence of MC routers at adjacent locations. As a result, increasing the number of injection ports to two in the top-bottom placement configuration results in a moderate HM speedup of 2.5% (results not shown). It is important to consider the interactions of placement and routing mechanism to better take advantage of multiport MC routers.

Adding extra ejection ports to MC routers is beneficial for benchmarks that have relatively high average injection rates at compute nodes (see Figure 14 for injection rates). High injection rates at compute nodes are typically accompanied by higher ratios of write to read requests (Figure 13). Although processing write requests faster does not directly translate into higher application-level performance, draining the network faster allows the read requests waiting behind the write requests to be processed earlier since the network resources are shared by read and write requests. Some benchmarks that benefit from extra ejection ports are TRA, LIB, and FWT.

We explained a smart port selection mechanism for multiport injection in Section 4.3.1. The goal is to keep the packets that are going to the same output direction together in the same injection port so that when a packet headed to a different direction arrives it goes to another port. This enables packets in different injection ports to start traversing the router earlier as their output directions are less likely to conflict with the output of packets in the other injection port. Enabling this techniques results in modest speedups over the default round-robin network selection mechanism with an HM average of less than 1% (results not shown). The maximum speedup is for the RD benchmarks with a 2% improvement. The speedup of smart port selection is more pronounced in double network designs that we evaluate in the following sections since it will be applied to more routers.

### 5.5. Double Network—Channel-Sliced Network

Conventionally, channel slicing is beneficial if combined with a reduction of the network diameter [Dally and Towles 2004; Kim et al. 2005]; however, we utilize channel slicing without reducing network diameter to reduce network area (Section 5.9). Overall area is reduced because the crossbar area has a quadratic dependency on channel bandwidth.

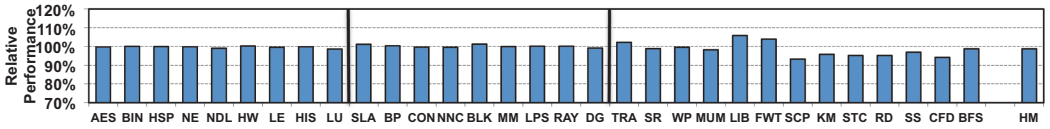



Fig. 28. Relative performance of using two physical subnetworks (combined double network) with channel width 8B compared to a single network with channel width 16B (both have checkerboard routing, checkerboard placement, 4 VCs, and 8 buffers per VC).

As described earlier in Section 4, the traffic with a double network can be distributed either with a dedicated double network or with a combined double network where traffic can be sent across either network. Unfortunately, using a dedicated network results in a 32% harmonic mean slowdown of the benchmarks since it cuts the terminal bandwidth of all nodes in half.[15]

A combined double network is a better choice since it has minimal impact on performance while reducing the crossbar area. Figure 28 compares the performance of the combined double network and the single network. On average, the combined double network has a 1.2% slowdown while total router area is reduced by 31.81% as we show in Section 5.9. The overall throughput effectiveness improves by an average of 2.7%. Note that the aggregate storage used for VC buffers remains constant compared to a single network as explained shortly. Each subnetwork of the double network has the same number of VCs as the single network. While the number of VC buffers in the whole network doubles, each buffer's storage amount is reduced to half since the channel width is also halved.

In addition to maintaining the terminal bandwidth, a combined network provides much better load-balancing of the traffic compared to a dedicated network. A combined network achieves load-balancing because both request and reply packets use both of the physical subnetworks. On the other hand, in the dedicated design the link utilization of the subnetwork dedicated to reply packets is on average 85% more than the link utilization of the subnetwork dedicated to request packets. This load-imbalance is the result of relative abundance of read requests compared to write requests as well as the larger size of read-reply packets compared to read requests.

A fundamental 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 3.3, the additional latency has minimal impact on most workloads and is well tolerated by the compute cores. There are only a few HH benchmarks that experience a slight slowdown.

It is possible to further improve the performance of combined double networks by incorporating some techniques such as an intelligent subnetwork selector that improves load-balancing. However we focus our efforts on the DCI network discussed next which is a more elegant version of double network.

### 5.6. Double Checkerboard Inverted Network (DCI)

In this section we evaluate the *Double Checkerboard Inverted* design and its enhanced version, DCIE, which were introduced in Sections 4.5 and 4.6.1, respectively. DCI is a double network organization where each compute or MC node has both a full and a half router. This design allows all nodes to communicate directly with no extra overhead, and uses a simplified routing algorithm which requires a maximum of one turn in all

[15]In our previous work [Bakhoda et al. 2010], due to a performance bug in our simulator we did not observe the performance loss of dedicated double networks and chose them as our preferred type of double networks.

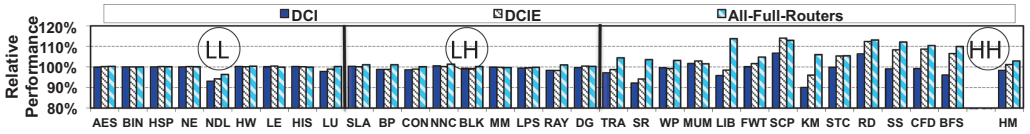
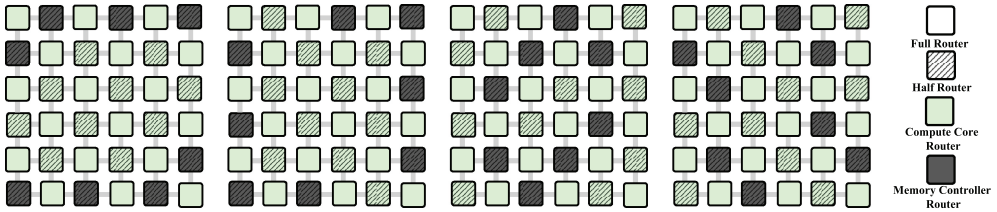



Fig. 29. Relative performance of using "double checkerboard inverted", its enhanced variant, and a combined double network that does not have half routers compared to a single network with channel width 16B (all have checkerboard placement, 4 VCs, and 8 buffers per VC).




(a) PeripheralRow (b) PeripheralCol (c) Scattered 1 (d) Scattered 2 (CP)

Fig. 30. MC placement examples. First two placements keep the MCs on the periphery of the chip while the next two examples allow MCs inside the chip. All these placements are valid checkerboard placements.

cases as described in Section 4.5. DCIE allows the packets that are never going to turn to select their injection subnetwork based on the local historical usage of subnetworks.

Figure 29 shows the relative application-level speedup of using DCI and DCIE over the combined double network design explained in Section 5.5. For comparison, this figure also shows the speedups of a configuration with equivalent resources except that it has no half routers (labeled All-Full-Routers). The All-Full-Routers design does not benefit from the area savings provided by the half router. It does not require any specific injection rules as opposed to DCI which injects to a certain subnetwork based on the distance of the destination node's column. Injection subnetwork is selected randomly for the All-Full-Routers configuration. As Figure 29 shows the harmonic mean speedup of All-Full-Routers over the combined double network is 2.9% across all the benchmarks which is closely followed by the 1.2% speedup of DCIE. DCI has a modest slowdown of 1.7%. DCIE has a 3.4% higher average link utilization compared to DCI and 2.6% compared to the double combined network. The HH benchmarks experience the highest speedups with DCIE as expected. As an example, CFD becomes 8.7% faster.

While it is possible to have DCI and DCIE networks with only two VCs per network, it is not as throughput effective as using four VCs. For example DCIE with 2 VCs would increase throughput effectiveness by only 0.6% compared to a combined double network but at the cost of a 1.7% reduction in performance (results not shown). On the other hand, using 4 VCs increases both performance and throughput effectiveness. Therefore, we employ 4 VC DCI and DCIE configurations.

In summary, DCIE improves system-level performance compared to the combined double network of Section 5.5, has the same NoC area footprint, simplifies the routing algorithm, and removes the restrictions on the placement of MCs.

### 5.7. MC Placement Effects

In this section we examine the impact of MC placement on the techniques introduced before. Figure 30 shows four valid checkerboard placements. As we mentioned in Section 5.2 we picked the placement shown in Figure 30(b) as the best performing placement. This placement scatters the MCs throughout the chip and we believe it can be realized given the current advanced state-of-the-art flip-chip technology. Flip-chip

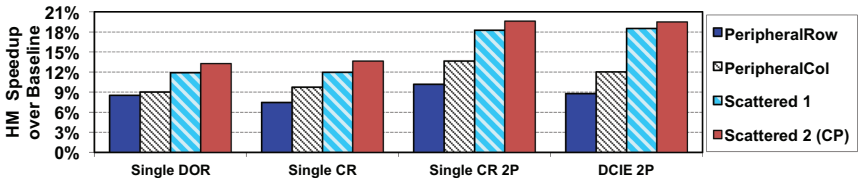



Fig. 31. Sensitivity of checkerboard routing, two injection port MCs, and double checkerboard inverted to placements shown in Figure 30. All speedups are measured against harmonic mean IPC of baseline top-bottom with DOR.

is a technology to connect face-down (flip) electrical components (die) to a package carrier. Flip-chip utilizes one or more top metal layer(s) called *ReDistribution Layer* (RDL) to connect I/O pads on the die to metal bumps on the package carrier. One possible flip-chip structure is area-I/O where by utilizing RDL the chip designers can put I/O pads on any arbitrary location on the area of the die [Fang and Chang 2010]. Alternatively peripheral-I/O flip-chip structure restricts the location of I/O pads to the periphery of the die [Fang and Chang 2010].

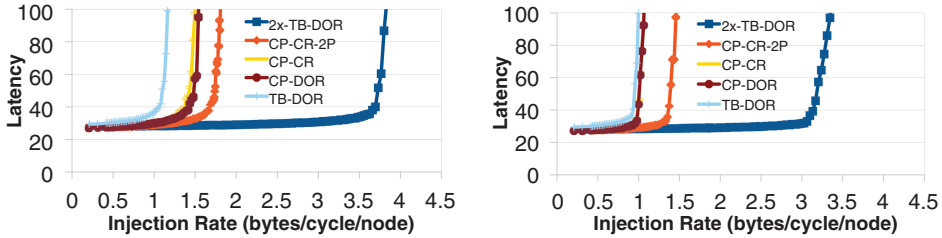
Peripheral MC placements such as Figures 30(a) and 30(b) assume peripheral-I/O while the scattered MC placements such as Figures 30(c) and 30(d) assume area-I/O. Despite many advantages of area-I/O structures such as shorter wire length and better signal integrity, they pose a more challenging routing problem in the RDL compared to peripheral-I/O designs [Fang and Chang 2010]. Nevertheless, we show that the throughput-effective techniques we introduced can still be applied to peripheral MC placements.

Figure 31 shows the sensitivity of the throughput-effective techniques discussed in this article to the placements shown in Figure 30. In this figure, single DOR, single CR and single CR 2P refer to optimizations presented in Sections 5.2, 5.3, and 5.4 (smart port selection enabled) respectively. DCIE 2P combines the enhanced version of double checker inverted presented in Section 5.6 with smart multiport MCs of Section 5.4. Note that these techniques are orthogonal to each other. That is, each MC is attached to one half router and one full router in a DCIE 2P configuration and these routers have two injection and ejection ports. Subnetwork selection and routing inside each subnetwork is governed by DCIE rules while port selection is governed by the smart multiport selection policy.

As shown in Figure 31, the scattered placement of Figure 30(d) consistently performs better than the other placements and when combined with DCIE 2P results in a 19.5% speedup over baseline TB. Single CR 2P also achieves a similar speedup but at a higher area cost. (Section 5.10 provides a detailed comparison of these two configurations.)

Among the peripheral placements, the column-based placement of Figure 30(b) outperforms the row-based placement of Figure 30(a). The PeripheralCol placement combined with DCIE 2P results in a 12% speedup over baseline TB while keeping the MCs on the periphery. (CDR routing provides better performance if a peripheral MC placement is desired as we will describe later in this section.) The reason that column-based placements outperform row-based placements is that the large reply packets leaving the MCs have a higher chance of contention in the row-based versions due to dominance of XY routing.

The most important factor for the MC placement is to *avoid* putting the MCs in neighboring locations when XY routing is employed. When MCs are placed beside each other the packets leaving the MCs start contending immediately as they leave the MC nodes. Additionally, the MCs should not be placed on the central nodes in the mesh because the asymmetrical nature of the mesh increases the traffic passing through




(a) uniform random many-to-few-to-many (b) hotspot many-to-few-to-many

Fig. 32. Latency versus offered load for different architectures. All configurations are simulated with 4 VCs to better distinguish the effects of the techniques introduced in this article. Traffic consists of 90% read requests and 10% write requests. Read requests and write replies are one flit and read-reply and write requests are 4 flits except for 2x-TB-DOR where they are 2 flits. Flits are 16 bytes except for 2x-TB-DOR where they are 32 bytes. The $x$-axis is the average number of bytes injected from each compute node per cycle.

the center of the mesh and putting a hotspot node in such a location creates extra contention.

While we believe a scattered MC placement can be realized in hardware as we explained in the beginning of this section, if a peripheral placement of MCs is required then slightly more complex routing techniques such as Class-based Deterministic Routing (CDR) will provide better performance compared to DOR routing. For example, employing DCIE 2P in the top-bottom placement of Figure 4 (which is a type of row-based peripheral placement) results in a 5.1% slowdown over the baseline. When we employ the CDR variant of DCIE 2P (introduced in Section 4.6) with the same placement, a 13.3% improvement is achieved over the baseline.

CDR performs better than DOR in row-based placements because it employs YX routing for reply traffic. In XY routing packets ejected from MCs start to contend in already congested neighboring MC routers. Employing YX routing for reply packets reduces the amount of contention for the reply packets as they leave the MCs.

As mentioned earlier, when we move MCs apart from each other, contention is reduced and performance is increased. That is, if we use DCIE 2P and CDR with the peripheral row-based placement of Figure 30(a) we will get a 15.4% performance improvement over the baseline. This combination (DCIE-CDR-2P with PeripheralRow placement) is our best performing peripheral configuration.

Similar to observations by Abts et al. [2009] employing CDR with a scattered placement does not result in further improvements in performance (compared to DOR routing). In our experiments changing DOR routing to CDR routing for the scattered placement of Figure 30(d) using DCIE 2P results in a slowdown of 1.3%.

### 5.8. Open-Loop NoC Simulation Study

Figure 32 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 hotspot traffic 20% of requests go to one MC as opposed to 12.5% (1/8) for uniform random. These open-loop simulations use a single network with two logical networks for request and reply traffic. Checkerboard Routing (CR) and DOR have almost the same latency in both traffic patterns as checkerboard routing is minimal. 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 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 the number of injection ports at the MCs (note read response packets are larger than read request packets). For the hotspot traffic the improvements of MC placement are more moderate. The hotspot node is MC 0 which is located on the top right corner of the chip in our scattered MC placement. Adding the extra injection ports at MCs improves performance significantly by alleviating the bottlenecks created by hotspot traffic. Although addresses are low-order interleaved among MCs every 256 bytes [Harris 2009] to reduce hotspots we have observed that temporary hotspots happen in closed-loop simulations.

### 5.9. Area Analysis

We use a modified version of Orion 2.0 [Kahng et al. 2009] to estimate the area of buffers, allocators, and links of the various NoC designs we explore in this article. Recent studies have highlighted the inaccuracies of Orion 2.0 especially for technology nodes smaller than 65nm [Sun et al. 2012]. Therefore, we calibrate the crossbar area estimation portion of Orion as described shortly. First we calculate the number of crosspoints needed to implement the crossbar and then multiply this number by the area of a single crosspoint. We use an empirical method to estimate the area of the crosspoint. The number of crosspoints is calculated according to the following formula.

$$nCrosspoints = nX * nY$$

$$nX = nI * CW$$

$$nY = nO * CW$$

$CW$ denotes channel width in bits and $nI$ and $nO$ denote the number of input and output ports, respectively.

To calculate the area of a single crosspoint we use an empirical model where we first measure the crosspoint area of three router designs for which we have access to their crossbar area numbers [Kumar et al. 2007a; Vangal et al. 2008; Salihundam et al. 2010]. We extract the crossbar area from the provided die photos. The crossbar in Salihundam et al. [2010] is implemented in 45nm technology. Its area is scaled to 65nm before calculating its crosspoint area. We calculate the crosspoint area as the average crosspoint area of the three designs mentioned earlier which turns out to be $2.07um^2$. Unmodified Orion's crosspoint area is around $4.24um^2$. That is, our estimation technique provides crossbar areas that are over 2 times smaller than unmodified Orion. Table VI provides the area estimates for the designs we evaluated.

The area of non-NoC parts of the chip was estimated by measurements from GTX280's die photo. We estimated each compute core to be $5.25mm^2$ and each MC to be $10.48mm^2$. Our baseline of 28 compute nodes and 8 MCs adds up to $230.84mm^2$. We also estimated an area of $13.84mm^2$ for L2 caches using Cacti. As a result the total area of non-NoC components of our baseline is $244.68mm^2$. Adding this number to the estimated NoC area for each network design results in the total chip area (last column of table). We assume the injection/ejection links have a length of $0.05mm$ and include their contribution to area under "Link Area" column. Note that this overhead is negligible compared to the size of other components as a 64-bit local link would take around $0.001mm^2$.

The first row of Table VI shows the area of the baseline mesh with a channel width of 16 bytes and the second row a mesh with a 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 40% of chip area, makes it impractical to build. By exploiting half routers, which occupy only 63%

Table VI.
Area Estimations ($mm^2$) A "/" separates different router types for configurations that have more than one router type.

| | Area of 1 link | Crossbar Area | Buffer Area | Alloc. Area | Area of 1 Router | Link Area | Router Area | % of NoC Overhead | Total Chip Area |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | 0.11 | 0.87 | 0.17 | 0.001 | 1.04 | 13.36 | 37.54 | 17.2% | 295.6 |
| 2x-BW | 0.219 | 3.5 | 0.34 | 0.001 | 3.82 | 26.68 | 137.56 | 40.1% | 408.9 |
| CP DOR 4VC | 0.11 | 0.87 | 0.34 | 0.004 | 1.22 | 13.36 | 43.95 | 18.8% | 301.99 |
| CP-CR | 0.11 | 0.87/ 0.42 | 0.34/ 0.34 | 0.004/ 0.004 | 1.22/ 0.76 | 13.36 | 35.83 | 16.7% | 293.87 |
| CP-CR 2P | 0.11 | 0.87/ 0.42/ 0.42 | 0.34/ 0.34/ 0.69 | 0.004/ 0.004/ 0.004 | 1.22/ 0.76/ 1.12 | 13.39 | 36.8 | 17.0% | 294.88 |
| Double CP-CR or DCI w/4VC | 0.055 | 0.22/ 0.10 | 0.17/ 0.17 | 0.004/ 0.004 | 0.40/ 0.28 | 13.40 | 24.43 | 13.39% | 282.51 |
| DCI w/2VC | 0.055 | 0.22/ 0.10 | 0.087/ 0.087 | 0.001/ 0.001 | 0.30/ 0.19 | 13.40 | 17.88 | 11.34% | 275.96 |
| DCI 2P w/2VC | 0.055 | 0.22/ 0.10/ 0.27/ 0.17 | 0.087/ 0.087/ 0.10/ 0.10 | 0.001/ 0.001/ 0.001/ 0.001 | 0.30/ 0.19/ 0.38/ 0.28 | 13.43 | 19.20 | 11.77% | 277.34 |
| DCI 2P w/4VC | 0.055 | 0.22/ 0.10/ 0.27/ 0.17 | 0.17/ 0.17/ 0.20/ 0.20 | 0.004/ 0.004/ 0.004/ 0.004 | 0.39/ 0.28/ 0.49/ 0.38 | 13.43 | 26.02 | 13.88% | 284.14 |

of the area of a full router[16], the checkerboard network results in a 4% reduction in total router area (comparing sum of router area numbers which are $37.6mm^2$ in 65nm for checkerboard and $36.2mm^2$ for baseline router). By further taking advantage of the quadratic dependency, the double network reduces the area further by 30% (comparing sum of router area numbers for combined double checkerboard and single checkerboard which keeps the buffer sizes constant). The area of DCI with 4 VCs is also the same as double CP CR network. Table VI's "DCI 2P w/4VC" row shows the area of the 4 VC DCI configuration with 2 injection/ejection ports at MC nodes; multiport MC routers increase the NoC area overhead by 4.2%. As the chip scales and the number of compute nodes increases in the future, the relative overhead of multiport router decreases because the number of MCs scales at a much slower pace.

DCI needs to determine its injection subnetwork based on the distance to destination node column. If we assume table-based routing, we need a lookup table with a 1-bit entry per destination, that is, overhead is a 36-bit lookup table per node. This table can be further optimized to have 8 entries for the compute nodes and 28 entries for MCs.

### 5.10. Throughput Effectiveness

Combining the optimizations in Sections 5.2, 5.3, and 5.4 (checkerboard placement, checkerboard routing, and 2 injection ports at MC routers, i.e., single CP-CR-2P) results in a 19.6% speedup versus our baseline introduced in Section 2 as shown in Figure 33. Compared with 42.3% speedup of an ideal network, this throughput effective network

[16]The half-router crossbar area is calculated by adding the area of four $2\times1$ crossbars (two for each dimension) and one $4\times1$ crossbar for the ejection port. Full-router crossbars are modeled as a single $5\times5$ crossbar which is pessimistic since a packet arriving on a given port cannot depart through the same port.

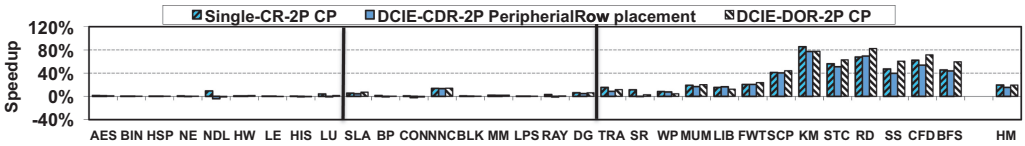



Fig. 33. IPC speedup of three throughput-effective designs over baseline top-bottom with DOR.

achieves 47% of the performance possible with an ideal network while reducing area. This design improves throughput effectiveness (IPC/$mm^2$) by 19.9% (using the area numbers in baseline and CP-CR 2p rows of Table VI). The majority of the gain in throughput effectiveness is the result of increased performance rather than reduced area.

Utilizing channel slicing and the simpler routing of the DCIE technique evaluated in Section 5.6 along with the preceding techniques results in a 19.5% speedup over the baseline while reducing the area compared to a single CP-CR-2P configuration. The speedups over baseline are shown as DCIE-DOR in Figure 33. This design offers the best trade-off of performance and area among the combinations of proposed techniques and results in a 24.3% improvement in throughput effectiveness (IPC/$mm^2$). It was shown as "Thr. Eff." point in Figure 2.

If a peripheral placement is desired then CDR routing can be employed. Using CDR routing with the PeripheralRow-based MC placement of Figure 30(a) results in a 15.4% speedup over baseline without changing the area footprint of DCIE. The speedups of this design can be seen as DCIE-CDR bars in Figure 33. Using this technique improves throughput effectiveness (IPC/$mm^2$) by 20.0%.

## 6. DISCUSSION

In this section, we discuss interactions of concentration with our proposed techniques and provide comparisons with other topologies. We also discuss scalability and power issues. In this section, references to a "throughput-effective" design refer to a DCIE 2P network with scattered placement of MCs similar to Figure 30(d) and 2 VCs per subnetwork.[17]

### 6.1. Concentration

Concentration is a form of "hierarchical" network with the first level of the network hierarchy being the "concentration" or a simple multiplexor (the actual implementation can vary). The techniques provided in this work are orthogonal to concentration. Our techniques are suited for cases with many compute nodes and few MCs. We expect the many-to-few-to-many property to be true regardless of compute node concentration. Given the slow rate of increase in the number of pins per chip [ITRS 2008] the number of MCs will remain at their current level well into the future.

Concentration has been proposed for NoCs in the form of high-radix routers like CMesh and flattened butterfly [Balfour and Dally 2006; Kim et al. 2007]. We refer to this form of concentration as integrated concentration and will discuss it in Section 6.2. An alternative implementation would be applying concentration to multiple input sources first (i.e., external concentration [Kumar et al. 2009]) and then feeding the router only with a single port. In this approach the router radix is the same as the case without concentration. We refer to this form of concentration as external concentration.

[17]In Section 5.6 we mentioned having a DCIE with 4 VCs is more throughput effective than having 2 VCs. The studies of this section were performed before we came to that conclusion and assume 2 VCs.

Current GPUs seem to be employing a combination of external concentration and crossbars. We now provide a comparison of this approach with mesh-based topologies and the reasons it needs to change in the future. As mentioned earlier, a crossbar would be connecting components that are scattered throughout the chip. Routing very long wires to and from a central crossbar is neither easy nor inexpensive. We simulated an asymmetric crossbar NoC with a $10 \times 8$ router for requests and an $8 \times 10$ router for replies and external concentration of 3 for the compute nodes. If we assume a flit size of 16 bytes, the overall HM IPC is 1.1% better than our baseline while the estimated NoC area is around a quarter of the baseline. We are not including the links in area comparison for simplicity. It is clear that an asymmetric crossbar is better in terms of throughput effectiveness compared to baseline. On the other hand, in order to achieve a performance similar to our best throughput-effective design we need to increase the flit size of the asymmetric crossbar to 24 bytes. At this point the crossbar has an HM IPC of 2% more than our DCIE 2P design while its total NoC area is 12% smaller.

Nevertheless as the number of nodes and bandwidth demand grows in the future the crossbar will lose its advantage as its complexity goes up with the number of ports and flit size requirements. Routing long links to connect to a central crossbar becomes problematic and will require multiple cycles. A large crossbar also becomes a single point of failure bringing down yield. On the other hand, if a mesh is faulty the chip can still be sold after disabling some rows and/or columns. A mesh would provide low area, good performance, and regularity in design and high yields. Another consideration is that our mesh-based DCI technique can be easily extended to support coherence in a system like Intel's Knights Corner [Seiler et al. 2008] by adding virtual channels. However, asymmetric crossbars cannot support coherence protocols that require compute nodes to communicate directly; coherence would require a fully connected symmetric central crossbar in such a case. Note that all the nodes connected to a symmetric crossbar can directly send packets to each other while an asymmetric crossbar only allows the nodes on opposing sides of the crossbar to communicate. For example, in the configuration simulated earlier, the 10 concentrated compute nodes cannot directly communicate with each other.

### 6.2. Topology

In this section we compare our proposed throughput-effective technique (DCIE 2P with scattered placement) with two previously proposed topologies: flattened butterfly [Kim et al. 2007] and CMesh [Balfour and Dally 2006]. Flattened butterfly is intended to minimize latency which is not the first priority in our system; our priority is increasing throughput. Since flattened butterfly is a high-radix topology, we used a 64-node system as our baseline to have a meaningful comparison. The 64-node baseline is configured as an $8 \times 8$ mesh with DOR routing and a top-bottom MC arrangement. For the following comparisons we simulated 56 compute cores and 8 MCs and kept the bisection bandwidth constant.

Our throughput effective technique results in a 28% HM speedup while flattened butterfly results in a 17% HM slowdown over the 64-node baseline. That is, our throughput-effective technique is 54% HM faster than flattened butterfly. As the bisection bandwidth is held constant, the link width of flattened butterfly is 8 bytes. This means the terminal bandwidth of MC nodes in the flattened butterfly configuration is half of the baseline single mesh and a quarter of DCIE 2P mesh configuration. This is the primary reason for the slowdown of flattened butterfly and results in DCIE 2P being more throughput effective, even though the flattened butterfly has a smaller area (total router area is 41% smaller than DCIE 2P). The performance of flattened butterfly can be improved with the multiport MC technique (Section sec:2port) but this would also increase the cost of the flattened butterfly network.

We also simulated a CMesh with express channels [Balfour and Dally 2006] while keeping the bisection bandwidth constant. CMesh with express channels results in a 19% HM speedup over baseline 64-node. That is, CMesh is 7% slower than our throughput-effective technique while its area is around 21% higher. If we assume a CMesh with no express channels, then keeping bisection bandwidth constant results in a flit size of 32 bytes which translates into a total router area that is 4.5× larger than our throughput-effective technique. This CMesh topology is also 12% faster than the DCIE 2P technique but not as throughput effective. Note that all of our techniques are applicable to CMesh and can increase its efficiency, although the benefit of half routers diminishes as the number of local ports increases.

### 6.3. Scalability

We expect the benefits of our techniques to increase as the number of compute cores increases in the future, as the number of MCs will not scale at the same pace as compute nodes which exacerbates the many-to-few-to-many issue. In fact, the number of MCs has been kept under eight by the industry. For example, while NVIDIA's GTX280 has 8 MCs, more recent GTX480 and GTX TITAN have 6 MCs.

In order to show the scalability of our proposed techniques we simulated an $11 \times 11$ system with 112 compute nodes and 8 MCs in addition to the 64-node configuration mentioned in Section 6.2. One node was left empty to prevent load-balancing issues in our benchmarks that can skew the results for the $11 \times 11$ configuration. Applying DCIE 2P technique results in a 28% speedup and a 50% total router area reduction for the 64-node system. When same techniques are applied to the 120-node configuration, a 43% speedup and a 51% total router area reduction is achieved. In our baseline 36-node configuration, similar comparisons result in a 11.9% speedup and a 49% total router area reduction. These results indicate that benefits of our techniques increase in terms of both speedup and area reduction as the network scales and becomes larger.

We also expect the benefits of reducing NoC area to increase as technology nodes get smaller. As the technology scales other components on the chip also scale down, so the ratio of routers to the total chip area does not go down. In fact, since: (1) the wire pitch is not scaling at a lower rate than transistors and (2) crossbars and links are wire dominated, we expect that the ratio of interconnect will increase in smaller technology nodes. For example in Intel's 65nm and 45nm technology nodes, transistors scale at $65/45 = 0.69$ rate while wire pitch scales at $210nm/160nm = 0.76$ rate [Bai et al. 2004; Ingerly et al. 2008]. This means wire-dominated parts like a crossbar become lager relative to other components. As a real example, 5-port 144-bit routers occupy 1.17 $mm^2$ in 45nm technology [Salihundam et al. 2010] which is not particularly small.

### 6.4. Power

While power is increasingly becoming an important factor in NoC design, our focus in this article is to increase performance while reducing area. Nevertheless the design process for a power-centric approach can be driven utilizing the same framework that we used to arrive at our throughput-effective organizations. For example, a weighted combination of power and area can be used as the denominator of throughput-effective equation (we used IPC/$mm^2$). In general, the power consumed in the router is dominated by buffer, crossbar, and the channels [Sun et al. 2012]. Our throughput-effective approach reduces NoC area by reducing complexity of these components—and thus, we expect the power consumed to be proportionally reduced as well. Our approach never increases the distance traveled by packets on the chip, as it has minimal routing. In fact, the scattered MC placement of checkerboard routing reduces the average packet hop count. The reduced hop counts should directly translate into dynamic power savings.

Crossbars in half routers have fewer connections and thus save dynamic power as their capacitance is smaller compared to full routers.

## 7. RELATED WORK

### 7.1. Accelerator Architectures

Rigel [Kelm et al. 2010a] is an accelerator that is fundamentally similar to our architecture but provides a more flexible programming model compared to CUDA and chooses an MIMD model rather than SIMT. NoC design of the Cell [Kistler et al. 2006] architecture is an example of making trade-offs between network area and latency. The Cell designers chose a ring over a crossbar to meet their area and power constraints [Krolak 2005]. The choice of centralized arbiters can limit scalability. UltraSPARC T2 [Sun Microsystems Inc. 2007] microprocessor is a multithreading, multicore CPU that uses a crossbar interconnect. GPUs and Cell are both related to stream computing [Dally et al. 2003; Ahn et al. 2004].

### 7.2. Interconnection Networks

Increasing the number of cores on a single chip has increased the importance of NoCs. Much of the research in NoC have focused on reducing network latency by improving different aspects of NoC such as lower-latency router microarchitectures [Mullins et al. 2004; Kumar et al. 2007b], lower-diameter topologies [Balfour and Dally 2006; Kim et al. 2007; Grot et al. 2009], or better flow control [Kumar et al. 2008, 2007a]. However, as we showed in Section 3, compute accelerator applications are more sensitive to bandwidth and reducing latency results in minor improvements in their overall performance. Bufferless routing [Moscibroda and Mutlu 2009] 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. [2009] where the impacts 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. [2009] proposed a complexity-effective DRAM access scheduling technique for manycore accelerators that relies on modification to the arbitration scheme in the request path of NoC.

There are other recent attempts to specialize the NoC for many-to-few-to-many traffic patterns. Lotfi-Kamran et al. [2012] segregate core and last-level cache slices into different tiles; as a result, the traffic pattern becomes many-to-few-to-many. Similar to our work, they apply the concept of reducing connectivity to decrease the area of NoC. Unlike throughput accelerator workloads, their target domain of server workloads is dominated by applications with high latency sensitivity.

Abts et al. [2009] studied alternative MC placements for core-memory traffic; however, they did not show the 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 [Nesson and Johnsson 1995]. 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 has been proposed [Balfour and Dally 2006; Kim et al. 2007] to reduce the network diameter and increase network performance, mainly through lower latency. The multiport 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 [Kim 2009]. 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. Volos et al. [2012] employ a double network organization where each subnetwork is specialized based on the packet types traversing it. The resulting asymmetric network enables energy saving in their target domain of cache-coherent server workloads.

Das et al. [2009] employ a hybrid of bus and mesh interconnects which is beneficial when there is locality in traffic. However, unlike CMPs there is no locality in our traffic pattern as the communication is between the many cores and the few MCs. The heterogeneous NoC work by Mishra et al. [2011] was done concurrently with our work and utilizes the same concepts as our work which is improving a few routers of NoC while improving the overall system performance. However, their work is orthogonal to our work—they leverage the heterogeneity of the mesh topology to create their heterogeneous network while our work leverages the heterogeneity of the traffic pattern between the many nodes and few MCs.

## 8. CONCLUSION

In this article, we analyze the impact of communication and on-chip network across a wide range of applications in throughput accelerators. We describe how reducing router latency does not significantly improve overall application-level performance but increasing costly bandwidth does. Then we show how the many-to-few-*to-many* traffic pattern creates a bottleneck in the on-chip network as a result of large and frequent read-reply packets. We focus on improving the *throughput effectiveness* of 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 crossbar area while maintaining a minimal routing algorithm. We then employ multiport routers at MCs to increase the terminal bandwidth of the network and alleviate the many-to-few-*to-many* bottleneck. We then propose the "double checkerboard inverted" network which utilizes half routers in a channel-sliced network. This technique further improves performance by providing better load-balancing and results in a 24.3% improvement of throughput effectiveness over the baseline. We also propose an extension of the double checkerboard inverted that employs class-based deterministic routing and can be utilized to provide a similar throughput effectiveness if a peripheral placement of MCs is desired.

## ACKNOWLEDGMENT

We thank Wilson W. L. Fung and the anonymous reviewers for their valuable feedback on this work.

## REFERENCES

Abts, D., Jerger, N. D. E., Kim, J., Gibson, D., and Lipasti, M. H. 2009. Achieving predictable performance through better memory controller placement in many-core cmps. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'09)*. ACM Press, New York, 451–461.

Ahn, J. H., Dally, W. J., Khailany, B., Kapasi, U. J., and Das, A. 2004. Evaluating the imagine stream architecture. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'04)*. IEEE Computer Society, Washington, DC, 14–25.

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

Bai, P., Auth, C., Balakrishnan, S., Bost, M., Brain, R., et al. 2004. A 65nm logic technology featuring 35nm gate lengths, enhanced channel strain, 8 cu interconnect layers, low-k ild and 0.57 um2 sram cell. In *Proceedings of the IEEE International Electron Devices Meeting, IEDM Technical Digest.* 657–660.

BAKHODA, A., KIM, J., AND AAMODT T. M. 2010. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO'10)*. IEEE Computer Society, Washington, DC, 421–432.

BAKHODA, A., YUAN, G. L., FUNG, W. W. L., WONG, H., AND AAMODT, T. M. 2009. Analyzing cuda workloads using a detailed gpu simulator. In *Proceedings of the IEEE Symposium on Performance Analysis of Systems and Software (ISPASS'09)*. 163–174.

BALFOUR, J. D. AND DALLY, W. J. 2006. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the ACM Conference on Supercomputing (ICS'06)*. ACM Press, New York, 187–198.

CHE, S., BOYER, M., MENG, J., TARJAN, D., SHEAFFER J. W., LEE, S.-H., AND SKADRON, K. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the IEEE Symposium on Workload Characterization (IISWC'09)*. 44–54.

COON, B. W. AND LINDHOLM, E. J. 2008. US patent 7,353,369: System and method for managing divergent threads in a simd architecture. https://www.google.com/patents/US7353369.

SDK, C. 2009. NVIDIA CUDA SDK code samples. http://developer.nvidia.com/object/cuda sdk samples.html.

DALLY, W. J., LABONTE, F., DAS, A., HANRAHAN, P., AHN, J.-H., GUMMARAJU, J., EREZ, M., JAYASENA, N., BUCK, I., KNIGHT, T. J., AND KAPASI, U. J. 2003. Merrimac: Supercomputing with streams. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. ACM Press, New York, 35.

DALLY, W. J. AND TOWLES, B. 2004. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Fransisco, CA.

DAS, R., EACHEMPATI, S., MISHRA, A. K., NARAYANAN, V., AND DAS, C. R. 2009. Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. In *Proceedings of the IEEE Symposium on High-Performance Computer Architecture (HPCA'09)*. 175–186.

FANG, J.-W., AND CHANG, Y.-W. 2010. Area-I/O flip-chip routing for chip-package co-design considering signal skews. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 29*, 5, 711–721.

FUNG, W. W. L., SHAM, I., YUAN, G., AND AAMODT, T. M. 2007. Dynamic warp formation and scheduling for efficient GPU control flow. In *Proceedings of the 40$^{th}$ IEEE/ACM Symposium on Microarchitecture (MICRO'07)*. IEEE Computer Society, Washington, DC, 407–420.

GROT, B., HESTNESS, J., KECKLER, S. W., AND MUTLU, O. 2009. Express cube topologies for on-chip interconnects. In *Proceedings of the IEEE Symposium on High-Performnce Computer Architecture (HPCA'09)*. 163–174.

HARRIS, M. 2009. UNSW CUDA tutorial part 4 optimizing CUDA. http://cs.anu.edu.au/files/systems/GPUWksp/PDFs/04 OptimizingCUDA full.pdf.

INGERLY, D., AGRAHARAM, S., BECHER, D., CHIKARMANE, V., FISCHER, K., ET AL. 2008. Low-k interconnect stack with thick metal 9 redistribution layer and cu die bump for 45nm high volume manufacturing. In *Proceedings of the International Interconnect Technology Conference (IITC'08)*. 216–218.

ITRS. 2008. International technology roadmap for semiconductors 2008 update. http://www.itrs.net/Links/2008ITRS/Home2008.htm.

JIANG, N., BECKER, D. U., MICHELOGIANNAKIS, G., BALFOUR, J., TOWLES, B., KIM, J., AND DALLY, W. J. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proceedings of the IEEE Symposium on Performance Analysis of Systems and Software (ISPASS'13)*. 86–96.

KAHNG, A., LI, B., PEH, L.-S., AND SAMADI, K. 2009. ORION 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the IEEE/ACM Conference on Design Automation and Test in Europe (DATE'09)*. 23–428.

KELM, J. H., JOHNSON, D. R., LUMETTA, S. S., FRANK, M. I., AND PATEL, S. 2010a. A task-centric memory model for scalable accelerator architectures. *IEEE Micro 30,* 1, 29–39.

KELM, J. H., JOHNSON, D. R., TOUHY, W., LUMETTA, S. S., AND PATEL, S. 2010b. Cohesion: A hybrid memory model for accelerator architectures. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'10)*. ACM Press, New York, 429–440.

KESSLER, R. E., AND SCHWARZMEIER, J. L. 1993. Cray t3d: A new dimension for cray research. In *Compcon Spring Digest of Papers*. 176–182.

GROUP, K. 2010. OpenCL - The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl/.

KIM, J. 2009. Low-cost router microarchitecture for on-chip networks. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO'09)*. 255–266.

KIM, J., BALFOUR, J., AND DALLY, W. 2007. Flattened butterfly topology for on-chip networks. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO'07)*. IEEE Computer Society, Washington, DC, 172–182.

KIM, J., DALLY, W. J., TOWLES, B., AND GUPTA, A. K. 2005. Microarchitecture of a high-radix router. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'05)*. IEEE Computer Society, Washington, DC, 420–431.

KISTLER, M., PERRONE, M., AND PETRINI, F. 2006. Cell multiprocessor communication network: Built for speed. *IEEE Micro 26,* 3, 10–23.

KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. 2005. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro 25,* 2, 21–29.

KROLAK, D. 2005. Cell broadband engine eib bus. http://www.ibm.com/developerworks/power/library/paexpert9/.

KUMAR, A., KUNDU, P., SINGH, A., PEH, L.-S., AND JHA, N. 2007a. A 4.6tbits/s 3.6ghz singlecycle noc router with a novel switch allocator in 65nm cmos. In *Proceedings of the IEEE Conference on Computer Design (ICCD'07)*. 63–70.

KUMAR, A., PEH, L.-S., KUNDU, P., AND JHAY, N. K. 2007b. Express virtual channels: Towards the ideal interconnection fabric. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'07)*. ACM Press, New York, 150–161.

KUMAR, A., PEH, L.-S., AND JHA, N. K. 2008. Token flow control. In *Proceedings IEEE/ACM Symposium on Microarchitecture (MICRO'08)*. IEEE Computer Society, Washington, DC, 342–353.

KUMAR, P., PAN, Y., KIM, J., MEMIK, G., AND CHOUDHARY, A. N. 2009. Exploring concentration and channel slicing in on-chip network router. In *Proceedings of the IEEE/ACM Symposium on Networks-on-Chip (NOCS'09)*. 276–285.

LEE, J. W., NG, M. C., AND ASANOVIC, K. 2008. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'08)*. IEEE Computer Society, Washington, DC, 89–100.

LEE, M. M., KIM, J., ABTS, D., MARTY, M., AND LEE, J. W. 2010. Probabilistic distance-based arbitration: Providing equality of service for many-core CMPs. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO'10)*. IEEE Computer Society, Washington, DC, 509–519.

LEVINTHAL, A., AND PORTER, T. 1984. Chap - A simd graphics processor. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'84)*. 77–82.

LOTFI-KAMRAN, P., GROT, B., AND FALSAFI, B. 2012. NOC-out: Microarchitecting a scale-out processor. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO'12)*. IEEE Computer Society, Washington, DC, 177–187.

MISHRA, A. K., VIJAYKRISHNAN, N., AND DAS, C. R. 2011. A case for heterogeneous on-chip interconnects for CMPs. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'11)*. ACM Press, New York, 389–400.

MOSCIBRODA, T., AND MUTLU, O. 2009. A case for bufferless routing in on-chip networks. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'09)*. ACM Press, New York, 196–207.

MULLINS, R. D., WEST, A., AND MOORE, S. W. 2004. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'04)*. IEEE Computer Society, Washington, DC, 188–197.

NESSON, T., AND JOHNSSON, S. L. 1995. ROMM routing on mesh and torus networks. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)*. ACM Press, New York, 275–287.

NICKOLLS, J., BUCK, I., GARLAND, M., AND SKADRON, K. 2008. Scalable parallel programming with CUDA. *ACM Queue 6,* 2, 40–53.

NICKOLLS, J. R., COON, B. W., AND SHEBANOW, M. C. 2011. US patent application 20110072213: Instructions for managing a parallel cache hierarchy (Assignee NVIDIA Corp.). March.

NVIDIA. 2009. NVIDIA's next generation CUDA compute architecture: Fermi. http://openclcomputing.com/index.php/cuda/10-fermi.

NVIDIA 2010. *NVIDIA CUDA Programming Guide,* 3.0 ed. NVIDIA.

PEH, L.-S. AND DALLY, W. J. 2001. A delay model and speculative architecture for pipelined routers. In *Proceedings of the IEEE Symposium on High-Performance Computer Architecture (HPCA'01)*. IEEE Computer Society, Washington, DC, 255–266.

PFISTER, G. F. AND NORTON, V. A. 1985. Hot-spot contention and combining in multistage interconnection networks. *IEEE Trans. Comput. 34,* 10, 943–948.

PULLINI, A., F., ANGIOLINI, A., MURALI, S., ATIENZA, D., MICHELI, G. D., AND BENINI, L. 2007. Bringing nocs to 65 nm. *IEEE Micro 27,* 5, 75–85.

RIXNER, S., DALLY, W. J., KAPASI, U. J., MATTSON, P., AND OWENS, J. D. 2000. Memory access scheduling. In *Proceedings of the 27$^{th}$ International Symposium on Computer Architecture*. ACM Press, New York, 128–138.

RYOO, S., RODRIGUES, C., STONE, S., BAGHSORKHI, S., UENG, S.-Z., STRATTON, J., AND HWU, W.-M. W. 2008. Program optimization space pruning for a multithreaded GPU. In *Proceedings of the IEEE/ACM Symposium on Code Generation and Optimization (CGO'08)*. ACM Press, New York, 195–204.

SALIHUNDAM, P., JAIN, S., JACOB, T., KUMAR, S., ERRAGUNTLA, V., ET AL. 2010. A 2tb/s 6*4 mesh network with DVFS and 2.3tb/s/w router in 45nm CMOS. In *Proceedings of the IEEE Symposium on VLSI Circuits (VLSIC'10)*.79–80.

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: A many-core x86 architecture for visual computing. *ACM Trans. Graph. 27,* 3, 18:1–18:15.

SEO, D., ALI, A., LIM, W.-T., RAFIQUE, N., AND THOTTETHODI, M. 2005. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proceedings of the IEEE/ACM Symposium on Computer Architecture (ISCA'05)*. 432–443.

SUN, C., CHEN, C.-H. O., KURIAN, G., WEI, L., MILLER, J., AGARWAL, A., PEH, L.-S., AND STOJANOVIC, V. 2012. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the IEEE/ACM Symposium on Networks-on-Chip (NOCS'12)*. IEEE Computer Society, Washington, DC, 201–210.

SUN MICROSYSTEMS INC. 2007. OpenSPARCTM t2 core microarchitecture specification. http://www.oracle.com/technetwork/systems/opensparc/t2-06-opensparct2-core-microarch-1537749.html.

VALIANT, L. G. 1990. A bridging model for parallel computation. *Comm. ACM 33,* 8, 103–111.

VALIANT, L. G. AND BREBNER, G. J. 1981. Universal schemes for parallel communication. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'81)*. ACM Press, New York, 263–277.

VANGAL, S. R., HOWARD, J., RUHL, G., DIGHE, S., WILSON, H., ET AL. 2008. An 80-tile sub-100-w teraflops processor in 65-nm CMOS. *IEEE J. Solid-State Circ. 43,* 1, 29–41.

VOLOS, S., SEICULESCU, C., GROT, B., POUR, N. K., FALSAFI, B., AND MICHELI, G. D. 2012. CCNoC: Specializing on-chip interconnects for energy efficiency in cache-coherent servers. In *Proceedings of the IEEE/ACM Symposium on Networks-on-Chip (NOCS'12)*. IEEE Computer Society, Washington, DC, 67–74.

WENTZLAFF, D., GRIFFIN, P., HOFFMANN, H., BAO, L., EDWARDS, B., RAMEY, C., MATTINA, M., MIAO, C.-C., BROWN III, J. F., AND AGARWAL, A. 2007. On-chip interconnection architecture of the tile processor. *IEEE Micro 27,* 15–31.

WONG, H., BRACY, A., SCHUCHMAN, E., AAMODT, T. M., COLLINS, J. D., WANG, P. H., CHINYA, G., GROEN, A. K., JIANG, H., AND WANG, H. 2008. Pangaea: A tightly-coupled ia32 heterogeneous chip multiprocessor. In *Proceedings of the IEEE/ACM Conference on Parallel Architectures and Compilation Techniques (PACT'08)*. ACM Press, New York, 52–61.

WONG, H., PAPADOPOULOU, M.-M., SADOOGHI-ALVANDI, M., AND MOSHOVOS, A. 2010. Demystifying GPU microarchitecture through microbenchmarking. In *Proceedings of the IEEE Symposium on Performance Analysis of Systems and Software (ISPASS'10)*. 235–246.

YUAN, G. L., BAKHODA, A., AND AAMODT, T. M. 2009. Complexity effective memory access scheduling for many-core accelerator architectures. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO'09)*. ACM Press, New York, 34–44.

Received May 2011; revised April 2013; accepted June 2013