Adaptive Virtual Channel Partitioning for Network-on-Chip in Heterogeneous Architectures

JAEKYU LEE, SI LI, HYESOON KIM, and SUDHAKAR YALAMANCHILI,
Georgia Institute of Technology

48

Current heterogeneous chip-multiprocessors (CMPs) integrate a GPU architecture on a die. However, the heterogeneity of this architecture inevitably exerts different pressures on shared resource management due to differing characteristics of CPU and GPU cores. We consider how to efficiently share on-chip resources between cores within the heterogeneous system, in particular the on-chip network. Heterogeneous architectures use an on-chip interconnection network to access shared resources such as last-level cache tiles and memory controllers, and this type of on-chip network will have a significant impact on performance.

In this article, we propose a feedback-directed virtual channel partitioning (VCP) mechanism for on-chip routers to effectively share network bandwidth between CPU and GPU cores in a heterogeneous architecture. VCP dedicates a few virtual channels to CPU and GPU applications with separate injection queues. The proposed mechanism balances on-chip network bandwidth for applications running on CPU and GPU cores by adaptively choosing the best partitioning configuration. As a result, our mechanism improves system throughput by 15% over the baseline across 39 heterogeneous workloads.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Heterogeneous (hybrid) systems*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms: Design, Performance

Additional Key Words and Phrases: Heterogeneous architecture, on-chip network, quality-of-service

ACM Reference Format:
Lee, J., Li, S., Kim, H., and Yalamanchili, S. 2013. Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. ACM Trans. Des. Autom. Electron. Syst. 18, 4, Article 48 (October 2013), 28 pages.
DOI: http://dx.doi.org/10.1145/2504906

1. INTRODUCTION

An on-chip heterogeneous architecture that integrates GPU cores on top of conventional CPU-only chip multiprocessors (CMP) has become a popular architecture trend, as can be seen in Intel's Sandy Bridge [Intel Sandy Bridge] and Ivy Bridge [Intel Ivy Bridge], AMD's accelerated processing units (APU) [AMD 2011], and NVIDIA's Denver project [NVIDIA Denver]. In this architecture, various on-chip resources are shared between

We gratefully acknowledge the support of the National Science Foundation (NSF) CARRER CCF 1054830, CNS 0855110, and Sandia National Laboratories. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF or Sandia Lab.
Author's addresses: J. Lee, School of Computer Science, Georgia Institute of Technology; email: jaekyu.lee@cc.gatech.edu; H. Kim, School of Computer Science, Georgia Institute of Technology; S. Li and S. Yalamanchili, School of Electrical and Computer Engineering, Georgia Institute of Technology.

DOI: http://dx.doi.org/10.1145/2504906

CPU and GPU cores, such as last-level cache (LLC), on-chip interconnection networks, memory controllers, and DRAM memories.

This resource-sharing problem has existed since CMP was introduced. In CPU-GPU heterogeneous architectures, however, we expect more shared resource contention, especially interference suffered by CPU applications from GPGPU applications due to the different nature of CPU and GPU cores. CPU cores typically employ 1- to 4-ways of simultaneous multi-threading and rely on larger caches to tolerate memory access latencies. On the other hand, GPU cores operate with tens of active threads to minimize the penalty of the off-chip memory latency. The high degree of thread-level parallelism (TLP) in GPU cores leads to more frequent network injections, which only exacerbates the resource-sharing problem.

We tackle the resource-sharing problem of the on-chip network (NoC) in this article. Sources of interference can be located in any shared resources, from shared last-level caches (LLC) to memory controllers (MC). Nonetheless, the NoC is one of the most important shared mediums because it connects all components and all communication traverses through it. The management of the NoC significantly affects the performance of each application as well as the system throughput. The baseline on-chip routers favor applications with high network demands under the round-robin or oldest-first arbitration policies. Consequently, GPGPU applications are naturally favored and CPU applications face unfair network resource utilization in heterogeneous architectures.

To solve the resource-sharing problem in the NoC, researchers have proposed router arbitration policies [Das et al. 2009, 2010; Lee et al. 2008; Grot et al. 2009] in the homogeneous CMP domain. These policies consider different application characteristics and prioritize critical packets or applications. However, these mechanisms may not be used directly for heterogeneous architectures because they do not consider the heterogeneity of cores. GPU cores inject packets much more frequently than CPU cores because they are capable of running many concurrent threads with SIMD executions, which leads to an unbalanced number of packets between the CPU and GPU in the network. These characteristics of GPU cores increase the thread-level parallelism (TLP) of cores, thereby making them more tolerant to latency and bandwidth than CPU cores. Therefore, NoC mechanisms for heterogeneous architectures need to consider different characteristics of GPU cores to be effective.

Here, we propose a virtual channel partitioning (VCP) mechanism, which is simple yet effective, to attack resource-sharing problems in the NoC for heterogeneous systems. A router typically has multiple input and/or output virtual channels (VC) that share physical links and thus bandwidth. By dedicating a number of VCs to CPU and GPU applications, we can guarantee a minimum service in the network to each type. Also, VCP naturally arbitrates packets that pass through the router because VCP forces GPU packets to occupy only a part of VCs, thus CPU packets can find a VC immediately upon arrival. To provide CPU and GPU packets according to their VC availability, VCP requires separate injection queues for CPU and GPU packets. Injection queues of shared caches and memory controllers have both types of packets. If the shared queue with a first-come first-serve (FCFS) scheduler is used for the injection, even if available VCs of a certain type exist, packets cannot be injected until they arrive at the head of the queue. VCP uses DAMQ-based injection queues [Tamir and Frazier 1992] to maintain separate queues so that VCP can supply packets to their corresponding partition with low overhead.

However, VCP may result in significant performance degradation for bandwidth-limited GPGPU applications. Without partitioning, bandwidth-limited GPGPU applications could utilize more bandwidth, but their performance may be degraded because of the reduced bandwidth from partitioning. Therefore, the performance trade-off between CPU and GPGPU applications should be carefully balanced. Moreover, how different partitioning configurations affect performance varies by the workload

characteristics. Also, they behave differently even in the same workload if different phases exist. Therefore, partitioning should cope with the run-time behaviors of applications. This naturally leads us to study an adaptive partitioning mechanism. For better adaptation, our proposed feedback-directed VCP uses a sampling technique to dynamically compare different partitioning configurations and enforces the best performing configuration.

We claim our contributions to be as follows.

(1) We propose a feedback-directed virtual channel partitioning (VCP) mechanism that can arbitrate packets that pass through the local router while providing a more balanced number of packets to the network.
(2) VCP considers different characteristics of GPU cores by directly collecting performance metrics from the cores themselves, while coarse-grain control of virtual channels in VCP enables us to use simple hardware control structures.
(3) VCP improves system performance by 15% across 39 heterogeneous workloads. More importantly, results show at most a 2.5% performance degradation and only two workloads show negative speedup, while VCP performs better than any static configurations.

The rest of this article is organized as follows. Section 2 provides the background on our target architecture and on-chip router microarchitecture. Section 3 describes the motivation for our article. Section 4 introduces our proposal, VCP. We present the evaluation methodology and results in Sections 5 and 6. We discuss related work in Section 7, and Section 8 concludes the article.

2. BACKGROUND

2.1. On-Chip CPU-GPU Heterogeneous Architecture

As seen in recent processors [Intel Sandy Bridge; Ivy Bridge; AMD 2011; NVIDIA Denver], incorporating a GPU architecture into CMP has become an architectural trend. Although details may vary by vendor, many on-chip resources are shared between both processors, such as on-chip interconnection, last-level cache, and memory controllers. Although current on-chip GPUs are not as powerful as today's discrete GPUs such as NVIDIA's Fermi and AMD's Evergreen, more powerful GPUs will be integrated in future heterogeneous architectures, as reported in Intel's latest Haswell [Intel Haswell].

Modern high-performance CPU cores are typically superscalar out-of-order cores with a variety of speculative mechanisms. Many CPU applications are latency-sensitive, so large private caches (L1 and/or L2) are often employed to avoid long-latency access to off-chip memory. On the other hand, GPUs pack more processing elements in each core; a core is an in-order SIMD processor. Multiple threads execute the same instruction with different data sets per core. To tolerate memory latencies, GPU cores utilize massive multi-threading. When a thread is stalled due to the memory instruction, the execution is switched to other available threads. This latency-hiding mechanism of GPUs essentially generates a high volume of memory requests per unit time, thereby requesting massive bandwidth. When latency-sensitive CPU applications and massive-bandwidth GPU applications share on-chip resources, the bandwidth requirement and pressure from cores are different. Therefore, new heterogeneous architectures expose different aspects of problems, unlike conventional CMPs.

2.2. On-Chip Router Microarchitecture

This section provides a brief background on the on-chip router. A router has P input and P output ports; $P = 3$ for a bidirectional ring network and $P = 5$ for a 2D-mesh/torus. Each input port has one or more virtual channels (VC). Packets are inserted into one of

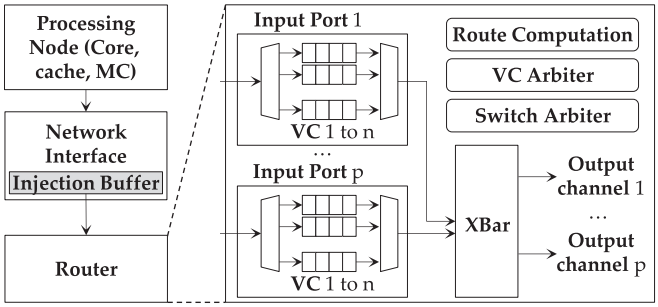


Fig. 1. Router microarchitecture.

the VCs from the processing nodes, specifically from the injection buffer in the network interface of a node. Then, packets go through the following pipeline stages to reach the next router.

(1) *Input buffering (IB)*. A packet received over a link of the source node is inserted into a VC.
(2) *Route computation (RC)*. The output port is determined.
(3) *Virtual channel allocation (VA)*. A packet acquires a downstream VC if it wins VC arbitration.
(4) *Switch allocation (SA)*. A packet acquires an exclusive grant to access the crossbar from its input buffer to the output port.
(5) *Switch traversal (ST)*. Once a packet acquires switch access, it can traverse to the output port over the crossbar.
(6) *Link traversal (LT)*. A packet is moved to the next router through the link.

Packets compete in multiple stages to acquire resources. To coordinate resource allocation between packets, a router has multiple arbiters, for example the VC and switch arbiters. Arbiters typically use very simple policies, such as round-robin or oldest-first arbitration policies. Figure 1 shows a microarchitecture of the on-chip router.

3. MOTIVATION

3.1. Interference Experienced by CPU Applications

The interapplication interference problem has existed even in homogeneous CMPs. However, as explained in Section 2.1, the problem becomes more complicated due to the heterogeneity of cores. Since GPU cores are capable of running more concurrent threads with SIMD executions, they will inject many more packets than CPUs. Since CPU and GPU packets have to compete with shared resources, hotspots will appear in shared caches and memory controllers. As a result, bandwidth-demanding GPGPU applications will interfere more with CPU applications than with homogeneous CMP workloads. Figure 2 shows the slowdown of CPU applications when they are running with a GPGPU application.[1]

In this figure, we run all combinations of one CPU and one GPGPU application in Table IV (i.e., 14 CPUs $\times$ 13 GPGPUs). Then, we measure the average and maximum slowdown by GPGPU applications for low and high CPU groups in Table IV.[2] We can observe that a significant performance degradation exists due to the interference by the GPGPU application, especially in more network-intensive benchmarks (high group).

[1]In Sections 3 and 4, we show data without prior description of the methodology. We use the same configuration for all data, but workloads may vary. The detailed methodology is given in Section 5.
[2]We measure the speedup by comparing the performance of a CPU application when it is running alone and with a GPGPU application.

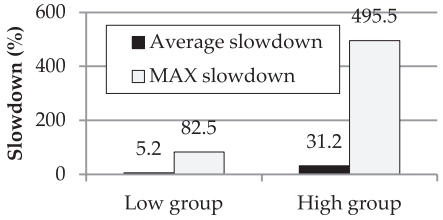


Fig. 2. Slowdown (%) of CPU applications (running alone vs. running with a GPGPU application).

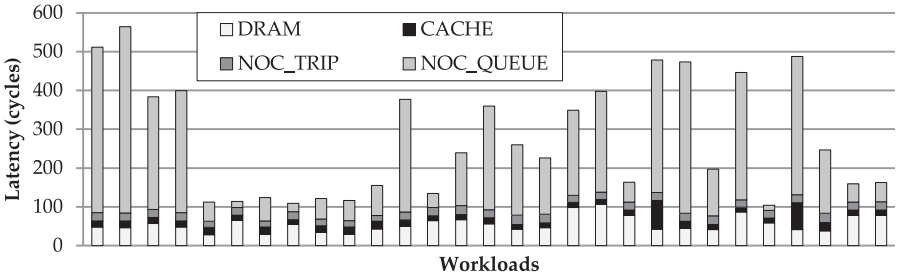


Fig. 3. Latency distribution of packets (x-axis: workloads).

3.2. Importance of NoC

All shared resources (last-level cache, interconnection network, memory controller and DRAM memory) are important and can be a source of inter-application interference. These resources are closely related to each other. For example, shared cache affects network pattern and the number of DRAM accesses. Off-chip DRAM accesses consume a significant amount of time, and different DRAM scheduling policies will affect cache hit ratio and network pattern. Finally, how the on-chip network is coordinated will change cache and DRAM access sequences. In this article, we consider *only* the interconnection network in heterogeneous systems to solve inter-application interference since it plays a very significant role in the system by connecting all components and governing access sequences in caches and DRAM controllers. Other than private cache accesses, all communications are made through the NoC, so memory traffic spends a significant amount of time in the NoC.

Figure 3 shows the latency distribution of packets of *W-HH* and *W-LH* workloads in Table V. We estimate latencies in the following categories.

—CACHE: cycles to access the LLC including delays in a queue
—DRAM: cycles in DRAM controllers to access off-chip DRAM
—NOC_QUEUE: queuing delay in the injection buffer
—NOC_TRIP: traverse time to reach a destination after injection into the network from the injection buffer

Although the DRAM waiting time accounts for a significant amount of time in some workloads, the NoC usually consumes most of the time, in particular due to queuing delays in shared routers (LLCs and memory controllers). We can expect that the time spent on the network will scale with a higher number of cores because of increased hop counts and traffic, so the importance of the NoC remains the same in the future.

3.3. Effectiveness of Previous Mechanisms in Heterogeneous Architectures

Many researchers studied various aspects of NoC, which is described in Section 7.1. However, there are few reasons why previous proposals may be less effective in

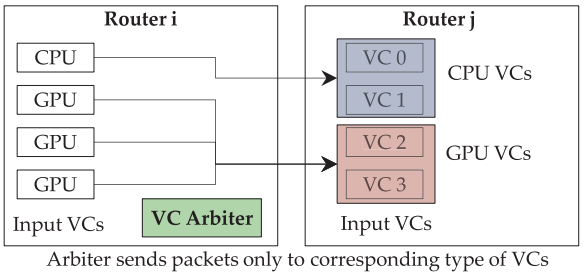


Fig. 4. Packet arbitration in VCP

CPU-GPU heterogeneous architectures compared to homogeneous systems due to GPU cores. First, most mechanisms consider only arbitrations of packets in a router. This is natural in homogeneous systems because we can expect that a similar number of packets from each application exists in the injection queues. However, due to bursty injections by GPUs, the occupancy of injection queues in shared resources is likely to be skewed in favor of GPU packets. Therefore, the effectiveness of previous mechanisms will be limited.

Previous mechanisms can be improved by having separate injection queues for a CPU and GPU or an out-of-order packet scheduler, but this will increase schedule complexity. The scheduler now needs to decide which queue (separate queues) or packet (out-of-order scheduler) to schedule. The decision made by the scheduler should be incorporated with arbitration decisions.

However, even if the previous mechanisms have separate injection queues, QoS for NoC needs to consider different characteristics of CPU and GPU cores. Since GPU cores can execute more concurrent threads, they have higher thread-level parallelism (TLP) and their ability to tolerate latency and bandwidth is different compared to CPU cores. As a result, the different nature of cores makes it difficult to use previous mechanisms.

Therefore, QoS mechanisms for heterogeneous architectures need to have separate queues/out-of-order packet schedulers and to take into consideration the nature of GPU cores to be more effective.

4. FEEDBACK-DIRECTED BANDWIDTH PARTITIONING

In this section, we describe the details of our proposed mechanism: a feedback-directed partitioning-based bandwidth control (VCP) for the NoC in heterogeneous architectures.

4.1. Virtual Channel Partitioning

To orchestrate bandwidth more effectively in heterogeneous systems, we propose a *virtual channel partitioning (VCP)* mechanism. VCP dedicates a number of VCs for CPU and GPU cores, similar to cache partitioning mechanisms. By splitting VCs, VCP naturally arbitrates packets that pass through the router (i.e., an intermediate node, not a destination). Since VCP forces GPU packets to occupy only a part of VCs, CPU packets can occupy a VC upon arrival. Therefore, CPU packets are prioritized as compared to unpartitioned VCs. Figure 4 shows the simplified VC arbitration. The VC arbiter is able to identify the type of available VC and tries to select a packet with the same type from input VCs.

To feed CPU and GPU packets based on the corresponding VC availability, VCP requires separate injection queues for CPU and GPU packets in the network interface. Even though dedicated VCs exist for CPU packets by VCP, until a CPU packet arrives at the head of the injection queue, none of the CPU packets can be injected into the

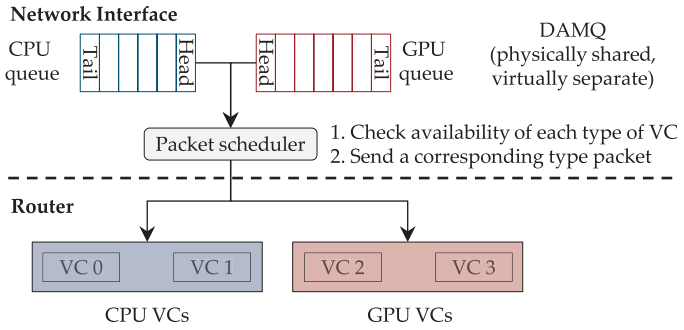


Fig. 5. Packet injection from the network interface.

NoC under the FCFS scheduling. Under the VCP, a packet can be sent to a router if an available CPU VC exists. To reduce the overhead of having two separate queues, VCP uses dynamically allocated multi-queue (DAMQ) buffers [Tamir and Frazier 1992]. DAMQ buffers can maintain separate virtual linked lists for each type of packet with very low overhead, so that the header packet of each type can easily be selected. Therefore, the injection scheduler needs to check only the availability of each VC type and send the corresponding type of VC from its queue; thus the scheduler is as simple as the baseline FCFS policy. Figure 5 shows the injection queue and the scheduler of the network interface. DAMQ enables a physically shared, but virtually separate, queue for CPU and GPU packets. The packet scheduler is simple FCFS, but it needs to check the availability of each type of VC, thereby sending a corresponding type of packet.

In this way, VCP can effectively arbitrate packets based on the partitioning configuration, while a more balanced number of packets are provided to the network from the injection queue. Moreover, VCP manages VCs in a very coarse-grain (CPU or GPU partition) manner, which makes the hardware very simple regardless of the number of cores.

4.1.1. Where Should VCP Be Applied? Virtual channel partitioning does not have to apply to routers that have only CPU or GPU packets since VCP aims to coordinate routers where CPU and GPU packets compete. However, in this case, we have to design routers with/without VCP, which may increase design cost. An alternative is for all routers to have enabled VCP by default, but VCP can be disabled by monitoring the number of CPU and GPU packets in a router.

4.2. VCP with Different Mixture of Workloads—Need For Adaptability

VCP can have N different partitioning configurations, where N is the number of VCs per port in a router. For example, if a router has six VCs per port, six partitioning configurations are possible: no-partitioning, 1:5 (1 CPU VC and 5 GPU VCs), 2:4, 3:3, 4:2, and 5:1. Since 0:6 or 6:0 configurations will only accept one type of packet, we exclude these configurations. Although VCP can effectively partition on-chip network bandwidth, the exact behavior will be affected by the partitioning configuration as well as by the mixture of workloads running on the heterogeneous system. For example, when CPU applications are running with non-network-intensive GPGPU applications, CPUs will not experience severe interference. Without partitioning, CPUs utilize more network bandwidth, but partitioning will decrease the bandwidth and degrade the performance of CPU applications. Therefore, partitioning should not be used in this case. On the other hand, with network-intensive GPGPU applications, partitioning must be applied to prevent interference. However, the ideal configuration (for example, 2:2 vs. 1:3 with 4 VCs) can vary for different workloads. To be effective with this type

Table I. The Length of Each Period in VCP

Length of initial period	500K cycles
Length of each training period	200K cycles
Length of main period	4M cycles

of workload, the performance trade-off between CPU and GPGPU applications should be carefully balanced. Also, the behavior will be affected by the run-time phase as well. Therefore, VCP needs to identify the ideal configuration and adapt run-time behavior.

4.3. Feedback-Directed VCP Using Sampling

To estimate the ideal partitioning configuration on heterogeneous workloads, we use a sampling technique for VCP, a feedback-directed VCP (F-VCP), as opposed to static VCP (S-VCP). We sample different VC partitioning configurations across periods and compare the performance of different configurations. F-VCP has the following sampling periods.

—*Initial warm-up.* To stabilize performance metrics, we idle and disable VCP during this period.
—*Training period.* To see the performance effect of each configuration, we maintain a configuration for a period. The training period consists of N sub-periods, each with a different configuration. For example, T_1 uses the baseline unpartitioned configuration. T_2 can be a 1CPU-3GPU partitioning configuration. Once a period is over, we collect the number of retired instructions from 1) all CPU applications and 2) a GPGPU application and calculate the speedup over the unpartitioned baseline (T_1) as in Equation (1).
—*Main period.* Once the training period is over, if the speedups of all configurations are less than 1, which means partitioning hurts performance, partitioning will be disabled for the main period. Otherwise, we choose the best performing configuration and apply it for the main period. Once a main period is over, T_1 of the training period will begin.

$$speedup_i = geomean\left(speedup^i_{CPU}, speedup^i_{GPU}\right) \tag{1}$$

$$speedup^i_{CPU} = \frac{\#inst\_retired^i_{ALL\_CPU}}{\#inst\_retired^{base(nopartition)}_{ALL\_CPU}} \tag{2}$$

$$speedup^i_{GPU} = \frac{\#inst\_retired^i_{GPU}}{\#inst\_retired^{base(nopartition).}_{GPU}}. \tag{3}$$

We set period lengths as in Table I from empirical data. With four VCs in the baseline, we test three partitioning configurations (unpartitioned, 1:3, and 2:2), so the training period resumes every 4.6 M cycles.

4.3.1. Central Decision Logic. To collect performance metrics from cores, VCP requires a central decision logic (CDL), which is located at the central node of the mesh. We use a similar approach in previous work [Das et al. 2009, 2010]. When a training period is over, CDL broadcasts to all cores a message that includes the configuration for the next period. Cores maintain the previous policy until they receive the message from CDL. Once they receive the message, they change the policy and send a performance metric during the last period to CDL. Once CDL collects messages from all cores, it will store the results. After all training periods are over, CDL decides the best configuration based on Equation (1) and broadcasts the decision to all routers. These processes may take up to a few hundred cycles. Since the length of periods is much longer, the overhead is not significant.

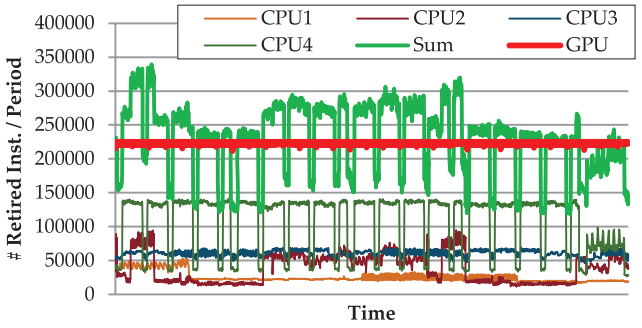


Fig. 6. Phases in a heterogeneous workload (Sum: sum of CPU retired instructions).

4.3.2. Why Sampling Works. This sampling mechanism can be viable since both CPU and GPGPU applications have shown a similar periodic progress in terms of the number of retired instructions. This is because 1) the GPGPU application is running in a single-program multiple-data (SPMD) manner. Although each thread has a different behavior at a time, this phase behavior becomes blurred by the other hundreds of concurrently running threads and 2) each CPU application has its own phase behavior. However, if we treat all CPU applications as a whole, the phase behavior of each application becomes unnoticeable as well. Figure 6 shows a phase example of a heterogeneous workload.

The GPGPU application (GPU line) shows similar progress across the entire duration. Although CPU applications show some fluctuations, the sum of all CPU applications (Sum line) maintains a similar progress for a sufficient time, so that our sampling mechanism can successfully differentiate the effect of different partitioning configurations.

4.3.3. Drawbacks and Improvements of Sampling. Sampling may have the following weaknesses: 1) effect of length of training period and 2) running non-optimal configurations during training periods.

First, for each training period, our F-VCP requires the collection of performance information as well as network injection information from cores. If the training period is too short, the overhead of communicating information will be too high. Also, we cannot acquire precise performance information due to a performance variation during a short duration. On the other hand, if the period is too long, we lose opportunities for improving performance of the system since F-VCP cannot adapt well to runtime behavior changes. To find the optimal period length, we have to identify the phase behavior and adjust the period length accordingly.

Second, during the training period, F-VCP will use N-1 non-optimal configurations, where N is the total number of configurations. However, N is very small in F-VCP since on-chip routers usually have very limited buffer space (only 4 to 6 VCs). Even if many VCs exist, since we observe that the overall system performance is linearly increasing, decreasing, or has its peak in the middle across consecutive configurations, we can employ existing *on-line training techniques* to reduce the overhead of having many non-optimal configurations. Moreover, we can *linearly lengthen the main period* if the same configuration is chosen consecutively after the training periods. If two consecutive decisions are different, then we reset the length of the main period to the original length.

In order to reduce the overhead of sampling, we can also consider on-demand sampling. As explained in Section 4.3.2, the performance of the system may show similar progress across periods. Once we find an optimal partitioning configuration through

the sampling periods, we can maintain this policy until the performance of the system changes (improves or degrades). Performance changes are due to the run-time phase change and indicate that the current configuration might not be optimal. However, this approach also has a drawback since it may cause a shorter main period and more communications when frequent phase changes exist.

4.4. Hardware Changes and Overhead

Our VCP requires the following hardware changes.

—Router arbiters should store VC partitioning configuration, which requires only a few bits, and be able to check the type of packets (CPU and GPU). Also, an arbitration algorithm should change such that dedicated VCs enforce that only the packet with the same type can acquire them. The arbiter with these changes is less complex than that of previous mechanisms since it only needs to match the type of packet with a VC.
—We have discussed the need for DAMQ [Tamir and Frazier 1992] in Section 4.1. The overhead of DAMQ is known to be insignificant.
—We have discussed the central decision logic in Section 4.3.1.

As a result, our VCP does not require significant changes to the baseline routers and the overhead is negligible.

4.5. Extension of VCP

VCP can be combined with other NoC mechanisms. Since the goal of VCP is to avoid significant interference by GPU cores, VCP only differentiates CPU or GPU packets. If we want to further differentiate individual applications, other mechanisms can be applied on top of VCP. For example, Aergia [Das et al. 2010] can set a different priority for each packet. Within the same VC partition (CPU or GPU), the arbiter can schedule packets based on their priority. We evaluate this VCP extension in Section 6.4.

VCP can also coordinate with cache management schemes, such as TAP [Lee and Kim 2012], which tries to find the best cache partitioning configuration between the CPU and GPU in heterogeneous architectures. Our VCP and TAP aim to solve a similar problem. Also, caches and NoCs are not independent and affect each other significantly. Therefore, combining VCP and TAP will yield even better results. However, managing one will affect the other, so combining two and studying their interactions are not trivial and is beyond the scope of our article.

4.6. Discussions

In this section, we discuss possible issues with VCP.

—*Deadlock* will not occur in VCP since CPU and GPU packets will occupy at least one VC and we use the oldest-first policy between the same type of packets.
—If applications always show unpredictable *phase changes*, sampling may misidentify the best performing configuration. Although we detect some workloads with phase changes, our observation is that if partitioning has a significant impact, it can overcome errors. Thus, the negative effect of dramatic phase changes is not as severe.
—No problem will occur during the *transition period* because the VC arbiter defines the allowed type and always searches all input VCs and matches the type.
—Although we consider only two types of heterogeneous cores (CPU and GPU) in the article, more complex heterogeneous systems exist. For example, most SoC (system-on-chip) architectures, including smartphones and tablets, have CPUs, GPUs, DSPs, and multiple modems, and all these components share the same system resources. Future multi- and many-core systems may also have several different types of

Table II. Processor Configuration

CPU	4 cores, 3.5GHz, 4-wide, out-of-order (OOO)
	gshare branch predictor
	8-way, 32KB L1 D/I cache, 2-cycle
	8-way 256KB L2 cache, 8-cycle
GPU	6 cores, 1.5GHz, in-order, 2-way 16 SIMD width
	8-way, 32KB L1 D (2 cycle), 4-way 4KB L1 I (1 cycle)
	16KB s/w managed cache
L3 Cache	4 tiles (each tile: 32-way, 2MB), 64B line, LRU
Memory Controller	DDR3-1333, 2 MCs (each 8 banks, 2 channels)
	41.6GB/s BW, 2KB row buffer, FR-FCFS scheduler

Table III. NoC Configuration

Frequency	1 GHz
Topology	4 × 4 2D Mesh
Pipeline	4-stage (IB, RC, VCA, SA/ST)
# VCs	4 per port, each VC can hold 5 flits * a packet can have at most 5 flits
# ports	5 per router
Link	128 bits (16 B) with 1-cycle latency
Routing	X-Y
Flow control	credit-based
Placement	Base in Figure 8

accelerators. In this case, we may need to add more VC types other than CPU and GPU VCs. However, considering the limited number of VCs, we may need to reduce the number of different VC types. We can achieve this by 1) forcing different types of processors to use the same type of VC or 2) letting some processors utilize any type of VCs. This decision should be made by identifying the characteristics of processors and applications running on them, but we do not discuss this further since this is beyond the scope of our article.

—Packets that carry performance metric information (from cores) and decision information (from CDL) are treated as special packets and they can utilize any VC type.

5. EVALUATION METHODOLOGY

5.1. Simulator

We use MacSim [HPArch Research Group 2011], a trace-driven and cycle-level heterogeneous architecture simulator, for evaluations. For all evaluations, we repeat early terminated applications until all applications have finished at least once. This is to model the resource contention uniformly across the duration of simulation, which is similar to the work in [Qureshi and Patt 2006; Xie and Loh 2009; Jaleel et al. 2010; Lee and Kim 2012]. Table II shows the processor configuration. To model a next-generation heterogeneous architecture, we model our baseline CPU similarly to Intel's Sandy Bridge [Intel Sandy Bridge], with high-end GPU cores that are similar to the SM (streaming multiprocessor) of NVIDIA Fermi [NVIDIA Fermi].

Table III shows the NoC configuration. Although we use a conservative five-stage pipeline model, we include the VCP result with a three-stage pipeline router model in Section 6.5. Also, the routers do not use any pipeline bypassing mechanisms, which can reduce latencies by skipping some pipeline stages when switches/links are idle. However, when operating in the regions where congestion dominates latency, bypassing provides minimal benefit. As explained, queuing delay, not trip delay, is

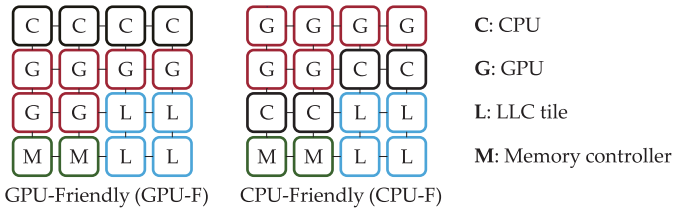


Fig. 7. Placement designs with the overlapped path.

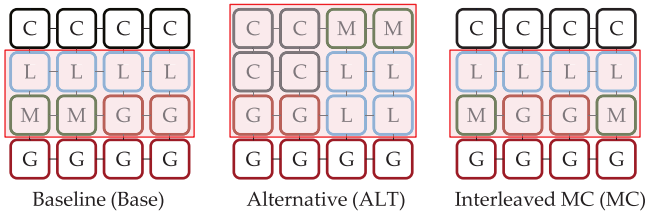


Fig. 8. Alternative placement designs (shaded area shows routers that may have both CPU and GPU packets).

dominant in our evaluated workloads, so the baseline router model performs similarly to the bypassing router.

5.2. Placement

In this section, we discuss the placement of components in the heterogeneous architecture. Several different methods to place CPU/GPU cores, LLC tiles, and memory controllers can exist. For example, Abts et al. [2009] discussed how to place memory controllers in a homogeneous mesh network. However, the placement of cores and other components is not discussed, to the best of our knowledge. In this article, identifying the best placement for a heterogeneous architecture is beyond the scope of our study. Instead, we discuss the basic placement ideas and reasoning for our designs.

First, placement must be carefully designed. For example, Figure 7 shows two designs where paths to memory routers (LLCs and MCs) are overlapped from CPU and GPU cores. The assumption here is that all communications between CPU and GPU cores are made only through caches. In these examples, intermediate nodes may suffer from much through traffic due to the overlapped path, which may lead to significant system performance degradations.

Figure 8 shows alternative designs that do not have overlapped paths. In all three alternatives, CPU and GPU cores have distinct routes to the memory routers while the placement of LLC tiles and memory controllers varies. The shaded area shows all routers that may have both CPU and GPU packets. Among these placements, we use Baseline (Base) placement in Figure 8 and we evaluate other placements in Figures 7 and 8 in Section 6.8.

5.3. Benchmarks

We use SPEC 2006 CPU benchmarks and CUDA GPGPU benchmarks from Nvidia CUDA SDK, Rodinia [Che et al. 2009], Parboil [The IMPACT Research Group, UIUC], and ERCBench [Chang et al. 2010]. For the CPU workloads, Pinpoint [Patil et al. 2004] was used to select a representative simulation region with the reference input set. Most GPGPU applications run until completion. Table IV shows the CPU and GPGPU benchmarks used for evaluations.

Table IV.
Benchmark characteristics based on the network-intensity (PKC is measured for an entire application. i.e. sum of core PKC).

	High (PKC > 20)		Low (PKC < 20)	
	Bench	PKC	Bench	PKC
CPU	GemsFDTD	58	povray	1
	wrf	63	gamess	2
	bwaves	69	namd	3
	cactusADM	73	sjeng	4
	milc	74	gobmk	6
	leslie3d	84	tonto	10
	lbm	90	perlbench	12
	High (PKC > 100)		Low (PKC < 100)	
	Bench	PKC	Bench	PKC
GPU	nearest-neighbor	166	Dxtc	0.4
	stencil	241	VolumeRender	3.0
	ScalarProd	253	cell	5.3
	bfs	304	raytracing	5.9
	cfd	331	AES	26
	Reduction	417		
	BlackScholes	437		
	SobolQRNG	452		

Table V. Heterogeneous Workloads

	# High type CPU	GPU type	#	Reference
W-LL	no more than 1	Low	10	6.1 only
W-HL	more than 2	Low	13	Entire Section 6
W-LH	no more than 1	High	13	
W-HH	more than 2	High	13	

We categorize benchmarks into two groups (high and low network-intensive) based on the packets per kilo cycles (PKC). We use this metric since PKC clearly shows the network intensity of an application. To distinguish low and high groups, we use a PKC of 20 and 100 for CPU and GPU applications, respectively.

Table V shows evaluated heterogeneous workloads. For all workloads, we run four CPU applications on four CPU cores along with one GPGPU application on six GPU cores. We categorize CPU workloads based on the number of high-type CPU applications (out of four). We choose each application pseudo-randomly.

5.4. Evaluation Metric

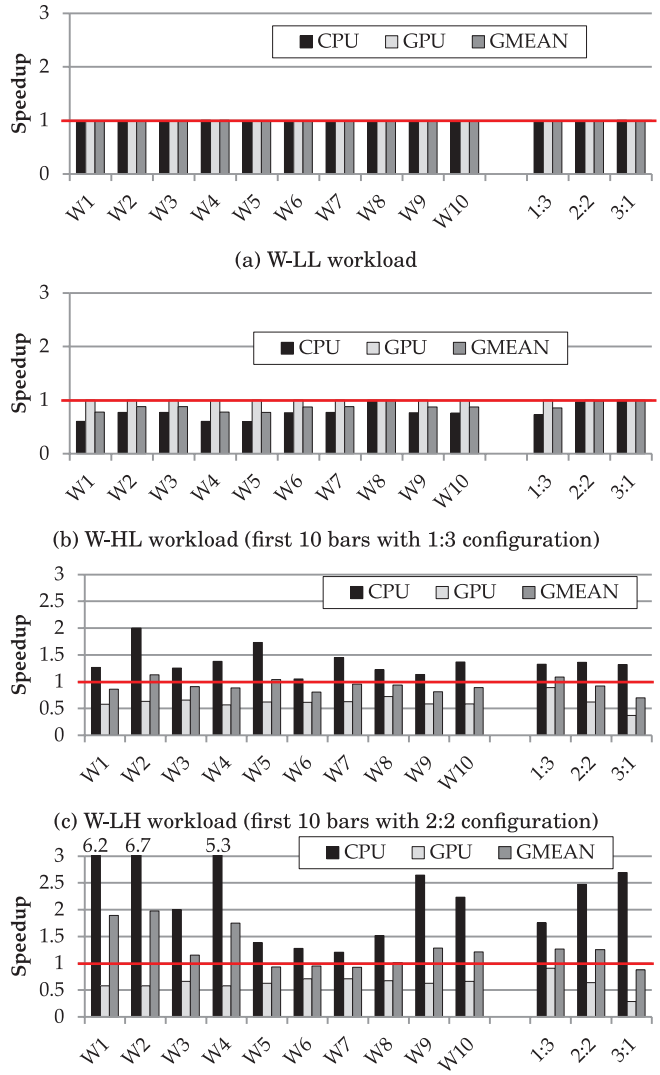
We use a speedup metric defined in Equation (4). First, we compute the speedup of each application with a configuration over the baseline unpartitioned configuration (Equation (6) for CPU and Equation (7) for GPGPU). Then, we calculate the average speedup of all CPU applications (Equation (5)). Finally, we take the average of Equation (5) and Equation (7).

$$speedup = geomean(speedup_{CPU}, speedup_{GPU}) \tag{4}$$

$$speedup_{CPU} = geomean(speedup_i, where\ 0 \leq i \leq 3) \tag{5}$$

$$speedup_i = IPC_i / IPC_i^{baseline(nopartition)} \tag{6}$$

$$speedup_{GPU} = IPC_{GPU} / IPC_{GPU}^{baseline(nopartition)} \tag{7}$$



(a) W-LL workload

(b) W-HL workload (first 10 bars with 1:3 configuration)

(c) W-LH workload (first 10 bars with 2:2 configuration)

(d) W-HH workload (first 10 bars with 2:2 configuration)

Fig. 9. Static VCP results.

Although we use a geometric speedup metric throughout the article, our mechanism is not limited by a specific metric. Since our target heterogeneous architecture is an emerging architecture, how to evaluate this architecture is debatable. Regardless, our VCP can easily adapt to any desirable metrics by replacing Equation (1) and Equation (4).

6. EVALUATION RESULTS

6.1. Static VCP Results

First, we show in Figure 9 the VCP results with static configurations (S-VCP) for different workloads. This indicates how VCP affects performance (detailed results of

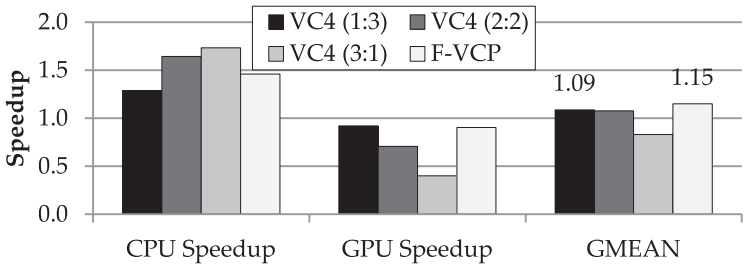


Fig. 10. Feedback-directed VCP results.

10 workloads and the average of all configurations). No significant difference exists across all configurations in *W-LL* workloads. Since CPU and GPGPU applications are not network-limited, performance is hardly affected by different configurations. For this reason, we exclude *W-LL* workloads in further evaluations.

For *W-HL* workloads, since CPU applications can utilize network bandwidth well without partitioning, VCP rather degrades the performance of CPU applications when only a small number of VCs are dedicated for them (1:3 configuration),[3] while the performance of the GPGPU application is not improved at all. As a result, the overall performance is degraded. On average, 1:3, 2:2, and 3:1 static partitioning show 15%, 2%, and 1% degradations, respectively. On the other hand, for *W-LH* workloads, although CPU applications are not network-limited, they experience moderate interference. As a result, dedicating one VC to CPU applications will be sufficient, but too many VCs will degrade the performance of GPGPUs severely. The 1:3 configuration shows an 8.5% improvement, while the 2:2 and 3:1 configurations show 9% and 30% degradations, respectively.

W-HH workloads show very complex behavior. The 2:2 and 1:3 configurations mostly show a benefit, but the always-winning configuration does not exist. Based on the workloads, the performance variance of the two configurations is significant. Dedicating more VCs for CPU applications will improve performance significantly, but providing more ways diminishes return. Meanwhile, the GPGPU application suffers severe degradation. We have to balance VC partitioning based on the workloads. Overall, 1:3 and 2:2 improve 25%, while 3:1 degrades 13%. The 2:2 configuration shows much better CPU performance, but the performance of the GPGPU application is significantly degraded.

6.2. Feedback-Directed VCP Results

Figure 10 shows the feedback-directed VCP (F-VCP) result along with static configuration results. As we give more VCs to CPU applications (1:3 to 3:1), the performance benefits of CPU applications diminishes, while performance degradation of the GPGPU application becomes more significant. F-VCP can identify the best static configuration with different workloads, so it can improve CPU applications significantly (46% improvement) while hurting GPGPU very little (9% degradation). F-VCP improves system performance by 15% on average, while the best static configuration (1:3) shows a 9% improvement.

We show the s-curve of F-VCP in Figure 11 for detailed analysis.[4] F-VCP mostly shows better results than the best of all static configurations. Moreover, across 39

[3]We use the *#CPU-VC*:*#GPU-VC* notation for static configurations. For example, 1:3 indicates one CPU VC and three GPU VCs.

[4]We sort workloads by the performance of F-VCP in ascending order.

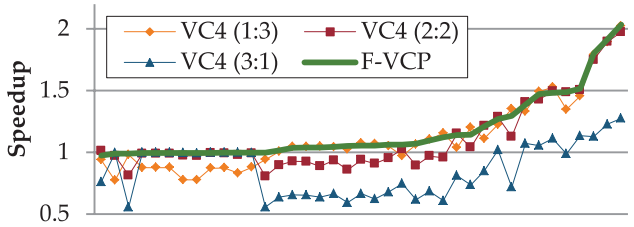


Fig. 11. F-VCP s-curve (workloads are sorted by the performance of F-VCP in ascending order).

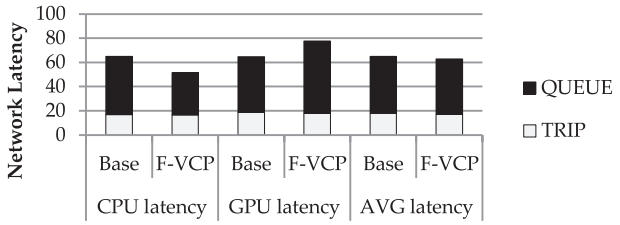


Fig. 12. Network latency changes with F-VCP.

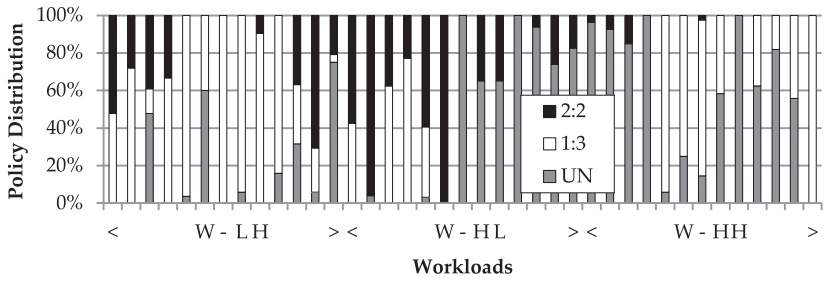


Fig. 13. F-VCP policy distribution (UN: unpartitioned).

workloads, the maximum performance degradation over the baseline is only 2.5% and only two workloads result in more than a 1% degradation.

Also, to show how F-VCP works, we show the average packet latency changes in Figure 12. We can observe that the traverse time is almost the same, but the queuing delay of CPU packets decreases significantly, while that of GPU packets increases.

Figure 13 shows a policy distribution histogram for each workload. Although F-VCP constantly chooses one configuration in some workloads, many workloads show that F-VCP adapts well to application phase changes if they exist.

6.3. Comparison with Different Packet Scheduling for Injection Buffer

As mentioned, an unbalanced number of packets between the CPU and GPU exists in the injection buffer when CPU and GPU applications share the network. In order to solve this imbalance, we can consider effective packet scheduling, which can be made through out-of-order scheduling. In the in-order injection buffer, a packet should wait until all previous packets are serviced. On the other hand, out-of-order scheduling can prioritize one packet over others. Thus, in this section, we evaluate several packet scheduling policies applied to the injection buffer. In addition, we also evaluate the DAMQ-based injection queue that has separate virtual queues for CPU and GPU packets, and F-VCP.

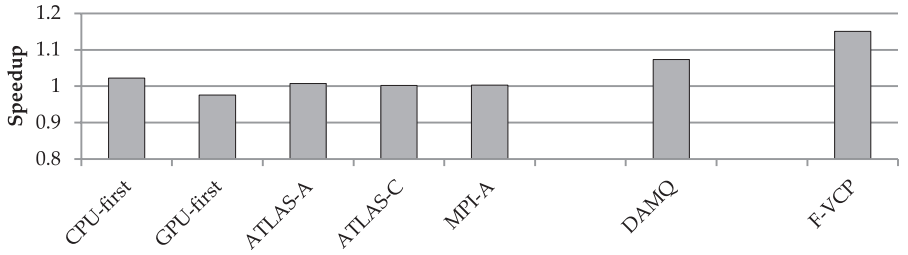


Fig. 14. Different injection buffer scheduling results.

We evaluate several packet scheduling policies applied to the injection buffer without VC partitioning. Also, we apply ATLAS [Kim et al. 2010a], one of the state-of-the-art memory schedulers, to the packet scheduler. Since ATLAS prioritizes applications that attained the least service during previous periods, ATLAS fits well to prioritize CPU packets that usually attain fewer services than GPU packets in heterogeneous workloads. We summarize all evaluated polices as follows.

—Baseline is first-come first-serve policy.
—CPU-first. CPU packets always have higher priority than GPU packets (batching is used for preventing starvation).
—GPU-first. GPU packets always have higher priority.
—ATLAS-A. ATLAS with application granularity.
—ATLAS-C. Similar to ATLAS-A, but we distinguish only two groups: GPGPU or CPU applications.
—MPI. Based on the private cache miss-per-instruction (MPI), we prioritize applications that have lower MPI.
—DAMQ. CPU and GPU packets have virtually separate queues using DAMQ. Scheduling between queues is round-robin.

Figure 14 shows the results. When out-of-order scheduling is applied to the shared injection buffer (other than `DAMQ` and `F-VCP`), CPU packets can be prioritized at a moment, but packet occupancy is still very unbalanced with the shared buffer. This limits the benefit of injection buffer scheduling. All evaluated policies show negligible benefits, but only the CPU-first policy shows a 2% improvement. On the other hand, by having separate queues with round-robin scheduling between them, we can mitigate the occupancy imbalance problem. As a result, `DAMQ` can improve performance by 8%. However, `DAMQ` cannot outperform F-VCP since the effect of coordination in the injection buffer is limited in the virtual channels of routers.

From the observations made in this section, we can draw the conclusion that separate queues are favorable to better performance in heterogeneous architectures, but the VC arbitration should be considered at the same time to be more effective, as in VCP.

6.4. Comparison with VC Arbitration Policies

In this section, we compare F-VCP with previous VC arbitration mechanisms, application-aware prioritization (denoted as STC) [Das et al. 2009], and Aergia [Das et al. 2010] along with two static policies, CPU-first and GPU-first.

STC computes the network demand of applications at intervals by looking at a number of metrics such as private cache misses per instruction, average outstanding L1 misses in MSHRs, and average stall cycles per packet. This produces a ranking of applications, and all packets of one application are prioritized over another, resulting in a coarse granularity of control. To prevent application starvation, a batching framework is implemented that prioritizes all packets of one time quantum over another,

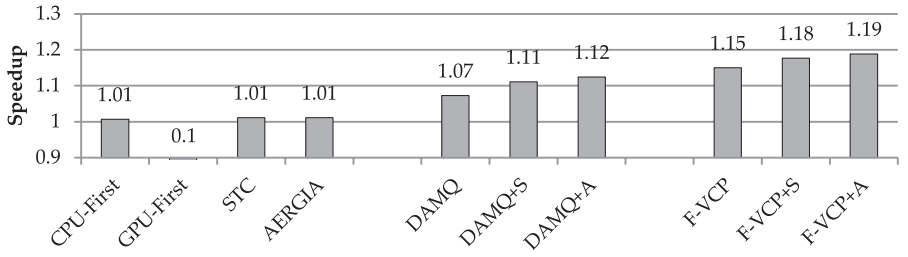


Fig. 15. Evaluation of virtual channel arbitration policies.

regardless of source application. Aergia predicts the available latency (slack) of any packet by the number of outstanding L1 misses and prioritizes low-slack (critical) packets over packets with higher slack when they are within the same batching interval. Static policies always give a higher priority to certain types of packets (either CPU or GPU) and form batches to prevent the starvation problem. Moreover, we apply STC and Aergia to the DAMQ-based injection buffer (DAMQ+S and DAMQ+A) and F-VCP (F-VCP+S and F-VCP+A). Figure 15 shows the results.

As explained in Section 3.3, NoC mechanisms for heterogeneous architectures should have separate injection queues for CPU and GPU packets. As a result, STC, Aergia, and CPU-first policies without separate injection queues show around a 1% improvement on average. In a few workloads, Aergia shows up to a 1.55 times speedup. These workloads have relatively high CPU-to-GPU packet ratio, so Aergia also can be effective. However, Aergia degrades the performance of almost half of the workloads (10% degradation at most). This is because fine-grain prioritization prevents some CPU applications from being prioritized, but Aergia mostly degrades GPGPU performance by not prioritizing them.

When `STC` and `Aergia` are applied along with separate injection queues (`DAMQ+S` and `DAMQ+A`), they can be more effective. Since separate injection queues provide a more balanced number of packets between CPU and GPU applications, router arbiters see a similar number of packets and are thereby effective. `STC` and `Aergia` provide 4% and 5% additional performance improvements on top of `DAMQ`.

On the other hand, our F-VCP successfully manages on-chip routers with almost no degradation cases. Moreover, as discussed in Section 4.5, VCP can be extended using previous VC arbitration mechanisms, such as STC and Aergia. We also evaluate these extensions, which are denoted `F-VCP+S` and `F-VCP+A` for STC and Aergia, respectively. Even though F-VCP already improves performance by 15%, STC and Aergia provide additional improvements of 3% and 4% by arbitrating packets in finer granularity.

6.5. VCP Results with Three-Stage Pipeline Model

As described in Section 5.1, we use a conservative five-stage pipeline model in all evaluations so far. This may incur extra latencies in the network. However, as claimed, a shorter-latency router model can improve performance, but it cannot entirely resolve the resource contention problem since the dominant delay occurs in the injection queues. In order to confirm that VCP works well with a faster router design, we reevaluate the same set of experiments as in Section 6.4 with a three-stage pipeline model; Figure 16 shows results.

Across 39 heterogeneous workloads, the three-stage pipeline model improves performance by 3% over the five-stage pipeline router. Interestingly, the benefit of the shorter-latency router model provides performance improvements for all configurations by 2% to 3%. Moreover, VCP yields an additional 16% performance improvement over the three-stage pipeline model. From this experiment, we can confirm that 1) the

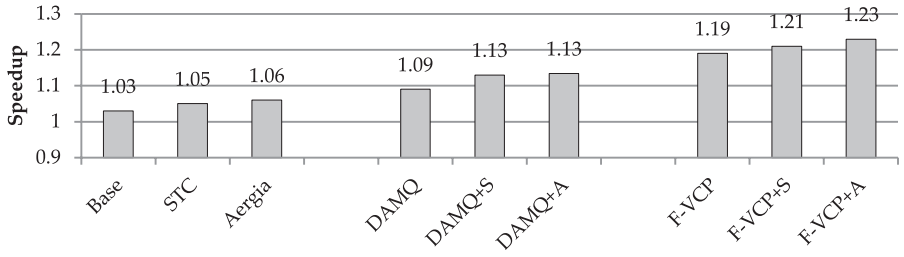


Fig. 16. Evaluation of three-stage pipeline router model (normalized to the router model with five-stage pipeline).

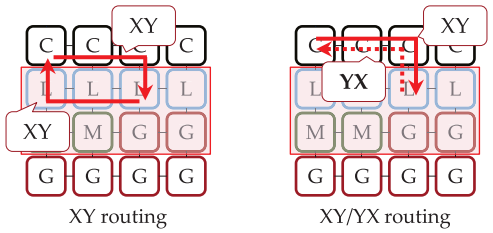


Fig. 17. Adaptive XY/YX routing.

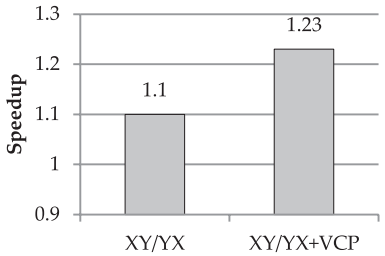


Fig. 18. Adaptive XY/YX routing results.

shorter-latency router model can improve the performance of the network as well as the system, 2) the network congestion still exists even with a shorter-latency model, and 3) VCP is still an effective solution.

6.6. XY/YX Adaptive Routing

In order to optimize the baseline network, we can consider adaptive routing as well. For example, in our baseline placement (Base in Figure 8), although memory routers (L3 and memory controllers) are shared, there are distinct routes from CPU and GPU cores to memory routers. When we use static XY routing only (Figure 17, left), packets must traverse within the shared memory routers. Instead, to reduce the contention in the memory routers, we can use XY/YX adaptive routing (Figure 17, right). When a core sends a request packet, it uses XY routing. When the packet is returned with data, it now uses YX routing. As a result, we can reduce the traversal within the memory routers.

Figure 18 shows the result of XY/YX adaptive routing along with VCP. As expected, XY/YX routing significantly improves performance by 10%. This is because XY/YX routing reduces the network congestion while improving the network utilization. We

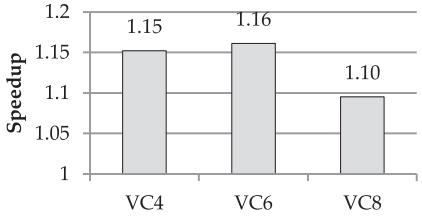


Fig. 19. F-VCP with different number of VCs.

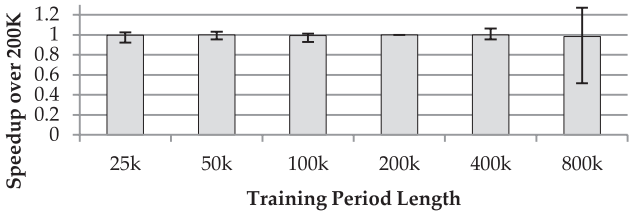


Fig. 20. F-VCP with different length of training period (base: 200K).

can also observe that VCP is still effective and gives an additional 13% performance improvements by mitigating network contentions to the memory routers.[5]

6.7. Sensitivity of VCP

In this section, we evaluate F-VCP with different configurations. Figure 19 shows the F-VCP results with a different number of VCs.[6] In each bar, we compare F-VCP with the baseline router with the same number of VCs (i.e., VC6-F-VCP with VC6). F-VCP performs well with more VCs, but the benefit decreases in VC8 and we expect diminishing improvement with more VCs. Generally, a higher number of VCs perform better by reducing the congestion, so the benefit of F-VCP can also decrease. However, the buffer space is limited in on-chip routers, which places a practical limit on the number of VCs. As a result, we expect that F-VCP will work well within this limitation.

We briefly discussed in Section 4.3 how different durations of training periods will affect F-VCP. We perform experiments with different durations of training periods. Figure 20 shows the results.[7] Generally, different durations of training periods would not matter on average, but the 800K configuration shows significant variances (from 0.53 to 1.28 speedup). A lengthy period can help reduce the overhead of sampling, but it may fail to adapt run-time behavior.

6.8. Different Placement Results

As discussed in Section 5.2, we evaluate different placements in this section. Figure 21 shows the results of placements in Figures 7 and 8. All results are normalized to the baseline (Base) placement in Figure 8.

Even though the designs in Figure 7 have the overlapped paths to the memory routers between CPU and GPU cores, CPU-Friendly and GPU-Friendly designs show 3% and 4% improvements over the baseline, respectively. The trip latency can increase, but dominant delays occur from the injection buffer. By having a shorter distance from

[5]Please note that this optimization is specific to our baseline placement and may not be effective on other configurations.

[6]We fix other configurations the same. Each VC has four buffer entries, so the number of total buffer entries is 4 * # VCs.

[7]We fix the length of the main period to be 20 times that of the training period.

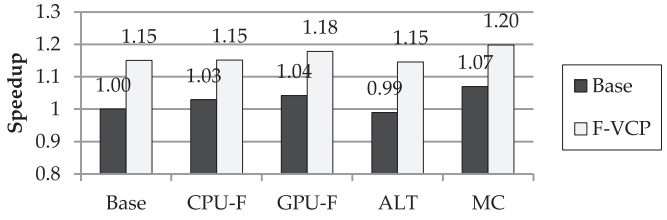


Fig. 21. Different placement evaluations.

memory to CPU (CPU-Friendly) or GPU (GPU-Friendly), queuing delays decrease. On the other hand, the design that distributes memory controllers shows the overall best performance (7%). As discussed in Abts et al. [2009], distributing congestion near memory controllers is a key reason for the improvements. With all different placement designs, VCP constantly shows higher than 11% improvement across all alternative designs. From this experiment, we conclude that different placements affect performance and VCP can control network bandwidth effectively when network congestion exists regardless of the placement.

6.9. Discussions

We discuss F-VCP with other possible configurations that we do not show in this section.

(1) Although we do not evaluate larger meshes with more cores, we expect that F-VCP will still be effective. As the size of the network increases, queuing delays can be reduced due to more diverse paths, but it can increase overall traffic from more cores and the average distance from source to destination will be increased. Also, we believe that the network contention problem will still exist even with a larger network, as more advanced applications that consume larger data will appear. As long as network congestion exists, F-VCP can successfully arbitrate between CPU and GPU packets.
(2) We have treated CPU and GPGPU applications with equal weight so far. When a user or a system wants to have a different weight for CPUs and GPUs, F-VCP requires a very minor change. It only requires changing the feedback metric, which is defined in Equation (1). Changes in the rest of the system are not necessary.

7. RELATED WORK

7.1. NoC Research

There has been an extensive amount of work in the past [Duato et al. 1997; Dally and Towles 2003] on non-on-chip networks. However, the time scales and the amount of resources available in non-on-chip network environments are much higher than what is permissible/acceptable in an NoC. Therefore, we limit our discussion only to NoC work.

A survey paper [Bjerregaard and Mahadevan 2006] and a keynote paper [Marculescu et al. 2009] laid out practical issues of implementing NoC, their solutions in the literature, and open problems in detail. In this section, we reiterate the work on QoS mechanisms among others and add recent work.

Previous QoS mechanisms can be categorized based on two aspects. First, based on whether a mechanism provides guaranteed services, we can categorize mechanisms into best-effort service (BE) and guaranteed service mechanisms (GS). While hard GS [Liang et al. 2000; Taylor et al. 2002; Millberg et al. 2004; Bjerregaard and Sparsø 2005; Goossens et al. 2005a; Weber et al. 2005; Hansson et al. 2009; Stefan et al. 2012] is favorable since it provides predictable outcomes within the tight requirement

such as real-time systems, BE [Goossens et al. 2002; Rijpkema et al. 2003] can better utilize system resources, thereby improving the system throughput. As a result, many researchers considered hybrid NoC, which combines GS and BE [Bolotin et al. 2004; Beigné et al. 2005; Dobkin et al. 2005].

Second, based on how QoS is provided, we can categorize previous mechanisms into (1) resource preallocation, (2) prioritization (arbitration), and (3) injection control (source throttling). In resource preallocation (or reservation) mechanisms [Liang et al. 2004; Millberg et al. 2004; Bjerregaard and Sparsø 2005; Goossens et al. 2005b; Leung and Tsui 2006], packets are assigned in different traffic classes based on the importance, and NoC resources, including virtual circuits, channels, and buffer space, are reserved for each class.

Priority-based mechanisms [Bolotin et al. 2004; Beigné et al. 2005; Harmanci et al. 2005; Marescaux and Corporaal 2007; Das et al. 2009, 2010] are similar to preallocation mechanisms since they determine a different priority for each application or packet (i.e., different class in preallocation) by estimating criticality from core/application-specific behaviors, including cache misses per instruction and number of miss-predecessors. However, they do not dedicate resources for a certain type and instead rely on arbitrations in various places in the network. In Das et al. [2009, 2010], priority is calculated in the centralized logic and each router has the same priority information for all applications. Arbiters of a router schedule packets based on the priority. To prevent the starvation problem, multiple packets often form a batch so that packets in old batches have higher priority than packets in newer batches.

On the other hand, injection (or congestion) control mechanisms [Nilsson et al. 2003; Duato et al. 2005; van den Brand et al. 2007; Ogras and Marculescu 2008] try to balance the injections from processing nodes or applications by injecting packets in the predefined rate or limiting packet injections. Globally synchronized frame (GSF) allows a limited number of packet injections for each source in one epoch (or frame) although GSF maintains future frames for handling bursty injections [Lee et al. 2008]. Each VC is now mapped to a different frame, and router arbiters prioritize packets in older frames. Therefore, GSF can guarantee minimum bandwidth as well as network delay. Grot et al. [2009] proposed a preemptive virtual clock (PVC), which uses a virtual clock to track each flow's bandwidth consumption while using frames to reduce the history effect of the virtual clock. PVC also uses the preemption of virtual channels for higher priority packets if lower priority packets occupy the VC so that the priority inversion problem does not occur. A recent proposal by Chang et al. [2012] controls injections from each application based on the MPKI (misses per kilo instructions) since MPKI can identify the memory intensity of the application.

Based on the classification, VCP is a basically BS mechanism. Also, VCP uses both resource preallocation and injection control categories since VCs are preallocated for CPU and GPU cores, and based on how we partition virtual channels, injection can be automatically controlled. As explained in Section 3.3, VCP has been improved in two aspects compared to previous mechanisms. First, as opposed to previous mechanisms that only considered run-time traffic analysis or targeted homogeneous processors, our work is intended for heterogeneous workloads on heterogeneous processors. To consider the different nature of CPU and GPU cores, in particular the effect of thread-level parallelism, VCP directly collects performance from cores instead of relying on indirect metrics. Second, VCP claims the need for separate injection queues for CPUs and GPUs. Since GPU packets mostly occupy the queue when a shared queue is used, the effect of any scheduling or arbitration is limited. In addition to these two improvements, one of the strengths of VCP is its extensibility with other mechanisms. As shown in Section 6.4, some previous mechanisms are orthogonal to VCP and can be combined with VCP to be more effective.

7.2. Virtual Channel Management Mechanism

In addition to NoC research in the previous section, we also discuss some of the adaptive virtual channel management mechanisms. Virtual channel size and organization can significantly affect system performance and power consumption [Varatkar and Marculescu 2002]. As a result, researchers have tried to find an optimal VC configuration in static or design time based on the characteristics of their target applications and traffic patterns. Also, dynamic buffer management mechanisms are proposed. Choi and Pinkston [2004] proposed dynamic VC allocation based on the traffic pattern using virtual channel DAMQs and DAMQs with recruit registers, which are improved DAMQs [Tamir and Frazier 1992]. Nicopolous et al. [2006] proposed ViChaR. The motivation of ViChaR is that the number of virtual channels and the depth of the buffer (based on the size of packet) can significantly affect the performance based on the traffic pattern. Thus, ViChaR optimizes the number of VCs and the depth of the buffer based on the traffic load using a unified buffer. Lai et al. [2008] also tried a similar dynamic VC allocation mechanism, but they considered congestion awareness. Evripidou et al. [2012] proposed a VC virtualization mechanism using VC renaming to support an arbitrarily large number of VCs. Trivinõ et al. [2012] also considered a VC virtualization mechanism as well as an NoC resource partitioning mechanism.

7.3. Shared Resource Partitioning in the System

To avoid severe interference and guarantee the quality-of-service (QoS) in shared resources, resource partitioning mechanisms have been widely studied by many researchers.

Shared last-level cache (LLC). Cache partitioning mechanisms [Suh et al. 2002, 2004; Kim et al. 2004; Qureshi and Patt 2006; Srikantaiah et al. 2009] have been most widely studied among others. In these mechanisms, cache ways are logically partitioned to each application, so that inter-application interference can be mitigated. Also, dynamic insertion policies [Qureshi et al. 2007; Jaleel et al. 2008, 2010] improve performance by providing adaptive cache insertions. Nontemporal or thrashing patterns are identified and then isolated by inserting cache blocks into a non-MRU position.

Memory controller. How the off-chip DRAM requests in memory controllers are managed significantly affects performance as well as fairness, so memory controllers are widely studied by many researchers. Most studies focus on DRAM request scheduling policies applied to the request buffer [Nesbit et al. 2006; Mutlu and Moscibroda 2007, 2008; Kim et al. 2010a, 2010b]. These mechanisms prioritize requests based on the behavior of applications, so that DRAM bandwidth is effectively shared by applications. On the other hand, Muralidhara et al. [2011] managed DRAM bandwidth by partitioning DRAM channels. In the proposed mechanism, DRAM requests from different applications are mapped to different DRAM channels.

7.4. Heterogeneous Architecture Research

The previous studies on shared resource partitioning in the system is discussed in Section 7.3. Here, we only discuss studies that specifically target CPU-GPU heterogeneous architectures. Lee and Kim [2012] studied the cache-sharing behaviors in heterogeneous workloads and proposed TLP-aware cache management schemes, which sample cores with different cache policies to see the performance effects by caches.

Yang et al. [2012] proposed a preexecution mechanism of GPGPU applications on CPU cores. The proposed mechanism automatically extracts memory operations of the GPGPU kernel and dispatches these operations on the CPU when the kernel is

launched. Preexecution from CPU cores brings data blocks of GPGPU kernels in the shared cache, so most off-chip accesses from GPGPU applications are hit in the cache.

Jeong et al. [2012] considered quality-of-service (QoS) in a multi-processor system-on-chip when off-chip bandwidth is shared between CPU and real-time constrained graphics applications. The proposed mechanism adaptively prioritizes CPU and GPU requests based on the progress made by graphics applications. Ausavarungnirun et al. [2012] proposed the staged memory scheduler (SMS). Due to massive memory accesses by GPU cores, the visibility of the memory requests by the memory scheduler is very limited. SMS attacks this problem with a multiple-stage memory scheduler. In the first stage, requests from the same source are inserted into the same queue and form a batch based on the row buffer locality. Then, a batch scheduler in the second stage picks an application batch based on the application characteristics and requirements. A scheduler in the final stage issues a ready DRAM command.

7.5. Heterogeneous Interconnection Network

We can consider a heterogeneous network to cope with the heterogeneity of cores, so we discuss previous work on heterogeneous on-chip networks in this section.

Mishra et al. [2011] proposed HeteroNoC, which asymmetrically allocates resources (buffers and links) to exploit non-uniform demand on a mesh topology. They used two types of routers, small and large, and placed more powerful large routers in congested areas. Grot et al. [2011] proposed Kilo-NOC, which isolates shared resources into QoS-enabled regions to minimize the network complexity. While proving QoS for shared resources, Kilo-NOC uses energy-efficient and cost-effective routers for the rest of the network. Bakhoda et al. [2010] proposed a throughput-effective NoC for GPU architectures. Due to many cores with a smaller number of memory controllers, a many-to-few traffic pattern is dominant in GPUs. To optimize such traffic, they used a half router, which cannot change the dimension of a packet, to reduce the complexity of the network while increasing the injection bandwidth from the memory controllers to provide burst data read.

7.6. NoC Research for GPU Architectures

In this section, we discuss NoC research proposed for GPU architectures. Yuan et al. [2009] proposed a complexity-effective memory scheduler for GPU architectures. NoC routers of the proposed mechanism reorder packets to increase row-buffer locality in the memory controllers. As a result, a simple in-order memory scheduler can perform similarly to a much more complex out-of-order scheduler. Bakhoda et al. [2010] proposed a throughput-effective NoC for GPU architectures. Due to many cores with a smaller number of memory controllers, a many-to-few traffic pattern is dominant in GPUs. To optimize such traffic, they used a half router, which cannot change the dimension of a packet, to reduce the complexity of the network while increasing the injection bandwidth from the memory controllers to provide burst data read.

8. CONCLUSION

How the NoC for heterogeneous architectures is handled has significant importance. Due to the heterogeneity of CPU and GPU cores, more specifically much higher network injections, CPU applications often suffer from severe interference. Previous mechanisms proposed for homogeneous CMPs have limitations to solve the network resource-sharing problem in this architecture. In this article, we propose feedback-directed virtual channel partitioning (F-VCP). On-chip network bandwidth can be controlled by the proposed VCP, which arbitrates packets that pass through the router while providing a more balanced number of packets to the NoC using DAMQ-based separate injection queues. Across 39 heterogeneous workloads, our VCP shows a 15%

improvement compared to the unpartitioned router. We perform thorough evaluations with many different configurations and VCP shows robustness. For future work, we will develop a performance model with on-chip bandwidth partitioning to improve the sampling-based technique in VCP.

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 31st Annual International Symposium on Computer Architecture*. ACM, New York, 451–461.

AMD. 2011. AMD Accelerated ProcessingUnits. http://www.amd.com/us/products/technologies/apu/Pages/apu.aspx.

AUSAVARUNGNIRUN, R., LOH, G., CHANG, K., SUBRAMANIAN, L., AND MUTLU, O. 2012. Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*. IEEE, 416–427.

BAKHODA, A., KIM, J., AND AAMODT, T. M. 2010. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of the 43rd International Symposium on Microarchitecture*. IEEE, 421–432.

BEIGNÉ, E., CLERMIDY, F., VIVET, P., CLOUARD, A., AND RENAUDIN, M. 2005. An asynchronous NOC architecture providing low latency service and its multi-level design framework. In *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'05)*. IEEE, 54–63.

BJERREGAARD, T. AND MAHADEVAN, S. 2006. A survey of research and practices of Network-on-chip. *ACM Comput. Surv. 38*, 1, Article 1.

BJERREGAARD, T. AND SPARSØ, J. 2005. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'05)*. IEEE, 1226–1231.

BOLOTIN, E., CIDON, I., GINOSAR, R., AND KOLODNY, A. 2004. QNoC: QoS architecture and design process for network on chip. *J. Syst. Archit. 50*, 2–3, 105–128.

CHANG, D. W., JENKINS, C. D., ET AL. 2010. ERCBench: An open-source benchmark suite for embedded and reconfigurable computing. In *Proceedings of the 20th International Conference on Field Programmable Logic and Applications (FPL'10)*. IEEE, 408–413.

CHANG, K. K.-W., AUSAVARUNGNIRUN, R., FALLIN, C., AND MUTLU, O. 2012. HAT: Heterogeneous adaptive throttling for on-chip networks. In *Proceedings of the 24th International Symposium on Computer Architecture and High Performance (SBAC-PAD'12)*. IEEE, 9–18.

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 International Symposium on Workload Characterization (IISWC'10)*. IEEE, 44–54.

CHOI, Y. AND PINKSTON., T. M. 2004. Evaluation of queue designs for true fully adaptive routers. *J. Parallel Distrib. Comput. 64*, 5, 606–616.

DALLY, W. AND TOWLES, B. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

DAS, R., MUTLU, O., MOSCIBRODA, T., AND DAS, C. R. 2009. Application-aware prioritization mechanisms for on-chip networks. In *Proceedings of the 42nd International Symposium on Microarchitecture*. ACM, New York, 280–291.

DAS, R., MUTLU, O., MOSCIBRODA, T., AND DAS, C. R. 2010. Aérgia: Exploiting packet latency slack in on-chip networks. In *Proceedings of the 32nd annual International Symposium on Computer Architecture*. ACM, New York, 106–116.

DOBKIN, R. (REUVEN), VISHNYAKOV, V., FRIEDMAN, E., AND GINOSAR, R. 2005. An asynchronous router for multiple service levels networks on chip. In *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'05)*. IEEE, 44–53.

DUATO, J., JOHNSON, I., FLICH, J., NAVEN, F., JAVIER, G. P., AND FRINÓS, T. N. 2005. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In *Proceedings of the 11st International Symposium on High Performance Computer Architecture*. IEEE, 108–119.

DUATO, J., YALAMANCHILI, S., AND NI, L. 1997. *Interconnection Networks: An Engineering Approach* 1st Ed. IEEE.

EVRIPIDOU, M., NICOPOULOS, C., SOTERIOU, V., AND KIM, J. 2012. Virtualizing virtual channels for increased network-on-chip robustness and upgradeability. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'12)*. IEEE, 21–26.

GOOSSENS, K., DIELISSEN, J., GANGWAL, O. P., PESTANA, S. G., RADULESCU, A., AND RIJPKEMA, E. 2005b. A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'05)*. IEEE, 1182–1187.

GOOSSENS, K., DIELISSEN, J., AND RADULESCU, A. 2005a. Æthereal network on chip: concepts, architectures, and implementations. *IEEE Des. Test Comput. 22*, 5, 414–421.

GOOSSENS, K., WIELAGE, P., PEETERS, A., AND VAN MEERBERGEN, J. 2002. Networks on silicon: Combining best-effort and guaranteed services. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'02)*. IEEE, 423–425.

GROT, B., HESTNESS, J., KECKLER, S., W., AND MUTLU, O. 2011. Kilo-NOC: A heterogeneous network-on-chip architecture for scalability and service guarantees. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture.* ACM, New York, 401–412.

GROT, B., KECKLER, S. W., AND MUTLU, O. 2009. Preemptive virtual clock: A flexible, efficient, and cost-effective QOS scheme for networks-on-chip. In *Proceedings of the 42nd International Symposium on Microarchitecture*. ACM, New York, 268–279.

HANSSON, A., SUBBURAMAN, M., AND GOOSSENS, K. 2009. aelite: A flit-synchronous network on chip with composable and predictable services. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'09)*. European Design and Automation Association, Leuven, Belgium, 250–255.

HARMANCI, M. D., ESCUDERO, N. P., LEBLEBICI, Y., AND IENNE, P. 2005. Quantitative modelling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip. In *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS'05,* Vol. 2. IEEE, 1782–1785.

HPARCH RESEARCH GROUP. 2011. MacSim. http://code.google.com/p/macsim/.

INTEL. HASWELL. http://www.intel.com/content/www/us/en/processors/core/4th-gen-core-processor-family.html.

INTEL. IVY BRIDGE. http://www.intel.com/content/www/us/en/silicon-innovations/intel-22nm-technology.html. Intel. Sandy Bridge. http://software.intel.com/en-us/articles/sandy-bridge/.

JALEEL, A., HASENPLAUGH, W., QURESHI, M., SEBOT, J., STEELY, S., JR., AND EMER, J. 2008. Adaptive insertion policies for managing shared caches. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT'08).* ACM, New York, 208–219.

JALEEL, A., THEOBALD, K. B., STEELY, S. C., JR., AND EMER, J. 2010. High performance cache replacement using re-reference interval prediction (RRIP). In *Proceedings of the 32nd annual International Symposium on Computer Architecture.* ACM, New York, 60–71.

JEONG, M. K., EREZ, M., SUDANTHI, C., AND PAVER, N. 2012. A QoS-aware memory controller for dynamically balancing GPU and CPU bandwidth use in an MPSoC. In *Proceedings of the 49th Annual Design Automation Conference (DAC'12)*. ACM, New York, 850–855.

KIM, S., CHANDRA, D., AND SOLIHIN, Y. 2004. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT'04)*. IEEE, 111–122.

KIM, Y., HAN, D., MUTLU, O., AND HARCHOL-BALTER, M. 2010a. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *Proceedings of the 16th International Symposium on High Performance Computer Architecture.* IEEE, 1–12.

KIM, Y., PAPAMICHAEL, M., MUTLU, O., AND HARCHOL-BALTER, M. 2010b. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *Proceedings of the 43rd International Symposium on Microarchitecture.* IEEE, 65–76.

LAI, M., WANG, Z., GAO, L., LU, H., AND DAI, K. 2008. A dynamically-allocated virtual channel architecture with congestion awareness for on-chip routers. In *Proceedings of the 45th annual Design Automation Conference (DAC'08).* ACM, New York, 630–633.

LEE, J. AND KIM, H. 2012. TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture. In *Proceedings of the 18th International Symposium on High Performance Computer Architecture.* IEEE, 91–102.

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 30th Annual International Symposium on Computer Architecture.* IEEE, 89–100.

LEUNG, L.-F. AND TSUI, C.-Y. 2006. Optimal link scheduling on improving best-effort and guaranteed services performance in network-on-chip systems. In *Proceedings of the 43rd Annual Design Automation Conference (DAC'06).* ACM, New York, 833–838.

LIANG, J., LAFFELY, A., SRINIVASAN, S., AND TESSIER, R. 2004. An architecture and compiler for scalable on-chip communication. *IEEE Trans. VLSI Syst. 12*, 7, 711–726.

LIANG, J., SWAMINATHAN, S., AND TESSIER, R. 2000. aSOC: A scalable, single-chip communications architecture. In *Proceedings of the 9th International Conference on Parallel Architectures and Compilation Techniques.* IEEE, 37–46.

MARCULESCU, R., OGRAS, U. Y., PEH, L.-S., JERGER, N. E., AND HOSKOTE, Y. 2009. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 28*, 1, 3–21.

MARESCAUX, T. AND CORPORAAL, H. 2007. Introducing the SuperGT network-on-chip; SuperGT QoS: More than just GT. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07).* ACM, New York, 116–121.

MILLBERG, M., NILSSON, E., THID, R., AND JANTSCH, A. 2004. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'04).* IEEE, 890–895.

MISHRA, A. K., VIJAYKRISHNAN, N., AND DAS, C. R. 2011. A case for heterogeneous on-chip interconnects for CMPs. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture.* ACM, New York, 389–400.

MURALIDHARA, S. P., SUBRAMANIAN, L., MUTLU, O., KANDEMIR, M. T., AND MOSCIBRODA, T. 2011. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *Proceedings of the 44th International Symposium on Microarchitecture.* ACM, 374–385.

MUTLU, O. AND MOSCIBRODA, T. 2007. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proceedings of the 40th International Symposium on Microarchitecture.* IEEE, 146–160.

MUTLU, O. AND MOSCIBRODA, T. 2008. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *Proceedings of the 30th Annual International Symposium on Computer Architecture.* IEEE, 63–74.

NESBIT, K. J., AGGARWAL, N., LAUDON, J., AND SMITH, J. E. 2006. Fair queuing memory systems. In *Proceedings of the 39th International Symposium on Microarchitecture.* IEEE, 208–222.

NICOPOULOS, C. A., PARK, D., KIM, J., VIJAYKRISHNAN, N., YOUSIF, M. S., AND DAS, C. R. 2006. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *Proceedings of the 39th International Symposium on Microarchitecture.* IEEE, 333–346.

NILSSON, E., MILLBERG, M., ÖBERG, J., AND JANTSCH, A. 2003. Load distribution with the proximity congestion awareness in a network on chip. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'03).* IEEE, 11126–11127.

NVIDIA. Fermi: NVIDIA's Next Generation CUDA Compute Architecture. http://www.nvidia.com/fermi.

NVIDIA. Project Denver. http://blogs.nvidia.com/2011/01/project-denver-processor-to-usher-in-new-era-ofcomputing/.

OGRAS, U. Y. AND MARCULESCU, R. 2008. Analysis and optimization of prediction-based flow control in networks-on-chip. *ACM Trans. Des. Autom. Electron Syst. 13*, 1, Article 11.

PATIL, H., COHN, R., CHARNEY, M., KAPOOR, R., SUN, A., AND KARUNANIDHI, A. 2004. Pinpointing representative portions of large Intel R Itanium R programs with dynamic instrumentation. In *Proceedings of the 37th International Symposium on Microarchitecture.* IEEE, 81–92.

QURESHI, M. K., JALEEL, A., PATT, Y. N., STEELY, S. C., AND EMER, J. 2007. Adaptive insertion policies for high performance caching. In *Proceedings of the 29th Annual International Symposium on Computer Architecture.* ACM, New York, 381–391.

QURESHI, M. K. AND PATT, Y. N. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th International Symposium on Microarchitecture.* IEEE, 423–432.

RIJPKEMA, E., GOOSSENS, K. G. W., RADULESCU, A., DIELISSEN, J., VAN MEERBERGEN, J., WIELAGE, P., AND WATERLANDER, E. 2003. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'03).* IEEE, 10350–10355.

SRIKANTAIAH, S., KANDEMIR, M., AND WANG, Q. 2009. SHARP control: Controlled shared cache management in chip multiprocessors. In *Proceedings of the 42nd International Symposium on Microarchitecture.* ACM, New York, 517–528.

STEFAN, R., MOLNOS, A., AND GOOSSENS, K. 2012. dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up. *IEEE Trans. Comput. 99,* PrePrints.

SUH, G. E., DEVADAS, S., AND RUDOLPH, L. 2002. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High Performance Computer Architecture.* IEEE, 117–128.

SUH, G. E., RUDOLPH, L., AND DEVADAS, S. 2004. Dynamic partitioning of shared cache memory. *J. Supercomputing 28*, 1, 7–26.

TAMIR, Y. AND FRAZIER, G. L. 1992. Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches. *IEEE Trans. Comput. 41*, 6, 725–737.

TAYLOR, M. B., KIM, J., ET AL. 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro 22*, 2, 25–35.

THE IMPACT RESEARCH GROUP, UIUC. Parboil Benchmark Suite. http://impact.crhc.illinois.edu/parboil.php.

TRIVIÑO, F., SÁNCHEZ, J. L., ALFARO, F. J., AND FLICH, J. 2012. Exploring NoC virtualization alternatives in CMPs. In *Proceedings of the 20th Euromicro International Conf. on Parallel, Distributed and Network-Based Processing (PDP'12)*. IEEE, 473–482.

VAN DEN BRAND, J. W., CIORDAS, C., GOOSSENS, K., AND BASTEN, T. 2007. Congestion-controlled best-effort communication for networks-on-chip. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'07)*. EDA Consortium, San Jose, CA, 948–953.

VARATKAR, G. AND MARCULESCU, R. 2002. Traffic analysis for on-chip networks design of multimedia applications. In *Proceedings of the 39th Annual Design Automation Conference (DAC'02)*. ACM, New York, 795–800.

WEBER, W.-D., CHOU, J., SWARBRICK, I., AND WINGARD, D. 2005. A quality-of-service mechanism for interconnection networks in system-on-chips. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'05)*. IEEE, 1232–1237.

XIE, Y. AND LOH, G. H. 2009. PIPP: promotion(insertion pseudo-partitioning of multi-core shared caches. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*. ACM, New York, 174–183.

YANG, Y., XIANG, P., MANTOR, M., AND ZHOU, H. 2012. CPU-assisted GPGPU on fused CPU-GPU architectures. In *Proceedings of the 18th International Symposium on High Performance Computer Architecture*. IEEE, 103–114.

YUAN, G. L., BAKHODA, A., AND AAMODT, T. M. 2009. Complexity effective memory access scheduling for many-core accelerator architectures. In *Proceedings of the 42nd International Symposium on Microarchitecture*. ACM, New York, 34–44.

Received January 2013; revised June 2013; accepted July 2013