A Heterogeneous Multiple Network-On-Chip Design:
An Application-Aware Approach

Asit K. Mishra
Intel Corporation
Hillsboro, OR 97124, USA
asit.k.mishra@intel.com

Onur Mutlu
Carnegie Mellon University
Pittsburgh, PA 15213, USA
onur@cmu.edu

Chita R. Das
The Pennsylvania State University
University Park, PA 16802, USA
das@cse.psu.edu

ABSTRACT

Current network-on-chip designs in chip-multiprocessors are agnostic to application requirements and hence are provisioned for the general case, leading to wasted energy and performance. We observe that applications can generally be classified as either *network bandwidth-sensitive* or *latency-sensitive*. We propose the use of two separate networks on chip, where one network is optimized for bandwidth and the other for latency, and the steering of applications to the appropriate network. We further observe that not all bandwidth (latency) sensitive applications are equally sensitive to network bandwidth (latency). Hence, within each network, we prioritize packets based on the relative sensitivity of the applications they belong to. We introduce two metrics, *network episode height and length*, as proxies to estimate bandwidth and latency sensitivity, to classify and rank applications. Our evaluations show that the resulting heterogeneous two-network design can provide significant energy savings and performance improvements across a variety of workloads compared to a single one-size-fits-all single network and homogeneous multiple networks.

Categories and Subject Descriptors

C.1.2 [**Computer Systems Organization**]: Multiprocessors; Interconnection architectures

Keywords

Heterogeneity, On-chip Networks, QoS, Packet Scheduling

1. INTRODUCTION

Network-on-Chips (NoCs) are envisioned to be a scalable communication substrate for building multicore systems, which are expected to execute a large number of different applications and threads concurrently to maximize system performance. A NoC is a critical shared resource among these concurrently-executing applications, significantly affecting each application's performance, system performance, and energy efficiency. Traditionally, NoCs have been designed in a monolithic, one-size-fits-all manner, agnostic to the needs of different access patterns and application characteristics. Two common solutions are to design a single NoC for (1) the common-case, or average-case, application behavior or (2) the near-worst case application behavior, by over-provisioning the design as much as possible to maximize network bandwidth and to minimize network latency. However, applications have widely different demands from the network, e.g., some require low latency, some high bandwidth, some both, and some neither. As a result, both design choices are suboptimal in either performance or energy efficiency. The "average-case" network design cannot provide good performance for applications that require more than

the supported bandwidth or that benefit from lower latency. Both network designs, especially the "over-provisioned" design, are power- and energy-inefficient for applications that do not need high bandwidth or low latency. Hence, monolithic, one-size-fits-all NoC designs are suboptimal from performance and energy standpoints.

Ideally, we would like a NoC design that can provide just the right amount of bandwidth and latency for an application such that the application's performance requirements are satisfied (or its performance maximized), while the system's energy consumption is minimized. This can be achieved by dedicating each application its own NoC that is dynamically customized for the application's bandwidth and latency requirements. Unfortunately, such a design would not only be very costly in terms of die area, but also requires innovations to dynamically change the network bandwidth and latency across a wide range. Instead, if we can categorize applications into a *small* number of classes based on similarity in resource requirements, and design multiple networks that can efficiently execute each class of applications, then we can potentially have a cost-efficient network design that can adapt itself to application requirements.

Building upon this insight and drawing inspiration from the embedded and ASIC designs where a single network is customized for *a single* application, this paper proposes a new approach to designing an on-chip interconnect that can satisfy the *diverse* performance requirements of general-purpose applications in an energy-efficient manner. We observe that applications can be divided into two general classes in terms of their requirements from the network: bandwidth-sensitive and latency-sensitive. Two different NoC designs, each of which is customized for high bandwidth or low latency, can, respectively, satisfy requirements of the two classes in a more power-efficient manner than a monolithic single network. We, therefore, propose designing two separate heterogeneous networks on a chip, dynamically monitoring executing applications' bandwidth and latency sensitivity, and steering/injecting network packets of each application to the appropriate network based on whether the application is deemed to be bandwidth-sensitive or latency-sensitive. We show that such a heterogeneous design can achieve better performance and energy efficiency than current average-case one-size-fits-all NoC designs.

To this end, based on extensive application profiling, we first show that a high-bandwidth, low-frequency network is best suited for bandwidth-sensitive applications and a low-latency, high-frequency network is best for latency-sensitive applications. Next, to steer packets into a particular network, we identify a packet's sensitivity to network latency or bandwidth. For this, we propose a new packet classification scheme that is based on an application's intrinsic network requirements. We introduce two new metrics, *network episode length* and *height*, to dynamically identify the communication requirements (latency and bandwidth sensitivity) of applications. Observing that not all applications are equally sensitive to latency or bandwidth, we propose a fine-grained prioritization mechanism for applications within the bandwidth and latency optimized networks. Thus, our mechanism consists of first *dynamically classifying* an application as latency or bandwidth sensitive, then *steering* it into the appropriate network and, finally within each network *prioritizing* each application's packets based on its relative potential to improve overall system performance and reduce energy.

Our evaluations on a 64-core 2D mesh architecture, considering 9 design alternatives with 36 diverse applications, show that our heterogeneous two-network NoC design consisting of a 64b link-width latency-
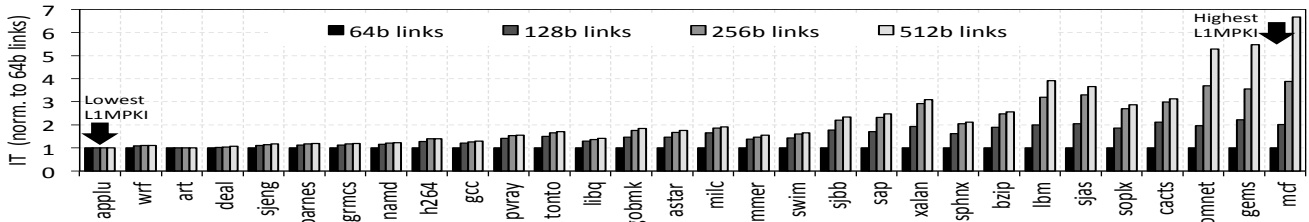
Figure 1: Instruction throughput (IT) scaling of applications with increase in network bandwidth.
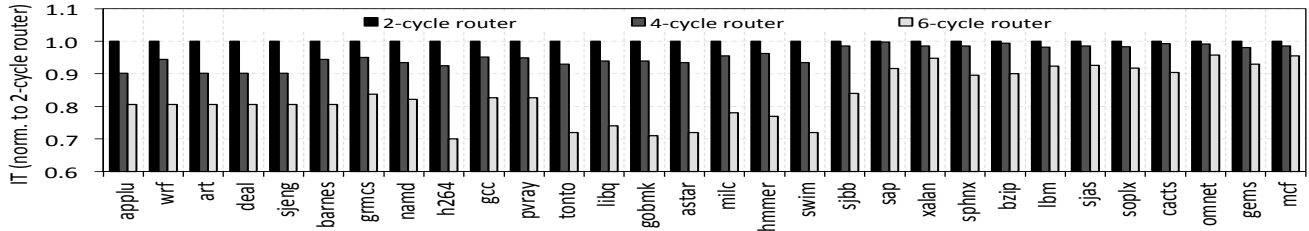

Figure 2: Instruction throughput (IT) scaling of applications with increase in router latency.

optimized network and a 256b link-width bandwidth-optimized network, provides 5%/3% weighted/instruction throughput improvement and 31% energy reduction over an iso-resource (320b link-width) monolithic network design. When compared to a baseline 256b link-width monolithic network, our proposed design provides 18%/12% weighted/ instruction throughput improvement and 16% energy reduction.

2. COMMUNICATION CHARACTERIZATION

We provide observations that highlight the intrinsic heterogeneity in network demand across applications. These observations form the motivation for an application-aware NoC design. We start by looking at two *fundamental* parameters: network channel bandwidth and latency.

Impact of channel bandwidth on performance scaling: Channel or link bandwidth is a critical design parameter that affects network latency, throughput and energy/power. To study the sensitivity of an application to variation in link bandwidth, we use a 64-core chip-multiprocessor (CMP) on an 8x8 mesh network, where both cores and network clocked at the same frequency, and run a copy of the same application on all nodes on the network. Table 1 shows the system configuration. We chose applications from commercial, SPEC CPU2006, SPLASH and SPEC OMP suites. We analyze scenarios where we double the bandwidth starting with 64-bit links up to 512-bit links. Figure 1 shows the results of this analysis for 30 of the 36 applications in our benchmark suite (6 non-network-sensitive applications are omitted to reduce clutter in the plots). In this figure, the applications are shown on the X-axis in order of their increasing L1MPKI (L1 misses per 1000 instructions). The Y-axis shows the average instruction throughput when normalized to the instruction throughput of the 64b network.

Observations from this analysis are: (1) Of the 30 applications shown, performance of 12 applications (the rightmost 12 in the figure after swim) scales with increase in channel bandwidth. For these applications, an 8X increase in bandwidth results in at least a 2X increase in performance. We call these applications *bandwidth-sensitive* applications. (2) The other 18 applications (all applications to the left of and including swim), show very little to no performance improvement with increase in network bandwidth. (3) Even for bandwidth-sensitive applications, not all applications' performance scales equally with increase in bandwidth. For example, while omnet, gems and mcf show more than 5X performance improvement for 8X bandwidth increase, applications like xalan, soplex and cacts show only 3X improvement for the same bandwidth increase. (4) L1MPKI is not necessarily a good predictor of bandwidth-sensitivity of applications. Intuitively, applications that have high L1MPKI would inject more packets into the network, and hence would benefit more from a higher-bandwidth network. But this intuition does not hold entirely true. For instance, bzip, despite having a higher L1MPKI than xalan, is less performance-sensitive to bandwidth than xalan. Thus, we need a better metric to identify bandwidth-sensitive applications.

Impact of network latency on performance scaling: Next, we analyze the impact of network/router latency on the instruction throughput of these applications. For this, we add an extra pipeline latency of 2 and 4 cycles to each router (in the form of dummy pipeline stages) on top of the baseline router's 2-cycle latency. The cores and the network are clocked at the same frequency for this analysis. Increasing the pipeline stages at each router *mimics* additional contention in the routers when compared to the baseline network. Figure 2 shows the results for this analysis, where the channel bandwidth is 128b (although the observation from this analysis holds true for other channel bandwidths as well).

Our observations are the following: (1) Bandwidth-sensitive applications (the rightmost 12 applications) are not very responsive to increase in network/router latency. On average, for a 3X increase in per-hop latency, there is only 7% degradation in application performance (instruction throughput) for these applications, i.e., an extra 4 cycle latency per router is tolerated by these applications. (2) On the other hand, for all applications to the left of and including swim, there is about 25% performance degradation when the router latency increases from 2-cycles to 6-cycles. We call these *latency-sensitive* applications. (3) L1MPKI is *not* a perfect indicator of latency-sensitivity (hmmer, despite having a higher L1MPKI than h264, does not show proportional performance improvement with reduction in router latency).

Application-level implications on network design: The above analysis suggests that a single monolithic network is not the best option for various application demands. Therefore, an alternative approach to designing an on-chip interconnect is to have multiple networks, each of which is specialized for common application requirements, and dynamically steer requests of each application to the network that matches the application's requirements. Based on Figures 1 and 2, a wide and low-frequency network is suitable for bandwidth-sensitive applications, while a narrow and high-frequency network suitable for latency-sensitive ones. To improve the performance of the latency-sensitive applications, a network architect can reduce the router pipeline latency from 2-cycles (our baseline) to a single cycle, while keeping the frequency constant, or increase the network frequency (to reduce network latency). Although there are proposals that advocate for single-cycle routers [10, 13, 12, 6], such designs often involve speculation, which increases complexity and can be ineffective at high or adverse load conditions, and require relatively sophisticated arbiters that are not necessarily energy efficient. Hence, while single-cycle routers are feasible, in this paper, we use frequency as a knob to reduce the network latency. According to our analysis, increasing the frequency of the network from 1 GHz to 3 GHz (3 times the core frequency) leads to less than 1.5% increase in *energy* for latency-sensitive applications (results for energy with frequency scaling are omitted for brevity).

Designing latency- and bandwidth-customized networks is the first step in achieving customization in the network. We also need a runtime mechanism to classify applications into one of the two categories: *latency or bandwidth sensitive*. In addition, since not all applications are equally sensitive to bandwidth or latency, we would like a mechanism that ranks the applications within each category in a more fine-grained manner within the latency- and bandwidth-customized networks. The next section discusses how we perform application classification.

3. DYNAMIC CLASSIFICATION OF APPLICATIONS

The goal of dynamically identifying an application's sensitivity to latency or bandwidth is to enable the local network interface (NI)
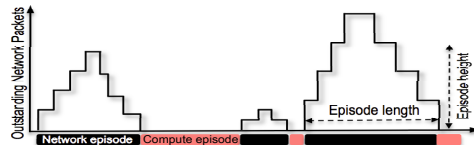

Figure 3: Network and compute episodes.

at each router to steer packets into a network that has been optimized for either latency or bandwidth. We propose two new metrics, called *network episode length* and *episode height*, that effectively capture the network latency and bandwidth demands of an application.

Episode length and height: During its life cycle, an application alternates between two kinds of episodes (shown in Figure 3): (1) *network episode*, where the application has at least one packet (to L2 cache or to DRAM) in the network, and (2) *compute episode*, where there are no outstanding cache/memory requests by the thread. During the network phase, there may be multiple outstanding packets from the application in the network owing to various techniques that exploit memory-level parallelism (MLP) [5, 14]. During this network phase, the core is likely stalled, waiting for L2 and memory requests to be serviced. Because of this, the instruction throughput of the core is low. During the compute episode, however, the instruction throughput is high [9]. We characterize a network episode by its length and height. *Length* is the number of cycles the episode lasts starting from when the first packet is injected into the network until there are no more outstanding packets belonging to that episode. *Height* is the average number of packets (L1 misses) injected by the application during the network episode. To compute height, the core hosting the application keeps track of the number of outstanding L1 misses (when there is at least 1 L1 miss) in the re-order buffer on a per-cycle basis.

A short episode height suggests that the application has low MLP. In other words, the latencies of the requests are not overlapped, and as a result the application's progress is sensitive to network latency. The application also does not require significant bandwidth because it has a small number of outstanding requests at any given time. On the other hand, a tall episode height suggests that the application has a large number of requests in the network, and the network latency of the packets are overlapped. This indicates that the application likely needs significant bandwidth from the network to make progress while its progress is less sensitive to network latency. Hence, we use the network episode height of an application as the main indicator of latency or bandwidth sensitivity of an application.

A short episode length (on average) suggests that the application is less network intensive. Contrast this with an application that has a long episode length, i.e., more network intensive. An equal amount of network delay would slow down the former application more than it does the latter application. As a result, the former application with a short average episode length is likely to be more sensitive to network latency, and therefore would likely benefit from being prioritized over another application with a longer average episode length. Note that this observation is similar to Kim et al.'s for memory controllers [9], which showed that prioritizing an application with shorter memory episodes over a one with longer memory episodes is likely to improve performance.

We compute running averages of the episode height and length to keep track of these metrics at runtime. We quantize episode height as *tall*, *medium* or *short* and episode length as *long*, *medium* and *short*. This allows us to perform fine-grained dynamic application (or, application phase) classification based on episode length and height. Figures 4 and 5 show these metrics for 30 applications in our benchmark suite. Note that, these figures show the *average* metrics for an entire application and that there are intra-application latency/bandwidth sensitive phases that our dynamic scheme (Section 4) captures. Based on Figures 1 and 2, we classify all applications whose average episode length and height are shorter than sjbb's episode length and height, respectively, to be short in length and height (shaded black in the figures). Applications whose average episode height is larger than sjbb's episode height but lower than 7 (empirically chosen) are classified as having medium (shaded blue in the figures) and the remaining as having tall episode heights (shaded with hatches in Figure 4). Empirically, a cut-off of 10K cycles is chosen to classify applications as having medium vs. long episode length.

Classification and ranking of applications: Figure 6 (left) shows the classification of applications based on their episode height and length. The figure also shows the bandwidth-sensitive and latency-sensitive applications based on such a classification. We use this fine-grained classification to steer applications to the two networks as well as rank applications for prioritization within a network.

Episode height, as a general principle, is used to identify the bandwidth/latency sensitivity of applications. Applications with tall height are considered bandwidth sensitive (due to high MLP, which leads to high bandwidth demand as well as high latency tolerance), and steered to the bandwidth-optimized network. Applications with low height are latency sensitive (due to low MLP, which leads to low latency tolerance for each packet) and are steered to the latency-optimized network. Figure 6 shows the resulting classification of applications. Note that applications whose episode height is medium but episode length is short are classified as latency-sensitive as they have neither short nor tall episode heights, but demand less from the network due to short episodes.

Episode length, for the most part, is used to determine the *ranking of applications within each network*. The general principle is to prioritize applications with shorter network episode length over others because delaying such applications in the network has much more of an effect on their performance (slowdown) than delaying applications with long episode lengths (since the latter set of applications are already slow due to long network episodes anyway). Figure 6 (right) shows the resulting ranking of the applications in their respective networks based on relative episode length and height.

4. DESIGN DETAILS

Since we use a canonical 2D network in our study, instead of discussing the standard router and network designs, we focus on the design aspects for supporting our classification and prioritization schemes in this section.

Computing episode characteristics: To filter out short-term fluctuations in episode height/length, and adapt our techniques to handle long-term traffic characteristics, we use running averages of the metrics. On every L1 miss, the NI computes the running average of episode height/length. To compute episode height, the outstanding L1 miss count is obtained from the miss-status handling registers (MSHRs). Counting the number of cycles (using an M-bit counter) the L1 MSHRs are occupied gives the information to compute episode length. This M-bit counter is reset every batching interval, B. We use the notion of batching [3, 15] to prevent starvation due to ranking of applications in each network, as done in [3, 15, 2] (more information on this is in Section S.3).

When the NI of a router receives a packet, it: (1) updates the episode height/length metric for the current application phase and (2) decides which network this packet is to be steered to, based on the classification scheme (Section 3). Thus, the episode metrics are computed per phase of an application. All packets belonging to a particular phase are steered into the network optimized for either latency or bandwidth. Note that each application's rank and network sensitivity are decided at runtime (although the classification analysis we provided in Section 3 was for the whole application). This helps our scheme to capture within-application variation in latency and bandwidth sensitivity. No central coordination is required in our technique to decide a uniform central ranking across all the applications in the system, which was needed in past works that ranked applications for prioritization [3, 2]. Once a packet's rank is decided, it is consistently prioritized with that rank across the entire network until it reaches its destination.

The NI tags the transmitted packet with its rank (2-bits) and its batch-id (3-bits). At each router, the priority bits in the header-flit are utilized by the priority arbiters in a router to allocate the virtual channels (VCs) and the switch. To prevent priority inversion due to VCs in routers, where a packet belonging to an older batch or higher rank is queued behind a lower ranked packet, we use atomic buffers [16].

Customized network design choices: As mentioned earlier, we opt for a high-frequency but low link-width network for the latency-sensitive applications and a high-bandwidth network operating at the same frequency as the cores for the bandwidth-sensitive applications. We use a 2-stage baseline router and increase the router frequency up to 3 times for the latency-sensitive network. High-frequency routers can be designed by a combination of both micro-architecture and circuit optimizations as shown
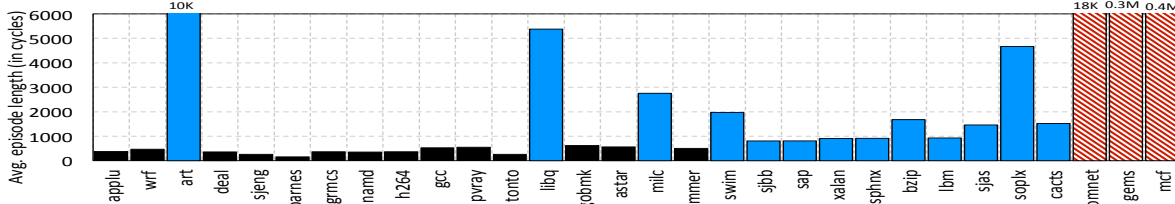
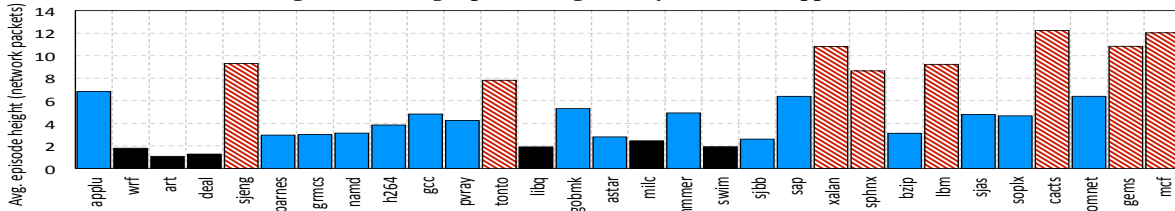Figure 4: Average episode length (in cycles) across applications.

Figure 5: Average episode height (in packets) across applications.

Classification		Length		
		Long	Medium	Short
Height	Tall	gems, mcf	sphinx,lbm, cactus, xalan	sjeng, tonto
	Medium	omnetpp, apsi	ocean, sjbb, sap, bzip, sjas, soplex, tpc	applu, perl, barnes, gromacs, namd, calculix, gcc, povray, h264, gobmk, hmmer, astar
	Short	leslie	art, libq, milc, swim	wrf, deal

Ranking		Length		
		Long	Medium	Short
Height	Tall	Rank-4	Rank-2	Rank-1
	Medium	Rank-3	Rank-2	Rank-2
	Short	Rank-4	Rank-3	Rank-1

Bandwidth-sensitive
Latency-sensitive

Figure 6: Application classification and ranking based on episode length and height.

by previous works [10, 11] as well as industrial prototypes [18, 7]. In canonical router designs, the arbitration and crossbar traversal stages are the bottleneck stages in terms of critical path [11]. In our design, since the latency-optimized network has only 3 VCs per physical channel and a narrow link width (64b), our analysis (based on synthesis of the RTL of VC and switch allocation and crossbar stages) shows that it is feasible to clock the routers in this network at a higher frequency.

Network bandwidth depends on link-width and frequency. A designer could think of increasing the frequency of the bandwidth-customized network to increase the total network bandwidth. However, increasing the frequency of the wider 256b network would adversely affect the power of this network. Hence, in our network designs, we only increase the frequency of the narrow 64b link-width network whose power envelope is 43% lower than the wider network. Increasing the frequency of the latency-customized network also increases this network's bandwidth, but since we steer only latency-sensitive applications (which are agnostic to bandwidth increase) into this network, the performance improvement of these applications is primarily due to network latency reduction.

The design space for optimizing network latency or bandwidth is large, and not possible to cover in this paper. Although we articulate frequency as a knob to improve latency in the latency-optimized network, a designer could use different topologies, flow-control, and arbitration mechanisms as various knobs to improve latency. The motivation of this paper is to demonstrate that heterogeneous multiple networks (customized for latency and bandwidth separately) provides a better design than a monolithic network or homogeneous multiple networks. The 64b 3X-frequency network for the latency-sensitive applications and 256b 1X-frequency network for the bandwidth-sensitive applications are only two design points to demonstrate the concept.

5. EVALUATION METHODOLOGY

Design scenarios: Starting with a monolithic network, we show the benefits of having two networks, each customized for either bandwidth or latency. We also show the benefits of our scheme compared to an iso-resource single network (with the same total bandwidth). Following are the nine design scenarios we evaluate on our experimental platform:

❶ **1N-128**: This design has a single homogeneous 128b link network. We assume this to be our starting point. Starting with this monolithic network, we increase its bandwidth to create a bandwidth-optimized network, and reduce its bandwidth (and increase its frequency) to design a latency-optimized network. ❷ **1N-256**: This configuration has a single homogeneous network with 256b links. We chose this as our *baseline* net-

work. Starting with this network, we first design a homogeneous multiple network design (where each network has equal bandwidth, 2N-128x128) and then customize one network for latency-sensitive applications and the other network for bandwidth-sensitive applications. ❸ **2N-128x128**: This design has two parallel networks, each with 128b link width. The buffer resources in each network is half that of the 1N-128 network and each of the networks operate at the same frequency as the cores. Packets are steered into each network with a probability of 0.5, i.e., there is load balancing across the networks. ❹ **1N-512**: This design has a single network with 512b link width. We call this a *high-bandwidth* configuration and analyze it to see how our proposal fares compared to a very high bandwidth network. ❺ **2N-64x256-ST**: In this design, there are two parallel networks, one with 64b link width and the other with 256b link width. The buffering resources in each network is half that of a single network, so that the total buffering resources are constant across this design and a design that has a single network. Further, in this configuration, the bandwidth-sensitive packets are steered (hence, the annotation **ST**) into the 256b network and the latency-sensitive packets are steered into the 64b network. Each network in this configuration is clocked at the frequency of the cores. ❻**2N-64x256-ST+RK(no FS)**: This design is the same as the previous network except that the network also prioritizes applications based on their ranks (hence, the annotation **RK**) at every cycle in a router. ❼**2N-64x256-ST+RK(FS)**: This design is same as the previous configuration except that the 64b network is clocked at 3X the frequency of cores. The 256b network is still clocked at the core frequency. This configuration is analyzed to see the benefits of frequency scaling (hence, the annotation **FS**) the latency-optimized network. ❽**1N-320(no FS)**: In this design, there is a single network with 320b (=64b+256b) links. The network operates at the frequency of the core. This configuration is iso-bandwidth with all our 64x256 networks and is analyzed to see the benefits of our proposal over an equivalent configuration. ❾**1N-320(FS)**: This design is similar to the above design, except that the network is now clocked at 3X the core frequency. This design is analyzed to see the effectiveness of our scheme over a scheme that is iso-resource as well as over-clocked to help latency-sensitive applications.

Experimental setup: Our proposals are evaluated on an instruction-trace-driven, cycle-level x86 CMP simulator. Table 1 provides the configuration of our baseline, which contains 64 cores in a 2D, 8x8 mesh NoC. The network connects the cores, shared L2 cache banks, and memory controllers (these all stay constant across all evaluated designs). A data packet consists of 1024b (=cache line size) and is decomposed into flits depending upon the link width in each design. Since wiring resources on die are abundant [1, 19], when simulating parallel networks, we assume the
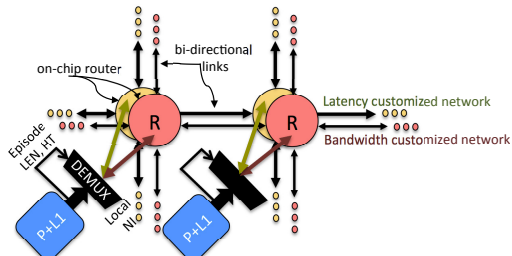
Figure 7: Schematic of the proposed CMP.

Table 1: Baseline core, cache, memory and network configuration

Core
128-entry instruction window, 2 INT/FP operations and 1 LD/ST per cycle
Caches and Main Memory
L1 Caches: 32 KB per-core (private), 4-way set associative, 128B block size, 2-cycle latency, write-back, 32 MSHRs
L2 Caches: 1MB per bank, shared, 16-way set associative, 128B block size, 3-cycle bank latency, 32 MSHRs
Main Memory: 4GB; up to 16 outstanding requests per processor, 320 cycle access
Network and Router
Network Router: 2-stage wormhole switched, virtual channel flow control, 6 VC's per port, 5 flit buffer depth, 1 flit/address packet, X-Y routing
Network Topology: 8x8 mesh, each node has a router, processor, private L1 cache and shared L2 cache bank (all nodes), 4 memory controllers (1 at each corner node), 256b bi-directional links (= data packet's flit width)

(a) Weighted speedup (WS) (system throughput)

(b) Instruction throughput (IT)

Figure 8: Performance comparison of various network designs, averaged across 25 multiprogrammed workload mixes.

networks to be implemented in the same 2D substrate as the cores. A high-level schematic of the proposed interconnection network is shown in Figure 7. The dynamic and leakage energy numbers for the network were extracted using Orion 2.0 [8] and Rock Creek router data [7], and incorporated into our simulator for detailed network energy analysis at 32nm technology node. Based on Orion 2.0 and Rock Creek layout estimates, the area of two networks (router and links) consisting of 256b and 64b links is only 1% larger than the iso-resource network with 320b links (2.4X larger area when compared to the 128b link network), and the power envelope of these two networks is 20% lower than the iso-area network (2.3X higher power when compared to the 128b link network). The counter configurations used in our techniques are: (1) counter size for number of cycles in a network episode, M=14bits (2) batching interval, B=16,000 cycles.

Application setup: We use a diverse set of multiprogrammed application workloads comprising scientific, commercial, and desktop benchmarks. We use the SPEC CPU2006 benchmarks, applications from SPLASH-2 and SPEC-OMP benchmark suites, and four commercial workload traces (sap, tpcc, sjbb, sjas) totaling 36 applications. All our experiments analyze multiprogrammed workloads, where each core runs a separate application. We simulate at least 640 Million instructions across 64 processors (minimum 10 Million instructions per core; a core keeps exerting pressure even after 10M instructions; on average each run simulates 3.5 Billion instructions in the system). All our results are aggregated across *25 workload combinations*. In each of these workload combinations, 50% (32) of the applications are latency-sensitive and 50% (32) of the applications are bandwidth-sensitive. This provides a good mix of bandwidth/latency-sensitive applications, which is likely to be a common mix for future multicore systems. Within each of these two categories, applications are *randomly picked* to form the workload.

Evaluation metrics: Our primary performance evaluation metrics are instruction throughput [4] and weighted speedup [17]. Instruction throughput is defined to be the sum total of the number of instructions committed per cycle (IPC) in the entire CMP. The weighted speedup metric sums up the speedup (inverse of slowdown) experienced by each application in a workload, compared to its standalone run in the same configuration, and represents *system throughput* [4].

6. ANALYSIS OF RESULTS

Performance comparison: Figure 8 shows the performance comparison across the network designs. The following observations are in order:

❶ Two 128b networks (2N-128x128) provide similar performance (both system and instruction throughput) as a bandwidth-equivalent single monolithic network with 256b links (1N-256). This is in spite of the increase in packet serialization in the networks. The primary reason for this per-
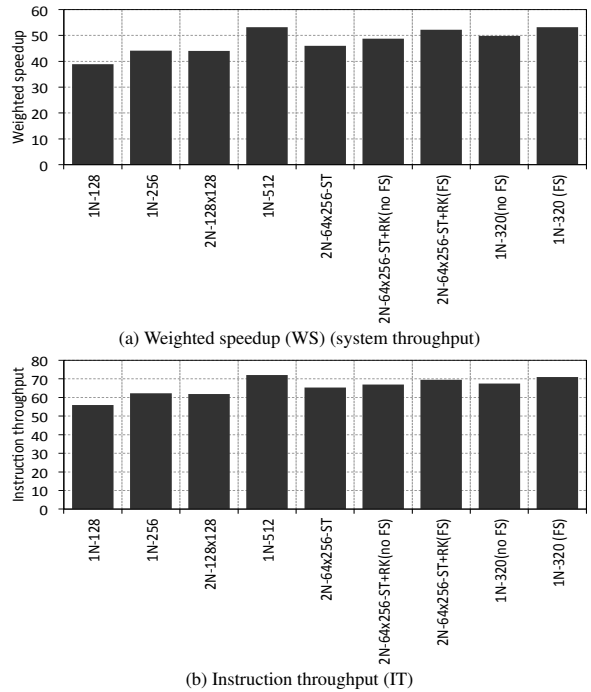
formance improvement is reduction in congestion in each network (each network now sees 50% fewer packets than a monolithic wider network). The total bandwidth in the 2N-128x128 design is the same as the 1N-256 design and hence bandwidth-sensitive applications' performance is not affected. On the other hand, the performance of latency-sensitive applications is improved because of the load balancing (reduced congestion is each network), and thus, the degradation in performance due to serialization latency increase is compensated by performance improvement due to reduced congestion.

❷ Bandwidth- and latency-optimized parallel networks operating at the same frequency as the processor along with steering of packets based on their bandwidth/latency sensitivity (2N-64x256-ST) provide 4.3%/5% system/instruction throughput improvement, over the baseline (1N-256) design. By providing bandwidth-sensitive applications more bandwidth (than 1N-128) and reducing the congestion compared to a monolithic network, the performance of both bandwidth- and latency-sensitive applications is improved. Prioritizing and ranking packets based on their criticality after steering them into a network (2N-64x256-ST+RK(no FS)) provides an additional 6%/3% improvement in system/instruction throughput over the 2N-64x256-ST design. This is because, our ranking scheme prioritizes the relatively network-sensitive applications in each network, and ensures no starvation using batching.

❸ Frequency scaling the latency-optimized network along with steering and ranking the applications (2N-64x256-ST+RK(FS)) provides the maximum performance improvement among our proposals: 18%/12% system/instruction throughput improvement over the baseline network. With frequency scaling, the latency-optimized network is clocked at a higher frequency, accelerating the latency-sensitive packets and this brings the additional benefits in performance.

❹ Frequency scaling and steering along with ranking of applications (2N-64x256-ST+RK(FS)) is better than an iso-resource network (1N-320(no FS)) by 5%/3% in weighted/instruction throughput. 2N-64x256-ST+RK(FS) design is within 2.0%/2.2% system/instruction throughput of the high-frequency iso-resource network with frequency increased by 3X (1N-320(FS)). High frequency of the 1N-320(FS) network helps latency-sensitive applications and high bandwidth of the same network (compared to 256b links) helps bandwidth-sensitive applications. But, as we will show shortly, the energy consumption of such a wide network is higher than our proposal.

❺ Our proposed 2N-64x256-ST+RK(FS) design's system performance is within 1.8% of a very high bandwidth network (1N-512). A high bandwidth network helps bandwidth-sensitive applications, but provides little
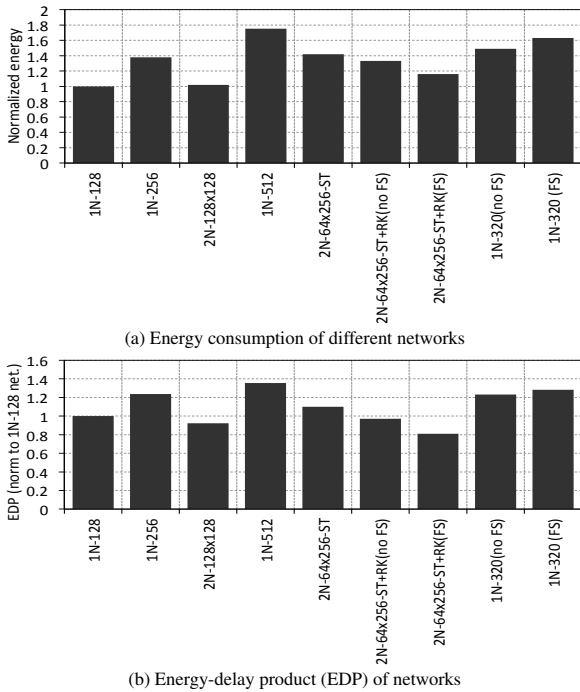
(a) Energy consumption of different networks

(b) Energy-delay product (EDP) of networks

Figure 9: Energy and EDP comparison of various network designs (all results normalized to the 1N-128 network).

benefit for other applications. Additionally, as will be shown next, a wide-link network's energy consumption is very high (about 75% higher than a 128b link-width network). Hence, although our proposed network provides similar performance as a very high bandwidth network, it does so at a lower energy envelope.

Energy and EDP comparison: Increasing the link width decreases the serialization (and zero-load) latency and hence, end-to-end latency is reduced. However, increasing the link width also affects router crossbar power quadratically. Figure 9 shows the energy and energy-delay product (EDP) of the applications across the 9 designs. We find that:

❶ The average energy consumption of a 256b link network (1N-256) is 38% higher than a 128b link network (1N-128). However, the two 128b network design (2N-128x128) has similar energy consumption as a single 128b link monolithic network. The energy reduction going from one network to two networks comes primarily from reduction in network latency (by reducing the congestion in each network). In fact, we observed that the energy consumption of two parallel networks, each with link width $N/2$, is always lower than a single network with link width N. When compared to the 1N-128 network, the 2N-128x128 network has the same buffering resources but double the number of routers and links, which leads to higher power. However, the 2N-128x128 has better performance and less contention, which leads to lower energy. These two effects mostly cancel out, leading to similar energy consumption for 2N-128x128 and 1N-128.

❷ The average energy consumption of a high bandwidth (512b links) network (1N-512) is 75% and 26% higher than a 128b and 256b link network respectively. When link width increases, although serialization latency reduces, the crossbar power starts to dominate the energy component.

❸ Our proposed design with two heterogeneous networks and fine-grained application prioritization in each network (2N-64x256-ST+RK(FS)) consumes 16% lower energy than the baseline 1N-256 network. This is 39% lower energy when compared to a high-bandwidth network (1N-512) and 31% lower energy than an iso-resource network which is frequency scaled (1N-320(FS)). Overall, our proposed design consumes lower energy than an iso-resource 320b link network, and the 2N-64x256-ST network. Note that using fine-grained prioritization within each network provides 6.7% energy reduction over not doing so.

❹ On the EDP metric, our proposed design (2N-64x256-ST+RK(FS)) is 35% better than the baseline (1N-256). Our scheme reduces network latency significantly, which lowers the delay component in the EDP metric. Even without frequency scaling, the 2N-64x256-ST+RK(no FS) design has 22% lower EDP than the baseline. Our proposed design always has lower EDP than a high-bandwidth network (1N-512), the iso-resource

320b link network, or the iso-resource 2N-64x256-ST network.

Summary: Overall, compared to an iso-resource 320b link monolithic network operating at 1X frequency, we find that the combination of a 64b network operating at 3X frequency and a 256b network operating at 1X frequency (along with our steering and prioritization algorithms) provides better performance, energy and EDP.

7. CONCLUSIONS

We proposed an application-driven approach for designing high-performance and energy-efficient heterogeneous on-chip networks (NoCs). The goal of our design is to cater to the applications' requirements more effectively and more efficiently than an application-agnostic monolithic NoC design. The main idea is to have one network customized for low latency and another customized for high bandwidth, and steer latency-sensitive applications to the former and the bandwidth-sensitive applications to the latter. Within each network, applications that are more likely to improve system throughput are prioritized over others. We find that network episode height and length are simple-to-measure metrics to classify applications in terms of bandwidth- and latency-sensitivity. Our results show that the proposed heterogeneous two-network design outperforms a single monolithic network or two homogeneous networks. We conclude that our application-driven methodology for designing heterogeneous NoCs provides high system and application performance at reduced energy, and hope that the proposed dynamic application classification scheme can provide a simple framework for designing heterogeneous on-chip networks.

References

[1] J. Balfour et al. "Design Tradeoffs for Tiled CMP On-Chip Networks". In *ICS*. 2006.

[2] R. Das et al. "Aergia: Exploiting Packet Latency Slack in On-Chip Networks". In *ISCA*. 2010.

[3] R. Das et al. "Application-Aware Prioritization Mechanisms for On-Chip Networks". In *MICRO*. 2010.

[4] S. Eyerman et al. "System-Level Performance Metrics for Multi-program Workloads". In *IEEE Micro* (2008).

[5] A. Glew. "MLP Yes! ILP No!" In *ASPLOS WACI*. 1998.

[6] M. Hayenga et al. "The NoX router". In *MICRO*. 2011.

[7] J. Howard et al. "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling". In *J. Solid-State Circuits* 46.1 (2011).

[8] A. B. Kahng et al. "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration". In *DATE*. 2009.

[9] Y. Kim et al. "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers". In *HPCA*. 2010.

[10] A. Kumar et al. "A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS". In *ICCD*. 2007.

[11] A. K. Mishra et al. "A Case for Dynamic Frequency Tuning in On-Chip Networks". In *MICRO*. 2010.

[12] T. Moscibroda et al. "A Case for Bufferless Routing in On-Chip Networks". In *ISCA*. 2009.

[13] R. Mullins et al. "Low-Latency Virtual-Channel Routers for On-Chip Networks". In *ISCA*. 2004.

[14] O. Mutlu et al. "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance". In *IEEE Micro* (2006).

[15] O. Mutlu et al. "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems". In *ISCA*. 2008.

[16] C. A. Nicopouloulos et al. "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers". In *MICRO*. 2006.

[17] A. Snavely et al. "Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor". In *ASPLOS*. 2000.

[18] S. Vangal et al. "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS". In *ISSCC*. 2007.

[19] D. Wentzlaff et al. "On-Chip Interconnection Architecture of the Tile Processor". In *IEEE Micro* (2007).
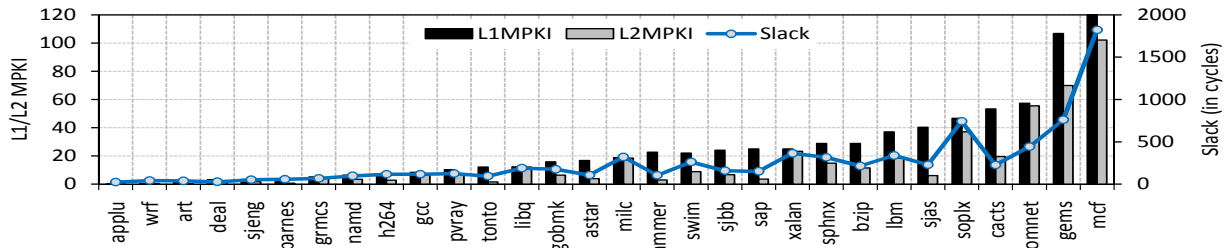
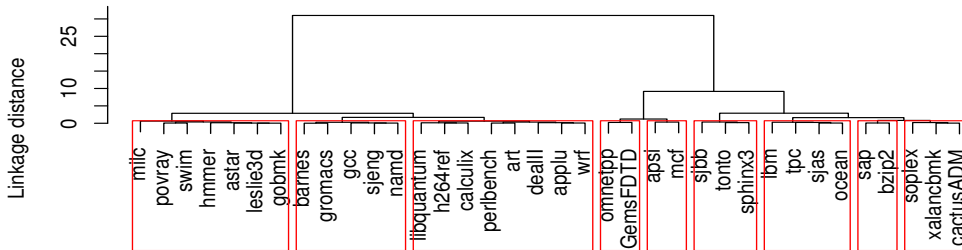**Figure 10: Average L1MPKI, L2MPKI and slack of applications.**

Figure 11: Hierarchical clustering of applications. Applications on the left branch of the root are latency-sensitive, those on the right branch are bandwidth-sensitive.
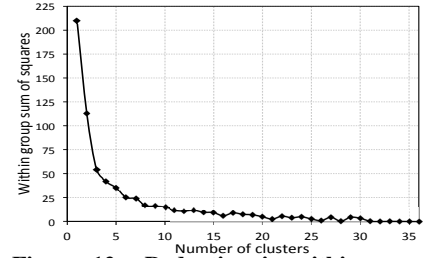

Figure 12: Reduction in within-group sum of squares with increase in number of clusters.

S. SUPPLEMENTAL

In this supplemental section, we present further analyses of our proposed metrics and how they compare to previously proposed criticality metrics, explain more the rationale for our application classification into sub-classes, discuss subtle but important design aspects and how we addressed them, provide more discussion on application characteristics, and present how our scheme compares to closely related previous works.

S.1 Network Episodes vs. Other Metrics

In Section 3, we use the notion of network episodes for classifying applications into latency-sensitive or bandwidth-sensitive categories. We propose two new metrics, network episode height and length for the classification. We contrast these proposed metrics to two recently proposed heuristics, L1MPKI [8] and slack [7], which were proposed to estimate a packet's criticality in the network.

Private cache misses per instruction (MPI): This metric captures an application's network intensity. If the network intensity is low, the application likely (but not always) has low MLP and hence its request are likely to be latency-sensitive as opposed to bandwidth-sensitive. Figure 10 shows the L1MPKI and L2 MPKI of 30 applications. We find that MPI (or MPKI) can help in identifying latency-sensitive applications from bandwidth-sensitive ones. In Figure 10, all applications to the left of sjbb have a lower MPKI than sjbb's MPKI. Since these applications are latency-sensitive, empirically we can think of having a threshold in MPKI (equal to sjbb's MPKI) to classify applications as bandwidth- or latency-sensitive. However, as mentioned earlier, this metric is not as effective in distinguishing between applications *within* the latency-sensitive class or bandwidth-sensitive class (as we will demonstrate in Section S.5.4).

Packet slack: Slack was investigated by Das et al. [7] to identify a packet's criticality in the network. It quantifies how long a packet can be delayed without affecting an application's performance. We measured an instruction's slack from when it enters the re-order buffer (ROB) to when the instruction actually becomes the oldest in the ROB and is ready to commit. Figure 10 shows how average slack varies across applications. Intuitively, the slack of an L1-miss instruction directly translates to the packet's criticality in the network: an application that has a larger average slack is more tolerant to network delay versus an application that has a smaller slack. We find that, although slack is very useful in prioritizing individual packets' criticality, it has low correlation in identifying increase in performance with increase in bandwidth or frequency for a given application.

S.2 Rationale for Application Classification

In Section 3, we classify applications into two gross classes, latency- and bandwidth-sensitive, and nine sub-classes. The sub-classes aid our mechanism in fine-grained prioritization within the networks to further improve performance and reduce energy. We take two decisions in our

application-level classification: (1) choosing sjbb's episode length and height as a threshold to distinguish between applications, and (2) choosing 9 smaller sub-classes after classifying the applications as bandwidth- or latency-sensitive. Here, we present more empirical results that had led us to these decisions.

Figure 11 shows the results of the *hierarchical clustering* of all the applications in our benchmark suite. Hierarchical clustering incrementally groups objects that are similar, i.e., objects that are close to each other in terms of some distance metric [13]. The input to the clustering algorithm consists of the improvement in IPC with bandwidth scaling (from 64b to 512b) and improvement in IPC with router latency scaling (from 1X to 3X), i.e., values from Figures 1 and 2. Our purpose is to observe whether a clustering algorithm perceives noticeable difference between applications' performance with frequency and bandwidth scaling. To perform clustering, we tried using various *linkage distance* metrics (linkage metric determines the affinity between sets of data-points as a function of the pairwise distances between them), such as Euclidean distance, Pearson correlation, and average distance between the objects, and found similar clustering results with all. In all cases, the clustering was consistent with that shown in Figure 11 (shown for Euclidean distance). Although the eventual hierarchical cluster memberships are different from that shown in our classification matrix in Figure 6, the broader classification of how hierarchical clustering groups applications into bandwidth- and latency-sensitive clusters (from the root, the left dendrogram represents latency-sensitive applications and the right dendrogram represents bandwidth-sensitive applications) matches exactly with our classification scheme, which is based on episode height and length – with the exception of sjeng. The reason for sjeng's misclassification is that its performance does not scale with bandwidth and hence, hierarchical clustering classifies it as a latency-sensitive application. However, sjeng's episode height is tall and length is short, on average, meaning it is very bursty during a small interval of time (as such, it has high MLP). Because of this, we classify it as the highest ranking application in the bandwidth-optimized network.

Why 9 sub-classes? To answer this question, we measure the total within-group sum-of-squares (WG-SS) of the clusters resulting from hierarchical clustering. Figure 12 shows this metric as the number of clusters increases. The total WG-SS is a measure of the total dispersion between individual clusters and is regarded as a metric to decide the optimal number of clusters from a hierarchical or K-means clustering algorithm [18, 27]. When all clustering objects are grouped into one cluster, the total WG-SS is maximized, whereas, if each object is classified as a separate object, WG-SS is minimized (=0). Figure 12 suggests that 8 or 9 clusters have similar WG-SS and, 8 or 9 clusters reduce the total WG-SS by 13X compared to a single cluster. Based on this, we choose 9 classes for our application classification and, hence, quantize episode height and length into three classes each.

7

S.3 More Design Details

We present two important design decisions we made while architecting our scheme: 1) how to handle starvation, 2) how to handle packet reordering.

Handling starvation: Prioritizing highly-ranked packets in a network may lead to starvation of other packets. To prevent starvation, we combine our application-aware prioritization with a "batching mechanism" proposed by Das et al. in [8, 7]. Each packet is part of a batch, and packets belonging to older batches are prioritized over packets from younger batches. Only if two packets belong to the same batch, they are prioritized based on their applications' rank order that is based on episode height/length. A batch also provides a convenient granularity in which the ranking of the applications is enforced.

To support batching, each node keeps a local copy of a batch-ID (BID) register containing the current (injection) batch number and maximum supported batch-ID register containing the maximum number of batching priority levels (L). BID is incremented every B cycles, and thus, BID values across all nodes are the same. Due to batch-ID wrap-around, a router cannot simply prioritize packets with lower batch-IDs over others with higher batch-IDs, and we use schemes suggested in [8, 7] to handle relative priorities inside a router. In all our experiments we used L=8 as the number of batching levels.

Handling packet reordering: Due to deterministic routing in the baseline design, for every pair of source and destination in the network, there is always a single path from the source to the destination. Multiple packets on this path do not get re-ordered. When using our scheme with multiple networks, since there are multiple routes between a source and destination pair (via the two different networks), there is a chance of packet reordering in the network. The effect arising out of reordering could be handled either by the software or application layer or by the network itself.

In our case, we use packet sequence numbers to handle reordering. For each destination, a source router maintains sequence numbers (8 bit) which every outgoing packet is tagged with. These sequence numbers are reset at the end of every batch. With this, whenever a cache controller or memory controller receives a request packet to be serviced (in the attached cache bank or memory), the controller inspects the sequence number of the packet. If the sequence number of the packet is *not* the next in sequence to the last serviced packet, then the controller does not service the packet and buffers it in the NIC queues. With this scheme, even if a node (cache or memory controller) receives packets out-of-order, the packets are always serviced in order. Note that reordering in the network could also be introduced due to the use of adaptive routing or deflection routing [10, 11]. Our scheme could leverage any mechanism that handles packet reordering in such systems.

S.4 Application Characteristics

Table 2 characterizes our application suite. The reported parameters are for the applications running alone on the baseline system without any interference. The table shows application characteristics based on network load intensity (high/low), episode height (tall/medium/short), episode length (long/medium/short) and the fraction of execution time spent in network episodes. It is this fraction of execution time spent in the network that is reduced by our scheme which leads to improvement in performance. Energy efficiency benefits come from 1) steering each application to a network that is more appropriately provisioned for the application's demand, 2) prioritizing applications appropriately such that contention that degrades both energy and performance reduces.

S.5 More Results and Analyses

S.5.1 Reply packets from L2 cache (DRAM) to L1 cache (L2 cache)

In all of our earlier evaluations, we route the L2 cache (DRAM) replies to the L1 cache (L2 cache) in either the 64b or the 256b network depending on where the request packet traversed the network: if the request packet was bandwidth-sensitive, the matching reply is sent on the 256b network and vice-versa.

Reply packets are L1/L2 cache line sized packets (1024b). Transmitting them over the 64b network increases their serialization latency. However, the 64b network is relatively less congested than the 256b network (because of lower injection ratio of latency-sensitive applications). Since

Table 2: Application characteristics when run on the baseline (Load: High/Low depending on network injection rate, Episode height: Tall/Medium/Short, Episode length: Long/Medium/Short, Net. fraction: Fraction of execution time spent in network episodes)

#	Benchmark	Load	Episode height	Episode length	Net. fraction
1	applu	Low	Medium	Short	8.2%
2	wrf	Low	Short	Short	9.4%
3	perlbench	Low	Medium	Short	8.8%
4	art	Low	Short	Medium	82.3%
5	dealII (deal)	Low	Short	Short	27.9%
6	sjeng	Low	Tall	Short	28.4%
7	barnes	Low	Medium	Short	72.5%
8	gromacs (grmcs)	Low	Medium	Short	48.6%
9	namd	Low	Medium	Short	51.6%
10	h264ref (h264)	Low	Medium	Short	61.5%
11	calculix	Low	Medium	Short	48.2%
12	gcc	Low	Medium	Short	47.6%
13	povray (pvray)	Low	Medium	Short	59.6%
14	tonto	Low	Tall	Short	53.0%
15	libquantum (libq)	Low	Short	Medium	99.0%
16	gobmk	Low	Medium	Short	64.9%
17	astar	Low	Medium	Short	82.8%
18	milc	Low	Short	Medium	88.2%
19	ocean	Low	Medium	Medium	90.1%
20	hmmer	Low	Medium	Short	66.1%
21	swim	Low	Short	Medium	41.0%
22	sjbb	High	Medium	Medium	87.3%
23	sap	High	Medium	Medium	88.9%
24	xalancbmk (xalan)	High	Tall	Medium	89.9%
25	sphinx3 (sphnx)	High	Tall	Medium	83.9%
26	bzip2 (bzip)	High	Medium	Medium	84.9%
27	lbm	High	Tall	Medium	81.1%
28	sjas	High	Medium	Medium	89.5%
29	soplex (soplx)	High	Medium	Medium	81.2%
30	tpc	High	Medium	Medium	86.8%
31	cactusADM (cacts)	High	Tall	Medium	82.3%
32	leslie3d	High	Short	Long	99.7%
33	omnetpp	High	Medium	Long	92.6%
34	GemsFDTD	High	Tall	Long	97.3%
35	apsi	High	Medium	Long	95.2%
36	mcf	High	Tall	Long	99.2%

the 64b network is clocked at 3X frequency, the network latency in this network is also lower. Our analysis shows that transmitting *all* the reply packets in the 256b network increases the system/instruction throughput by an additional 1.6%/2.4% and reduces energy consumption by an additional 4% when compared to the baseline network. Also, since coherence packets are usually latency-sensitive, our design can route them in the latency-optimized network.

S.5.2 Intra-application latency/bandwidth sensitivity

Figure 13 shows the percentage of packets in an application *dynamically steered* into each network via our mechanism. The data for this figure is collected by averaging 25 workload combinations consisting of 50% latency and 50% bandwidth-sensitive applications (the same workloads used for Figures 8 and 9). The results highlight the dynamic nature of our scheme, where each application phase is classified as either latency or bandwidth-sensitive and all packets belonging to a phase are steered into either of the networks. The figure depicts the intra-application variance: at one extreme is gems which has 99% bandwidth-sensitive packets and on the other extreme is wrf which has 98% latency-sensitive packets. All other applications in our benchmark suite lie within these two extremes.

S.5.3 Sensitivity to fraction of bandwidth- and latency-sensitive applications in the workload

In all results shown so far, each workload is a multiprogrammed mix of applications with equal percentage of latency- and bandwidth-sensitive applications. To analyze the sensitivity of our scheme across more diverse multiprogrammed application mixes, we vary the fraction of bandwidth-sensitive applications in a workload from 0% to 100%. Figure 14 shows the results of this analysis (the results are normalized to the 128b monolithic network). We find that our proposal, in general, has higher system/instruction throughput across the entire spectrum of workload mixes. However, as expected, the benefits are small (4%/9% system/instruction
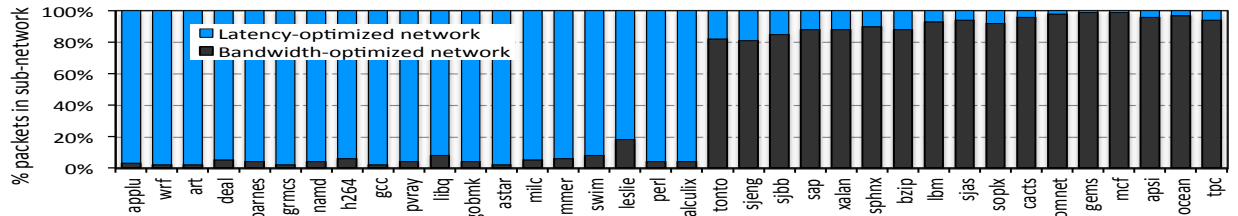
Figure 13: Fraction of packets dynamically steered into latency- vs. bandwidth-optimized networks.
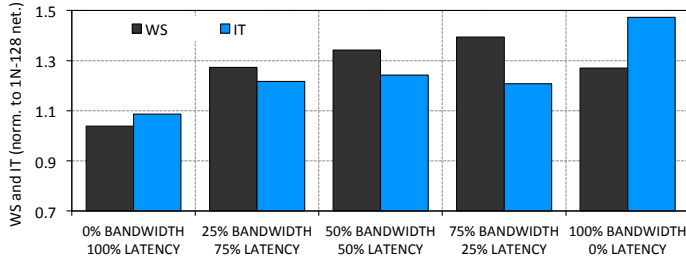

Figure 14: Performance improvement versus the proportion of bandwidth- and latency-sensitive applications in each workload.
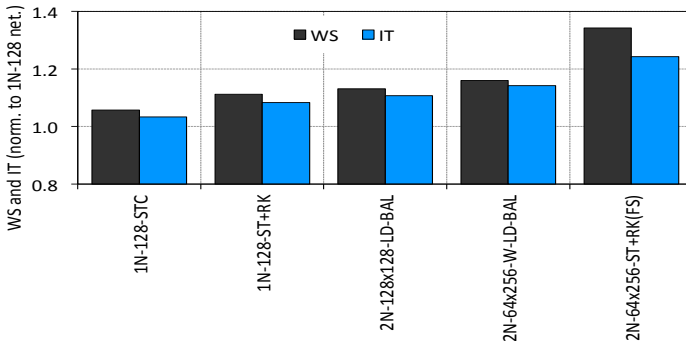

Figure 15: Weighted speedup (WS) and instruction throughput (IT) versus state-of-the art designs [8, 5] (normalized to 1N-128).

throughput improvement over 1N-128 design) when the system has 100% latency-sensitive applications. When the application mix is skewed (i.e., the system has *only* bandwidth *or* latency-sensitive applications), we weighted-load-balanced the two networks (as described in Section S.5.4). As such, with 100% latency-sensitive applications in the workload mix, benefits arise only due to load distribution and reduction in congestion, rather than customization of the network to application requirements (which is the purpose of heterogeneity). We conclude that, even though our design provides performance (and energy) benefits when the application mix is homogeneous in terms of latency- and bandwidth-sensitivity, our heterogeneous network design provides the largest performance and energy improvements when the application mix is heterogeneous.

S.5.4   Comparison with prior works

Das et al. [8] proposed a ranking framework, called stall time criticality (STC), which is based on the dynamic identification of the criticality of a packet in the network. The authors use the L1MPKI metric to estimate the criticality of an application's packets and use a ranking framework that ranks applications with lower L1MPKI over applications with higher L1MPKI. In their work, a *central* decision logic periodically gathers information from each node, determines a global application ranking and communicates this information to each node. Each node then prioritizes packets 1) belonging to the oldest batch over others and 2) in their application rank order within a batch. Since our proposal also prioritizes applications in the network using a rank order, we compare our scheme with STC. When comparing with STC for a single network design, we utilize a 2-level ranking scheme for our technique. The first-level ranking prioritizes latency-sensitive applications over bandwidth-sensitive applications. Within the latency- and bandwidth-sensitive application groups, we use episode width and height to rank the applications (Section 3).

Balfour and Dally [5] showed the effectiveness of load balancing traffic *equally* over two parallel networks. In this work, each of the networks

is a concentrated mesh with similar bandwidth. With detailed layout/area analysis, the authors found that a second network has no impact on the chip area since the additional routers can reside in areas initially allocated for wider channels in the first network. As we also propose parallel networks (although our results in Section 6 show heterogeneous networks are better than homogeneous), we compare our scheme with a similar load-balancing scheme proposed by Balfour and Dally [5].

Figure 15 shows the results, where we compare the performance of our schemes with the two prior proposals mentioned above. All numbers in these plots are normalized to that of a 128b-link network with no prioritization (the link width used in the STC work [8]). The STC schemes are annotated as **-STC** and the load-balancing schemes are annotated as **-LD-BAL** with a given network design in the figure.

The performance improvement of STC is 6%/3% system/instruction in a 128b link monolithic network. Our 2-level ranking scheme shows 11%/8% system/instruction throughput improvement over the 1N-128 design. As shown earlier, L1MPKI, which STC uses, is not a very strong metric to decide the fine-grained latency/bandwidth sensitivity of applications. L1MPKI metric does not take into account the *time* factor, which is taken into account by network *episode* height and length metrics. We found that taking time into account provides a better indication of 1) an application's average MLP (bandwidth demand), 2) how important prioritizing the application will be in relative terms to other applications. Both of these factors are inherently is affected by the notion of *time*. Averaging *episode height* over time, and measuring *episode length* provides better estimators for these two factors than L1MPKI, which leads to the performance benefit with our schemes compared to STC.

In Figure 15, 2N-128x128-LD-BAL design is the one proposed by Balfour and Dally in [5]: *equal* load balancing across two parallel networks. Since we propose heterogeneous networks, when load balancing between two networks in comparison to [5], we steer packets in the weighted-ratio of $\frac{256}{256+64}$ and $\frac{64}{256+64}$ between the 256b and the 64b network. This scheme is annotated as **-W-LD-BAL**. We consider weighted-ratio load balancing, because our evaluations show that, steering packets with *equal* probability into each network leads to more congestion in the 64b-link network and under-utilizes the 256b-link network, leading to suboptimal performance. Overall, we find that our proposal (2N-64x256-ST+RK(FS)) has an additional 18%/10% system/instruction throughput improvement over the weighted load-balancing scheme (2N-64x256-W-LD-BAL). The load balancing scheme is oblivious to application characteristics. With this scheme, a latency-sensitive packet is steered into the bandwidth-optimized network with a probability of 0.8 and a bandwidth-sensitive packet is steered into the latency-optimized network with a probability of 0.2. As a result, in both cases performance either degrades or does not improve. We conclude that, with heterogeneous networks, application-unaware load balancing (either equal or weighted-ratio) is suboptimal and that intelligently steering packets based on the applications' latency versus bandwidth sensitivity can lead to significant performance benefits.

S.6   Related Work

To our knowledge, this is the first proposal for a heterogeneous on-chip network where one network is customized for latency-sensitive workloads and the other for bandwidth-sensitive ones. Our paper is also the first to use the metrics network episode height and length to characterize the latency- and bandwidth-sensitivity of applications. Our proposal is related to other proposals of multiple networks, heterogeneous networks, and request prioritization, which we review below.

Multiple networks: We have already compared our scheme with Balfour and Dally's proposal for homogeneous multiple networks [5], and showed that our scheme is more effective than a (weighted) load balanc-

ing scheme. Other prior works that have proposed multiple networks for NoCs include TRIPS [24], RAW [26], Tilera [29], and IBM cell [14, 2]. The motivation for including multiple networks in all these designs is entirely different from ours. In TRIPS, multiple networks are used to connect operand networks. RAW has two static networks (routes specified at compile time) and two dynamic networks (one for trusted and other for untrusted clients). Cell's EIB has a set of four unidirectional concentric rings (arranged in groups of four and interleaved with ground and power shields) primarily to reduce coupling noises. DASH multiprocessor [19] had multiple networks (request and reply meshes) to eliminate request-reply deadlocks. Tilera's iMesh network consists of five separate networks to handle different packet sizes: memory access, streaming packet transfers, user data, cache misses, and interprocess communication. In contrast to all these multiple-network designs, our proposal customizes each network to cater to a different class of applications.

Heterogeneity in networks: HeteroNoC [20], polymorphic NoCs [15] and Kilo-NOC [12] have explored heterogeneity in NoCs from router micro-architecture, topology and QoS perspectives, respectively. HeteroNoC constructs heterogeneous networks by using two types of routers and is agnostic to application properties. Polymorphic NoCs provide per-application network reconfiguration (that needs to be done before the application is run) and incurs a high area overhead to provide performance benefits. Kilo-NOC uses two kinds of routers, QoS-enabled and not QoS-enabled, to provide low cost, scalable and energy-efficient QoS guarantees in a network. Prior work has also investigated co-designing the NoC with caches [6, 4], asymmetric cores [20] and memory controllers [1, 20]. In particular, works in [6, 4] have examined heterogeneous wires with varying width, latency and energy, and proposed mapping coherence messages with differing latency and bandwidth characteristics onto the different wires. Similar to ours, Volos et al. [28] proposes two asymmetric networks, one customized for coherence and short messages and the other for cache block reply packets. Most of these past works have investigated heterogeneity or customization in the network based on micro-architectural (long or short packets) or hardware characteristics (coherence). Our approach is different because it provides an *application-aware* design for heterogeneous networks – as such, we expect our techniques can potentially be combined with these other proposals for heterogeneity.

Request Prioritization: There has been extensive research on prioritizing memory accesses based on their importance to overall application performance and system throughput/fairness [8, 7, 17, 22, 16, 23, 9, 21, 3, 25]. We already presented comparisons of our proposal to [8]. Closely related to our work is Thread Cluster Memory Scheduling (TCM) by Kim et al. [17]. In this work, the authors propose mechanisms to group threads with similar memory access behavior into either latency-sensitive or bandwidth-sensitive clusters and prioritize memory accesses (to DRAM) of the latency-sensitive cluster over those of the bandwidth-sensitive cluster at the memory controller. Our work is related to TCM in the sense that we also exploit latency and bandwidth sensitivity of applications to improve system performance. However, our proposal is different and complementary since it is in the context of NoCs, uses new metrics for classification of applications, and exploits heterogeneous networks.

The concept of compute and non-compute episodes has been used in ATLAS [16]. ATLAS defines *memory* episode length to be the duration for which an application awaits at least one memory request (i.e., L2 miss). ATLAS exploits the notion of memory episodes to prioritize threads with the least attained memory service times. In contrast, we use the concept of *network* episode length, which is the duration for which an application awaits at least one L1 miss. We use network episode characteristics to classify applications into categories. Further, we introduce the notion of network episode *height* to measure an application's MLP, which no previous work has done.

Acknowledgments

References

[1] D. Abts et al. "Achieving Predictable Performance Through Better Memory Controller Placement in Many-Core CMPs". In *ISCA*. 2009.

[2] T. W. Ainsworth et al. "Characterizing the Cell EIB On-Chip Network". In *IEEE Micro* (2007).

[3] R. Ausavarungnirun et al. "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems". In *ISCA*. 2012.

[4] R. Balasubramonian et al. "Microarchitectural Wire Management for Performance and Power in Partitioned Architectures". In *HPCA*. 2005.

[5] J. Balfour et al. "Design Tradeoffs for Tiled CMP On-Chip Networks". In *ICS*. 2006.

[6] L. Cheng et al. "Interconnect-Aware Coherence Protocols for Chip Multiprocessors". In *ISCA*. 2006.

[7] R. Das et al. "Aergia: Exploiting Packet Latency Slack in On-Chip Networks". In *ISCA*. 2010.

[8] R. Das et al. "Application-Aware Prioritization Mechanisms for On-Chip Networks". In *MICRO*. 2010.

[9] E. Ebrahimi et al. "Parallel Application Memory Scheduling". In *MICRO*. 2011.

[10] C. Fallin et al. "CHIPPER: A Low-Complexity Bufferless Deflection Router". In *HPCA*. 2011.

[11] C. Fallin et al. "MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect". In *NOCS*. 2012.

[12] B. Grot et al. "Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees". In *ISCA*. 2011.

[13] T. Hastie et al. *The Elements of Statistical Learning (2nd edition)*. Springer-Verlag, 2008.

[14] J. A. Kahle et al. "Introduction to the Cell Multiprocessor". In *IBM J. of Research and Development* (2005).

[15] M. M. Kim et al. "Polymorphic On-Chip Networks". In *ISCA*. 2008.

[16] Y. Kim et al. "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers". In *HPCA*. 2010.

[17] Y. Kim et al. "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior". In *MICRO*. 2010.

[18] W. J. Krzanowski et al. "A Criterion for Determining the Number of Groups in a Data Set Using Sum-of-Squares Clustering". In *Biometrics* 44.1 (1988).

[19] D. Lenoski et al. "The Stanford Dash multiprocessor". In *IEEE Computer* (1992).

[20] A. K. Mishra et al. "A Case for Heterogeneous On-Chip Interconnects for CMPs". In *ISCA*. 2011.

[21] S. P. Muralidhara et al. "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning". In *MICRO*. 2011.

[22] O. Mutlu et al. "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems". In *ISCA*. 2008.

[23] O. Mutlu et al. "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors". In *MICRO*. 2007.

[24] K. Sankaralingam et al. "Exploiting ILP, TLP, and DLP with The Polymorphous TRIPS Architecture". In *ISCA*. 2003.

[25] L. Subramanian et al. "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems". In *HPCA*. 2013.

[26] M. B. Taylor et al. "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs". In *IEEE Micro* (2002).

[27] R. Tibshirani et al. "Estimating the Number of Clusters in a Data Set via the Gap Statistic". In *Journal of the Royal Statistical Society*. 63.2 (2001).

[28] S. Volos et al. "CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache-Coherent Servers". In *NOCS*. 2012.

[29] D. Wentzlaff et al. "On-Chip Interconnection Architecture of the Tile Processor". In *IEEE Micro* (2007).