# Design of Heterogeneous On-Chip Networks: An Application Driven Approach

Asit K. Mishra
Pennsylvania State University
amishra@cse.psu.edu

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Chita R. Das
Pennsylvania State University
das@cse.psu.edu

SAFARI Technical Report No. 2011-010

October 2, 2011

**Abstract**

An on-chip interconnect is a critical shared resource that affects the performance-energy envelope of an entire multicore system. This aspect has led to a plethora of proposals in recent years for efficiently architecting the NoC substrate. However, most of these designs are agnostic to the actual application requirements in that they attempt to optimize a generic set of objective functions such as latency and throughput and/or energy/power. In this paper, we show that not all applications demand similar resources from the underlying interconnection substrate. Consequently, an alternative approach to design an NoC is to utilize multiple networks, each of which is specialized for common application requirements, and dynamically steer requests of an application to the network that matches the application's requirements.

To this end, we start with a top-down approach by analyzing the communication requirements of several applications. Our key observation is that, although applications, in general, can be classified as either network bandwidth sensitive or latency sensitive, not all bandwidth (latency) sensitive applications are equally sensitive to bandwidth (latency). Thus, we propose a novel set of metrics that can dynamically classify applications as either bandwidth or latency sensitive to steer them into appropriate networks. We propose two separate heterogeneous networks in the on-chip interconnection substrate, where one network is tailored to optimize for bandwidth sensitive applications and the second network for latency sensitive applications. Within each sub-network, we prioritize applications based on their criticality of network resource demand. Simulations with different designs of a 64-core 2D mesh architecture demonstrate that our heterogeneous network architecture is 5%/3% better in weighted/instruction throughput, while consuming 47% lower energy compared to an iso-resource single network.

## 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. The NoC is a critical shared resource among these concurrently-executing applications, significantly affecting each application's performance, system performance, and energy efficiency. Applications that share the NoC are likely to have diverse characteristics and performance requirements, resulting in different performance demands from the network. The design parameters and algorithms employed in the NoC critically affect the latency and bandwidth provided to each application, thereby affecting the performance and efficiency of each application's execution. Therefore, devising NoCs that can efficiently satisfy diverse characteristics of different applications is likely to become increasingly important.

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) common-case, or average-case, application behavior or 2) near-worst case application behavior, by overprovisioning 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 terms of either performance or efficiency. The "average-case" network design cannot provide good performance for applications that require more than the supported bandwidth or benefit from lower latency. Both network designs, especially the "overprovisioned" design, is power- and energy-inefficient for applications that do not need the provided high bandwidth or low latency. Hence, monolithic, one-size-fits-all NoC designs are sub-optimal from performance and energy standpoints.

Ideally, we would like an NoC design that can provide just the right amount of bandwidth and latency for an application such that the application's performance is 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, this paper proposes a new approach to designing an on-chip interconnect that can satisfy the diverse performance requirements of applications in an energy efficient manner. We observe that applications can be generally 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 sub-network, we identify a packet's sensitivity to network latency or bandwidth. To do this, we propose a new packet classification scheme that is based on an application's intrinsic network requirement property. For this, we use an application's *network episode length* and *height* metric to dynamically identify the communication requirements (latency/bandwidth criticality). Further, observing the property that not all applications are equally sensitive to latency or bandwidth, we propose a fine grain prioritization mechanism for applications within the bandwidth and latency optimized sub-networks.

Evaluations on a 64 core 2D mesh architecture considering 9 design alternatives with 36 diverse applications, show that our proposed two-layer heterogeneous network architecture outperforms all competitive monolithic network designs in terms of system/application performance and energy/energy-delay envelope. Overall, the primary contributions of this work are the following:

- We identify that a monolithic network design is sub-optimal when hosting applications with diverse network demands. As a step further, with extensive application level profiling, we identify that applications can be divided into two general classes in terms of their requirements from the network: bandwidth-sensitive and latency-sensitive.
- To exploit these two application classes, we propose a two-tier heterogeneous network architecture suitable for bandwidth and latency sensitive applications. For steering packets to an appropriate network, we propose a novel dynamic mechanism that utilizes the communication episodes of an application, called network episode length and height. These two metrics combined helps us to classify applications as latency or bandwidth sensitive. Further, using these two metrics, we classify applications into 9 sub-groups for the purpose of identifying an application's criticality within each of the bandwidth-sensitive or latency-sensitive class. This fine grain classification allows us to facilitate a prioritization mechanism for applications within the bandwidth and latency optimized sub-networks for further improving the performance. This dynamic ranking/prioritization scheme is shown to perform better than two recently proposed schemes.
- We show that our two-layer NoC design consisting of a 64b link-width latency optimized sub-network and a 256b link-width bandwith optimized sub-network, provides 5%/3% weighted/instruction throughput improvement over an iso-resource (320b link-width) monolithic network design, and consumes 47% lower energy compared to the iso-resource monolithic network. When compared to a baseline 256b link-width monolithic network, our proposed design provides 18%/12% weighted/instruction throughput improvement and consumes 16% less energy.

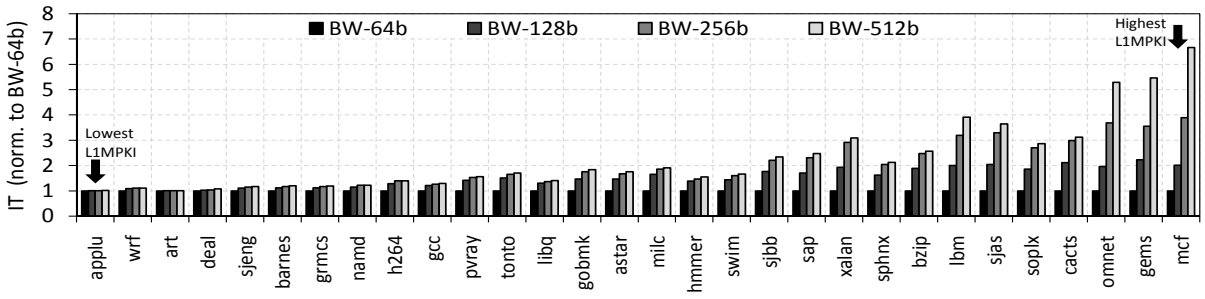



Figure 1: Instruction throughput (IT) scaling of applications with increase in network bandwidth.

# 2 Application-Driven Communication Characterization

As mentioned above, existing NoC designs are implicitly built on the paradigm that all the hosted applications place similar demands on the underlying network. In this paper, we argue against this paradigm by observing how different packets (even within the same application, but particularly across different applications) have vastly differing network resource demand and how each individual network packet impacts application-level performance. In this section, we contrast few observations that highlight the intrinsic heterogeneity in network demands across applications. These observations put together, provide the motivation for our application-aware design of NoC, which is described in Section 3. We start by looking at two of the fundamental parameters: network channel bandwidth and latency.

**Impact of channel bandwidth on performance scaling of applications**: Channel or link bandwidth is a critical design parameter that affects network latency, throughput and energy/power of the entire network. By increasing the link bandwidth, the packet serialization latency reduces, however increase in link bandwidth adversely affects a router crossbar power envelope. To study the sensitivity of an application to variation in link bandwidth, we perform a simple analysis. For this analysis, we use an 8x8 mesh network and run 64 copies of the same application on all nodes on the network[1].

Figure 1 shows the results of this analysis for 30 out of the 36 applications in our benchmark suite (6 applications are omitted to reduce clutter in the plots). We analyze scenarios, where we double the bandwidth starting with 64b links to 512b links (annotated as BW-64b, BW-128b, BW-256b and BW-512b in the figure). In this figure, the applications are shown on the X-axis in order of their increasing L1MPKI (L1 misses per 1000 instructions), i.e. `applu` has the lowest L1MPKI and `mcf` has the highest L1MPKI. The Y-axis shows the average instruction throughput when normalized to the instruction throughput of the 64b network.

Observations from this analysis are as follows: (1) Out of the 30 applications shown, performance of 12 applications (the rightmost 12 in the figure after `swim`) scale with increase in channel bandwidth. For these applications, an increase in 8x bandwidth results in at least 2x increase in performance. We call these applications *bandwidth sensitive* applications. (2) The rest 18 applications (all applications to the left of `swim` and including it), show very little to no improvement in performance with increase in network bandwidth. (3) Even for bandwidth sensitive applications, not all applications' performance scale 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 require more bandwidth from the network. But this intuition does not hold entirely true. For instance, `bzip` in spite of having higher L1MPKI than `xalan`, is less sensitive to bandwidth than `xalan`. Thus, we need a better metric to identify bandwidth sensitive applications.

**Impact of network latency on performance scaling of applications**: Next, we analyze the impact of network/router latency on the instruction throughput of these applications. Router pipeline critically affects the network throughput and also dictates the network frequency. To study the latency sensitiveness of applications, 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 1.5GHz for this analysis as well. Increasing the pipeline stages at

[1]The network is wormhole switched, uses deterministic X-Y routing, has 6 virtual channels per physical channel and 5-flit buffer depth. Each router in the network tile is connected to a core, a private L1 cache, and an 1MB per core shared L2 cache (Table 1). The network is clocked at the same frequency as the cores (1.5GHz). Table 2 mentions the application details.

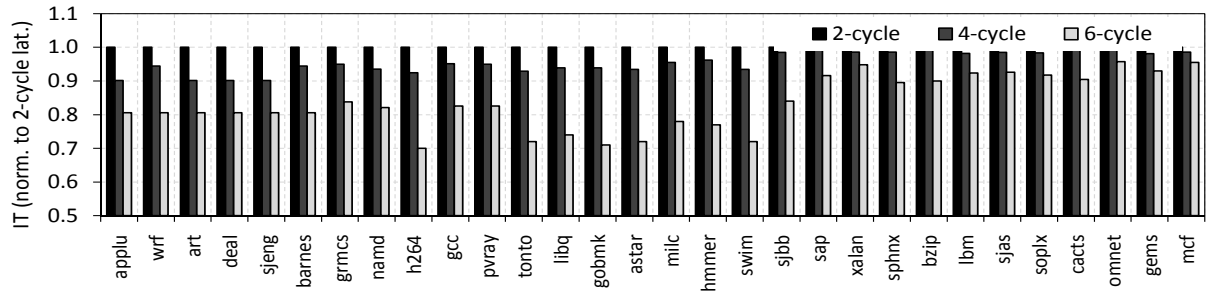



Figure 2: Instruction throughput scaling of applications with increase in router latency.

each router *mimics* additional contention at 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 are not very responsive to increase in network/router latency and on an 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 easily tolerated by these applications. (2) On the other hand, for all applications to the left of `swim` (and including `swim`), there is about 25% performance degradation when the router latency increases from 2-cycles to 6-cycles. These applications are clearly very sensitive to network latency and we call these *latency sensitive* applications. (3) Further, L1MPKI is not a good indicator of latency sensitivity (`hmmer` in spite of having higher L1MPKI when compared to `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 catering various application demands. Therefore, an alternative approach to designing an on-chip interconnect is to explore the feasibility of 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 wider and a low frequency network is suitable for bandwidth sensitive applications, while a narrow and high frequency network is best for latency sensitive benchmarks. To improve the performance of the latency sensitive applications, a network architect can, reduce the router pipeline latency from 2-cycles (our baseline) to 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 [13, 15], their design is often complex (involves speculation which can be ineffective at high or adverse load conditions) and requires sophisticated arbiters. Hence, while single cycle routers are certainly feasible, in this paper, we use frequency as a knob to reduce the network latency. Note, our analysis shows that, increasing the frequency of the network from 1.5GHz to 4.5GHz (3 times the core frequency) leads to less than 1.5% increase in energy for the latency sensitive applications (results for energy with frequency scaling is omitted for brevity).

Designing latency and bandwidth customized networks is one part of the solution space. We also need a mechanism to classify applications at runtime to one of the two categories: *bandwidth/latency sensitive* for guiding them to the appropriate customized network. In addition, since not all applications are equally sensitive to bandwidth or latency, we propose a fine grain prioritization of applications within the bandwidth and latency optimized sub-networks. This scheme further improves the overall application/system performance.

# 3 Application-Driven Approach for Designing NoCs

## 3.1 Dynamic classification of applications

The goal of identifying an application's sensitivity to latency/bandwidth, is to enable the network interface (NI) to inject or steer packets into a sub-network that has been optimized for either latency or bandwidth. We propose two novel metrics, called *episode length* and *episode height*, that effectively capture the latency and bandwidth demands of an application and help the NI to classify an application as either bandwidth or latency sensitive. We contrast the new metrics against two heuristics (L1MPKI [5] and Slack [6]), which were recently proposed to estimate a packet's criticality in the network.

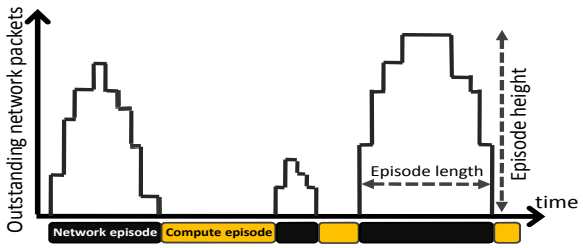



Figure 3: Network and compute episodes.

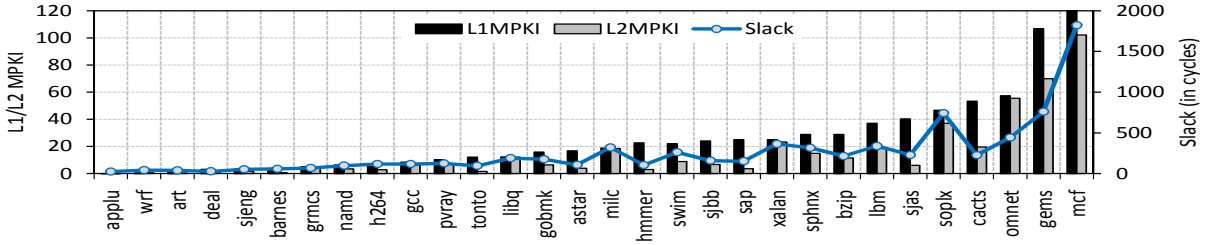



Figure 4: L1MPKI, L2MPKI and slack in applications.

**Episode length and height:** During an application's life cycle, the 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) [8, 19]. During this network phase, the processor is most likely to be stalling for the L2 and memory requests to be serviced. Because of this, the instruction throughput of the processor is low during this episode. During the compute episode, however, the instruction throughput is high. In this paper, we quantify 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 till 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 this average height, the processor 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. For example, if the episode lasts for 3 cycles and there are 2, 3 and 1 L1 misses in each of those cycles, then the average episode height is $\frac{2+3+1}{3} = 2$.

If an episode lasts for a very few cycles, intuitively it reflects that all packets belonging to this episode are very critical for the application to make progress. Any delay of packets belonging to this short lasting episode will delay the start of the following computation phase, and, thus the performance of the application will degrade. Hence, these packets are latency sensitive. On the other hand, if an episode is long lasting, the application is most likely tolerant to this long episode length, and delaying any packets belong to this episode will not degrade the performance much.

If an episode's height is short, it suggests that the application is likely to have low MLP in this episode and hence, its requests are likely to be very critical for the application to make progress. The packets belonging to this phase are likely to be latency sensitive. On the other hand, if an episode height is high, then the application has a large number of requests in the network, and the network latency of all those packets are overlapped. Large number of packets in the network means that the application most likely needs more bandwidth, but the network latency is not very critical for the application. Our analysis shows that, these two heuristics are least affected by the system state or network characteristics such as interference from other applications in the network. Therefore, these two metrics provide an intuitive, easy-to-compute, accurate and stable characterization of an application's network demand.

**Private cache misses per instruction (MPI):** This metric captures an application's network intensity. If the network intensity is lower, the application has low MLP and hence, its request are latency sensitive as opposed to bandwidth sensitive. Figure 4 shows the L1MPKI and L2 MPKI of several applications. We find that, MPI (or MPKI) can help in identifying latency sensitive applications from bandwidth sensitive ones. In Figure 4, 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

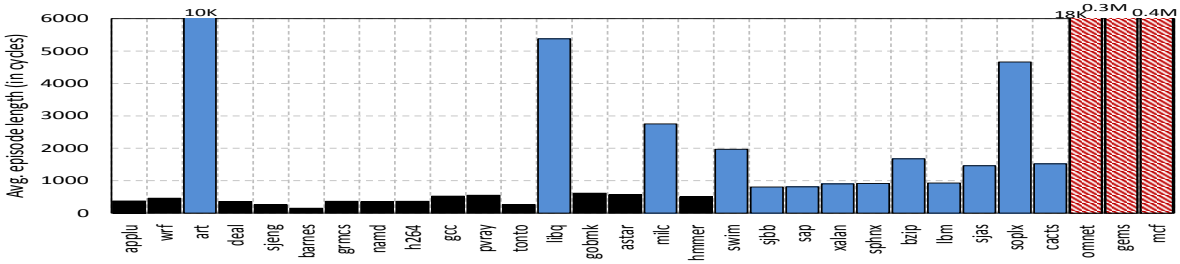



Figure 5: Average episode length (in cycles) across applications.

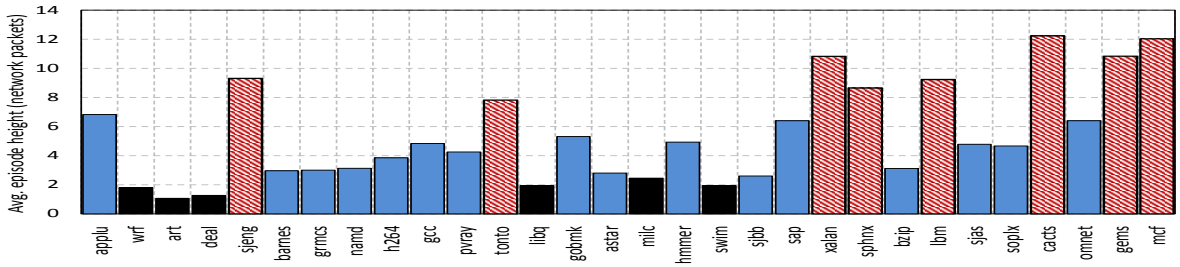



Figure 6: Average episode height (in packets) across applications.

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 accurate in estimating the criticality of applications *within* the latency sensitive class or bandwidth sensitive class. For instance, `bzip` in spite of having higher L1MPKI than `xalan`, is less sensitive to bandwidth than `xalan`. Similarly, `hmmer` and `swim`, in spite of having higher L1MPKI when compared to `gobmk` and `astar`, do not show proportional performance improvement with increase in bandwidth as the later applications show.

**Packet slack:** Slack, as a metric, was recently investigated [6] to identify a packet's criticality in the network. 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 4 shows how slack varies across applications. Intuitively, slack of a L1-miss instruction directly translates to the instruction's criticality in the network. Based on this, applications that have a longer slack are more tolerant to network delays when compared to applications that have smaller or no slack. Unfortunately, slack does not capture the MLP of an application and has low correlation in identifying increase in performance with increase in bandwidth/frequency. Also, slack is influenced by network contention.

To avoid short term fluctuations, we use running averages of the episode height and length to keep track of these metrics at runtime. Further, we quantify episode height as *high*, *medium* or *short* and episode length as *long*, *medium* and *short*. This allows us to perform a fine grain application classification based to episode length and height to classify them as either latency sensitive or bandwidth sensitive. Section 3.3 provides empirical data to support such a classification scheme. Figures 5 and 6 show these metrics for 30 applications in our benchmark suite. Based on Figures 1 and 2, we classify all applications whose 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 is larger than `sjbb`'s episode height but lower than 7 (empirically chosen) are classified as medium (shaded blue in the figures) and the remaining as high episode heights (shaded with hatches in Figure 6). Empirically, a cut-off of 10K cycles is chosen to classify applications as having medium episode length.

## 3.2 Analysis of episode length and height

Figure 7 shows the classification of applications based on their episode height and length. The figure also shows the bandwidth sensitive applications and the latency sensitive applications based on such a classification. In general, we classify applications having high episode height as bandwidth sensitive and vice-versa for latency sensitive.

| Classification | | Length | | |
|---|---|---|---|---|
| | | Long | Medium | Short |
| Height | High | 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 | High | 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 7: Application classification and ranking based on episode length and height.

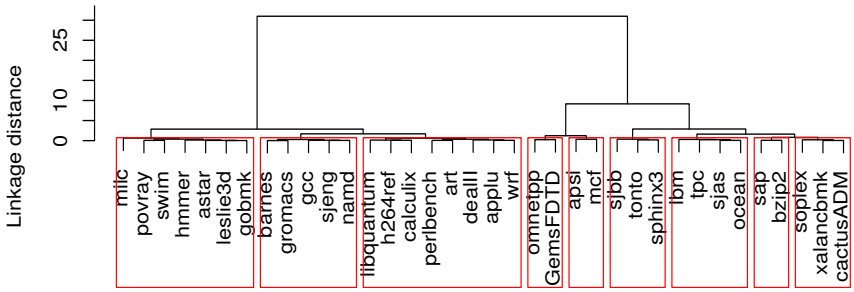



Figure 8: Hierarchical clustering of applications. The input to the clustering algorithm consists of improvement in IPC with bandwidth scaling (from 64b to 512b) and improvement in IPC with frequency scaling (1.5GHz to 4.5GHz).

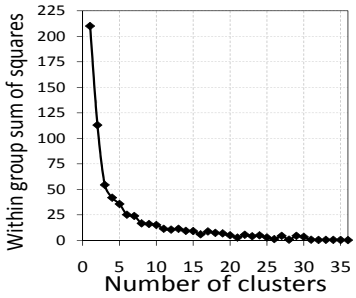



Figure 9: Reduction in within group sum of squares with increase in number of clusters.

## 3.3 Ranking of applications

We use the above fine-grained classification to rank applications for providing customized prioritization in a network. Essentially, applications whose episode length lasts longer, are prioritized the least in the network over other applications. Below, we discuss the steering and ranking of a few application classes and our intuition behind doing so.

**(1) Episode length is short and height is short**: Applications belonging to this category have very low MPKI and since their episode lasts for a very short period, delaying any packet is most likely to delay the start of the computation phase. This makes these applications highly latency sensitive and ranks them with the heights priority (rank 1). **(2) Episode length is short and height is high**: These applications are bursty, but for a very short period of time. Because of this burstiness, the packets' network latency are overlapped and hence, we classify these applications as bandwidth sensitive but rank them the highest in the bandwidth optimized sub-network (owing to their criticality to network latency because of a very short episode length). **(3) Episode length is long and height is short**: These applications are still latency sensitive, but are relatively latency tolerant compared to applications having medium/short episode length. So, these applications are prioritized the least (rank 4) in the latency optimized sub-network. **(4) Episode length is long and height is high**: These applications are the most bandwidth sensitive applications and owing to their large episode height, they are the most tolerant to network delay. Thus, these applications are classified as bandwidth sensitive and we prioritize them the least in the bandwidth optimized network.

Applications that do not belong to the above classes, have either latency or bandwidth sensitivity that lie within the extremes and are prioritized based on their relative tolerance to network delays when compared to others. Figure 7 shows the ranking of the applications in their respective sub-networks.

We took two critical decisions in our classifications - (1) choosing `sjbb`'s episode length and height as a threshold for short lasting episodes and episodes with smaller heights, and (2) choosing 9 smaller sub-classes after classifying the applications as bandwidth or latency sensitive. We next outline the empirical results that led us in taking these decisions.

**Rationality of our classification**: Figure 8 shows the results of a 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. In our case, the input to the clustering algorithm consists of the improvement in IPC with bandwidth scaling (from 64b to 512b) and improvement in IPC with frequency scaling (from

1.5GHz to 4.5GHz) i.e. values from Figures 1 and 2. The hypothesis behind this is to observe whether a clustering algorithm perceives noticeable difference between applications' performance with frequency and bandwidth scaling. We tried various linkage distance metrics like Euclidean distance, Pearson correlation and average distance between the objects, and in all cases the clustering was consistent with that shown in Figure 8 (shown for Euclidean distance). Although the eventual hierarchical cluster memberships are different from that shown in our classification matrix, the broader classification of how hierarchical clustering groups applications in bandwidth and latency sensitive clusters matches exactly with our classification scheme, which is based in episode height and length (with the exception of `sjeng`). The reason for `sjeng`'s misclassification is because its performance does not scale with bandwidth and hence, hierarchical clustering classifies it as a latency sensitive application. However, `sjeng`'s episode has a high episode height but a short episode length on average, meaning it is very bursty (and hence, high MLP) during a small interval of time. Because of this, we classify it as the highest ranking application in the bandwidth optimized sub-network.

**Why 9 sub-classes?** To answer this question, we measure the total within-group sum-of-squares (WG-SS) of the clusters resulting with hierarchical clustering. Figure 9 shows this metric as the number of clusters increase. The total WG-SS is a measure of the total dispersion between individual clusters and often regarded as a metric to decide the optimal number of clusters from a hierarchical or K-means algorithm [12, 20]. When all clustering objects are grouped into one cluster, the total WG-SS is maximum, whereas, if each object is classified as a separate object, the WG-SS is minimum (=0). Figure 9 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 chose 9 classes for our application classification and hence, sub-divided episode height and length into three quantitative class each.

# 4 Design Details

Since, we are using a canonical 2D network in our study, instead of discussing the standard router and network designs, we focus on the critical 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 these metrics, i.e. on every L1 miss, the NI computes the running average episode length/height. To compute episode height, 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, $B$.

When an NI of a local router receives a packet, it computes the episode length/height and based on the classification scheme mentioned in Section 3, decides which network this packet is to be steered. Further, the NI also tags the packet with its rank (2-bits) and it batch-id (3-bits). Note that, although the classification is static, each applications' rank and network sensitivity is decided at runtime. Thus, no central co-ordination is required in our technique to decide a uniform central ranking across all the applications in the system. Moreover, once a packet's ranking has been decided, it is consistently prioritized across the entire sub-network until it reaches its destination. At each router, the priority bits in the header-flit are utilized by the priority arbiters in a router to allocate VCs and the switch. Fast priority arbiters can be designed using high speed adders as comparators within the arbiters and our estimates (based on [21]) show that priority arbiters do not skew the pipeline latencies of a router.

To prevent priority inversion due to virtual channels (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]. With atomic buffers, a head-flit of a packet cannot occupy a particular VC unless the tail-flit of a packet occupying that VC has released it. Atomic buffers can lead to network under-utilization, but our experiments show that the performance loss due to this is very minimal.

**Handling starvation**: Prioritizing high ranked packets in a network may lead to starvation of low ranked packets. To prevent starvation, we combine our application-aware prioritization with a "batching mechanism" [5]. Each packet is added to 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 simply 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 [5, 6] to handle relative priorities inside a router.

**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 router and increase the router frequency up to 3 times(4.5Ghz) for the latency sensitive network. Note that the total network bandwidth depends on both link-width and frequency. Therefore, a designer could think of increasing the frequency of the bandwidth customized network (256b link-width) network to increase the total network bandwidth and consequently help the bandwidth sensitive applications. However, increasing the frequency of the wider 256b link network would adversely affect the power envelope of this network. Hence, in our network designs, we only increase the frequency of the narrow 64b link-width network whose power envelope is 40% lower when compared to the wider network. Consequently, the total network bandwidth of our baseline 256b network is 384 Gbps (256b*1.5GHz), of latency customized sub-network is 288 Gbps (64b*4.5GHz), and that of the bandwidth customized sub-network is 384 Gbps. Further, increasing the frequency of the latency customized sub-network, also increases this network's bandwidth (compared to no frequency scaling), but since we only steer latency sensitive applications (which are agnostic to bandwidth increase), into this sub-network, the performance improvement of these applications is primarily due to latency reduction in the network.

The design space for optimizing network latency or bandwidth is huge and is not possible to cover in this paper. The motivation of this paper is to demonstrate that a dual network with one optimized for latency and the other optimized for bandwidth is a better design than a monolithic or un-customized dual networks. The 64b 4.5GHz network for the latency sensitive applications and 256b 1.5GHz network for the bandwidth sensitive applications are just 2 of the design points to demonstrate the concepts.

# 5 Evaluation Methodology

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

❶ **1N-128**: In this configuration, there is a single homogeneous 128b link network. We assume this to be our starting point in network design since, starting with this monolithic network, we increase its bandwidth to create a bandwidth optimized sub-network, and reduce its bandwidth (and increase its frequency) to design a latency optimized sub-network.

❷ **1N-256**: In this configuration, there is a single homogeneous network with 256b link width. We chose this as our *baseline* network since starting with this network, we first design a homogeneous multiple sub-network design (where each sub-network has equal bandwidth, 2N-128x128) and then customize one sub-network for latency sensitive applications and the other sub-network for bandwidth sensitive applications.

❸ **2N-128x128**: This design has two parallel sub-networks, each with 128b link width. The buffer resources in each sub-network is half that of the baseline 1N-128 network and each of the sub-networks operate at the same frequency as that of the processors (= 1.5GHz). Packets are steered into each sub-network with a probability of 0.5 i.e., there is load balancing across the sub-networks.

❹ **1N-512**: This design has a single network with 512b link width. We call this a *high-bandwidth* configuration and is analyzed to see how our proposal fares when compared to a very high bandwidth network.

❺ **2N-64x256-ST**: In this design, there are two parallel sub-networks, one with 64b link width and the other with 256b link width. The buffering resources in each sub-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 sub-network and the latency sensitive packets are steered into the 64b sub-network. Each sub-network in this configuration is clocked at 1.5GHz (the frequency of the processors).

❻**2N-64x256-ST+RK(no FS)**: This design is the same as 64x256-steering network except that, in addition to steering the application packets into the appropriate sub-network, the network also prioritize applications based on their ranks (hence, the annotation **RK**) at every cycle in a router.

❼**2N-64x256-ST+RK(FS)**: This design is similar to above configuration except that the 64b sub-network is clocked at 4.5GHz (3x the frequency of processor). The 256b sub-network is still clocked at 1.5GHz. This configuration is analyzed to see the benefits of frequency scaling (hence, the annotation **FS**) the latency optimized network.

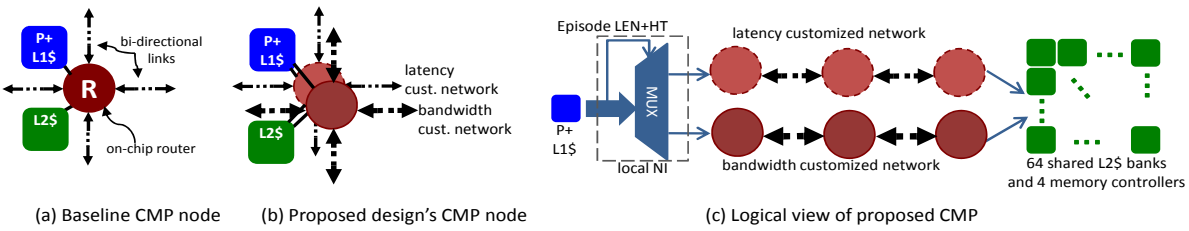



Figure 10: CMP schematics.

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

| | |
|---|---|
| Processor Pipeline | 1.5GHz processor, 128-entry instruction window |
| Fetch/Exec/Commit width | 2 instructions per cycle in each core; only 1 can be a memory operation |
| L1 Caches | 32 KB per-core (private), 4-way set associative, 128B block size, 2-cycle latency, write-back, split I/D caches, 32 MSHRs |
| L2 Caches | 1MB banks, shared, 16-way set associative, 128B block size, 3-cycle bank latency, 32 MSHRs |
| Main Memory | 4GB DRAM, up to 16 outstanding requests for each processor, 320 cycle access, 4 on-chip memory controllers |
| Network Router | 2-stage wormhole switched, virtual channel flow control, 6 VC's per port, 5 flit buffer depth, 1 flit/address packet |
| Network Topology | 8x8 mesh, each node has a router, processor, private L1 cache and shared L2 cache bank(all nodes)<br>4 memory controllers (1 at each corner node), 256b bi-directional links (= data packet's flit width). |

❽**1N-320(no FS)**: In this design, there is a single network with 320b (=64b+256b) bandwidth per link. The network operates at 1.5GHz. This configuration is iso-resource configuration when compared to all our 64x256 networks and is analyzed to see the benefits of our proposal over an equivalent iso-bandwidth configuration.

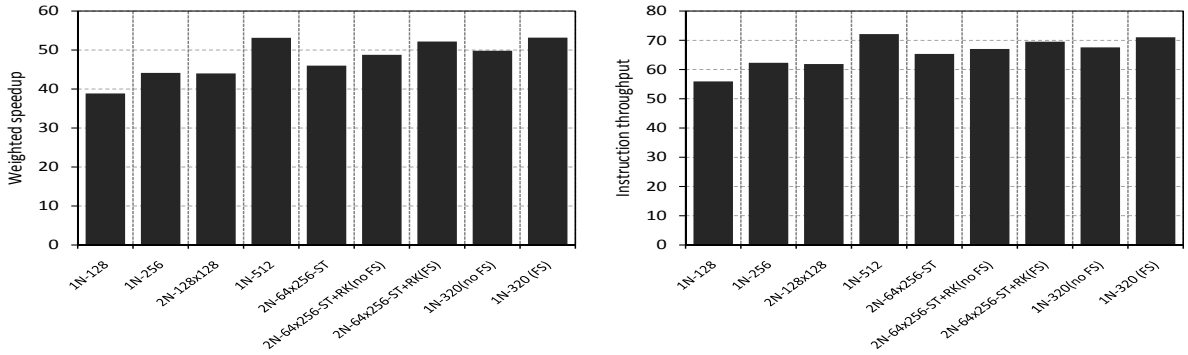
❾**1N-320(FS)**: This design is similar to the above design, except that the network is now clocked at 4.5GHz. This design is analyzed to see the effectiveness of our scheme over a scheme that is iso-resource as well as over-clocked to help latency sensitive applications.

**Experimental setup**: Our proposals are evaluated on a trace-driven, cycle-accurate x86 CMP simulator. Table 1 provides the configuration of our baseline, which contains 64 cores in a 2D, 8x8 mesh NoC. A single node of this configuration is shown in Figure 10 (a). Each core has a private write-back L1 cache. The network connects the cores, shared L2 cache banks, and memory controllers. Each router uses a state-of-the-art two-stage pipeline. We use the deterministic X-Y routing algorithm, finite input buffering, wormhole switching, and virtual-channel flow control. A data packet consists of 1024b (= cache line size) and is decomposed into 8 flits in the baseline design (with 128b links). Since wiring resources on die are abundant [1, 4, 17, 3], when simulating parallel networks, we assume the sub-networks to be implemented in the same 2D substrate as the cores (a single node of this configuration is shown in Figure 10 (b) and the resulting logical view of the CMP is shown in Figure 10 (c)). The dynamic and leakage energy numbers for the network were extracted using Orion 2.0 [10] and incorporated into our simulator for detailed network energy analysis. Based on Orion 2.0 estimates, the area of two sub-networks (router and links) consisting of 256b and 64b links is just 1% larger than an iso-resource 320b links network (2.4X larger area when compared to baseline 128b link network), and the power envelope of these two sub-networks is 20% lower than the iso-area network (2.3X higher power when compared to baseline 128b link network). The various counter bits and parameters used in our techniques are: (1) counter size for number of cycles in a network phase, $M$ = 14bits (2) batching interval, $B$ = 16,000 cycles (3) batching levels, $L$ = 8.

**Application characteristics**: 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 workloads traces (`sap`, `tpcc`, `sjbb`, `sjas`) totalling 36 applications. All our experiments analyze multiprogrammed workloads, where each core runs a separate application. We simulate at least 320 million instructions across 64 processors (minimum 5 million instructions per core). Table 2 characterizes our application suite. The reported parameters are for the applications running alone on the baseline CMP system without any interference. The table shows application characteristics based on network load intensity (high/low), episode height (high/medium/short), episode length (long/medium/short) and the fraction of execution time spent in network episodes. 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 that 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. In Section 6 we analyze the sensitivity of our scheme when the percentage of latency/bandwidth applications vary in

Table 2: Application characteristics when run on the baseline (Load: High/Low depending on network injection rate, Episode height: High/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 | # | Benchmark | Load | Episode height | Episode length | Net. fraction |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | applu | Low | Medium | Short | 8.23% | 19 | ocean | Low | Medium | Medium | 90.10% |
| 2 | wrf | Low | Short | Short | 9.42% | 20 | hmmer | Low | Medium | Short | 66.03% |
| 3 | perlbench | Low | Medium | Short | 8.78% | 21 | swim | Low | Short | Medium | 41% |
| 4 | art | Low | Short | Medium | 82.33% | 22 | sjbb | High | Medium | Medium | 87.29% |
| 5 | dealII (deal) | Low | Short | Short | 27.92% | 23 | sap | High | Medium | Medium | 88.89% |
| 6 | sjeng | Low | High | Short | 28.41% | 24 | xalancbmk (xalan) | High | High | Medium | 89.86% |
| 7 | barnes | Low | Medium | Short | 72.51% | 25 | sphinx3 (sphnx) | High | High | Medium | 83.92% |
| 8 | gromacs (grmcs) | Low | Medium | Short | 48.58% | 26 | bzip2 (bzip) | High | Medium | Medium | 84.90% |
| 9 | namd | Low | Medium | Short | 51.60% | 27 | lbm | High | High | Medium | 81.10% |
| 10 | h264ref (h264) | Low | Medium | Short | 61.45% | 28 | sjas | High | Medium | Medium | 89.47% |
| 11 | calculix | Low | Medium | Short | 48.21% | 29 | soplex (soplx) | High | Medium | Medium | 81.23% |
| 12 | gcc | Low | Medium | Short | 47.55% | 30 | tpc | High | Medium | Medium | 86.82% |
| 13 | povray (pvray) | Low | Medium | Short | 59.56% | 31 | cactusADM (cacts) | High | High | Medium | 82.33% |
| 14 | tonto | Low | High | Short | 52.99% | 32 | leslie3d | High | Short | Long | 99.70% |
| 15 | libquantum (libq) | Low | Short | Medium | 99.00% | 33 | omnetpp | High | Medium | Long | 92.62% |
| 16 | gobmk | Low | Medium | Short | 64.88% | 34 | GemsFDTD | High | High | Long | 97.26% |
| 17 | astar | Low | Medium | Short | 82.78% | 35 | apsi | High | Medium | Long | 95.15% |
| 18 | milc | Low | Short | Medium | 88.16% | 36 | mcf | High | High | Long | 99.18% |




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

(b) Instruction throughput (IT) (application throughput)

Figure 11: Performance comparison across various network designs with multiprogram mixes.

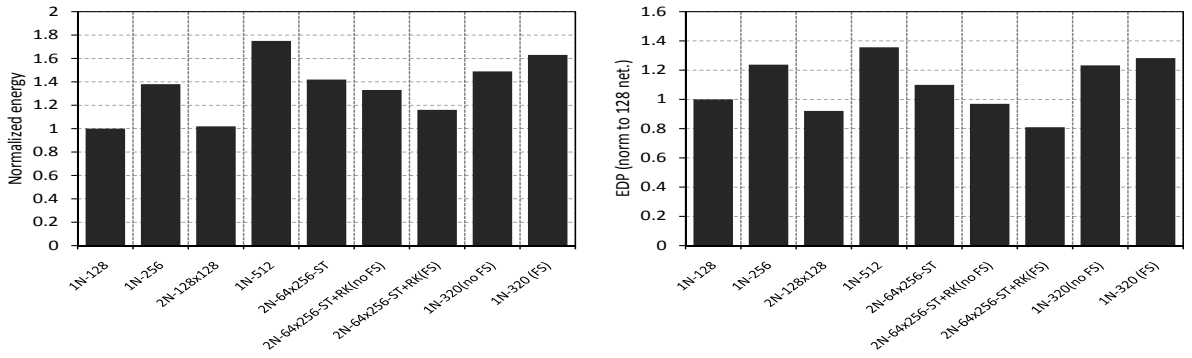
a workload.

**Evaluation metrics**: Our primary performance evaluation metrics are instruction throughput and weighted speedup. Instruction throughput is defined to be the sum total of the number of instructions committed per cycle (IPC) in the entire CMP and is considered as an *application throughput* metric [7]. The weighted speedup metric [18] sums up the slowdown experienced by each application in a workload, compared to its stand alone run under the same configuration and is widely regarded as a *system throughput* metric [7].

# 6 Analysis of Results

**Performance comparison**: Figure 11 shows the performance comparison across the various network designs. The following observations are in order:

- Two 128b sub-networks (2N-128x128) provide similar performance (both system and application throughput) as compared to a bandwidth equivalent single monolithic network with 256b link width (1N-256). This is in spite of the increase in packet serialization in the sub-networks. The primary reason for this performance improvement is reduction in congestion across each sub-network (each sub-network now sees 50% less packet) when compared to a monolithic wider network. The total bandwidth in the 2N-128x128 design is the same as 1N-256 design and hence bandwidth sensitive applications performance is not affected. One the other hand, the performance of latency sensitive applications is improved because of the load balancing (reduced congestion is each sub-network), and thus,




(a) Energy consumption of different networks

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

Figure 12: Energy and EDP comparison across various network designs (all results normalized to 128 net.).

the degradation in performance due to serialization latency increase is compensated by improvement in performance due to reduced congestion.

- Bandwidth and latency optimized parallel sub-networks operating at the same frequency as the processor along with steering of packets based on their bandwidth/latency sensitivity (2N-64x256-ST) provides 4.3%/5% system/application throughput improvement, over the baseline (2N-256) design. By providing bandwidth sensitive applications more bandwidth (when compared to 1N-128) and reducing the congestion when compared to a monolithic network, the performance of both bandwidth and latency sensitive applications are improved. Prioritizing and ranking packets based on their criticality after steering them into a sub-network (2N-64x245-ST+RK(no FS)) provides an additional 6%/3% improvement in system/application throughput, over the 2N-64x256-ST design when compared to the baseline design. This is because, our ranking scheme prioritizes the more (relatively) network-sensitive applications in each sub-network, and ensures no starvation using batching.
- Frequency scaling the latency/bandwidth sub-network along with steering and ranking the applications (2N-64x245-ST+RK(FS)) provides the maximum performance improvement among our proposals: 18%/12% system/application throughput improvement over the baseline network. With frequency scaling, the latency optimized sub-network is clocked at a higher frequency, accelerating the latency sensitive packets and this brings the additional benefits in performance.
- Frequency scaling the sub-networks and steering along with ranking of applications (2N-64x245-ST+RK(FS)) is better than an iso-resource network (1N-320(no FS)) by 5%/3% in weighted/instruction throughput. The performance of 2N-64x245-ST+RK(FS) is within 2.0%/2.2% (system/application throughput) of the high frequency iso-resource network with frequency increased by 3x (1N-320(FS)). Frequency scaling the 320b link width network helps latency sensitive applications and more bandwidth (when compared to 256b link width) helps the bandwidth sensitive applications. But as will be shown shortly, the energy consumption of such a network is higher when compared to our proposal.
- Our proposed network (2N-64x245-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 benefit for latency sensitive applications. Additionally, as will be shown next, a wide-channel 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 high bandwidth network, it does so at a lower energy envelope.

**Energy and EDP comparison**: Increasing the channel bandwidth decreases the serialization (and zero-load) latency and hence, end-to-end latency is reduced. However, increasing the channel bandwidth also affects router crossbar power quadratically. Figure 12 shows the energy and energy-delay product (EDP) of the applications across the 9 designs. We find that:

- 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 sub-networks design (2N-128x128) has similar energy consumption as a single 128b link monolithic network. The energy reduction going from one network to two sub-networks comes primarily from reduction in network latency (by reducing the congestion in each sub-network). In fact, we observed that the energy consumption of two parallel sub-networks, each with channel width $\frac{N}{2}$, is always lower than a single network with channel width $N$.

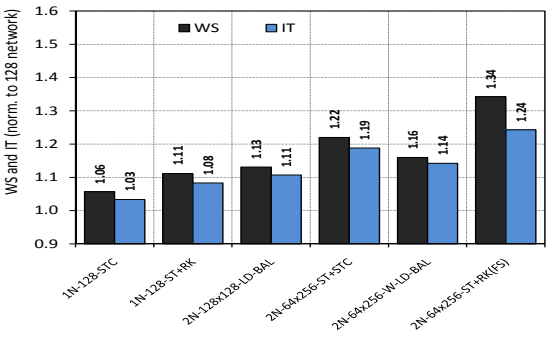



Figure 13: Weighted speedup (WS) and instruction throughput (IT) when compared to state-of-the art design (all results normalized to 1N-128 net.).

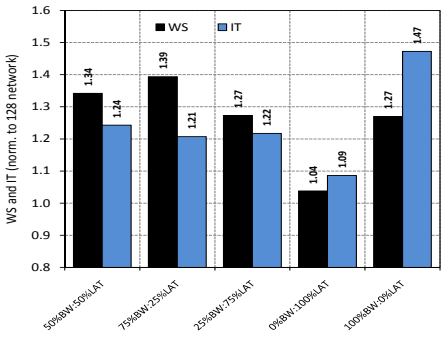



Figure 14: Performance comparison when varying proportion of bandwidth/latency intensive applications in each workload.

- The average energy consumption of a high bandwidth network with 512b links (1N-512) is 75% higher than a 128b link network and 26% higher than the 256b link network. When link width increases, although serialization latency reduces, the crossbar power starts to dominate the energy component.
- Steering packets along with application prioritization in the routers (2N-64x256-ST-RK(no FS)) reduces energy consumption by 6.7% when compared to just steering packets (2N-64x256-ST). Amongst our proposed designs, steering along with ranking in frequency scaled sub-networks (2N-64x256-ST-RK(FS)), consumes 16% lower energy than the 2N-128 network . This is 59% lower energy when compared to a high-bandwidth network (1N-512) and 47% lower energy than an iso-resource network which is frequency scaled (1N-320(FS)). Overall, our proposed scheme consisting of heterogeneous parallel sub-network architecture always consumes lower energy than an iso-resource 320b link width network, and the 2N-64x256-ST network.
- When comparing EDP metric, steering along with ranking in frequency scaled sub-networks (2N-64x256-ST-RK(FS)) design is 35% better than the baseline design(1N-256). This is because, our scheme reduces network latency significantly and this lowers the delay component in EDP metric. Even without frequency scaling, the 2N-64x256-ST-RK(no FS) design has 22% lower EDP than the baseline design. Again, our proposed schemes always have lower EDP than a high-bandwidth network (1N-512) or an iso-resource 320b link network or the iso-resource 2N-64x256-ST network.

  Thus, compared to a 256b link monolithic network operating at 1.5GHz, we find a 64b sub-network operating at 4.5GHz and a 256b sub-network operating at 1.5GHz is an optimal design from performance, energy and EDP perspective.

  **Reply packets from L2 cache (DRAM) to L1 cache (L2 cache)**: In all the above evaluations, we routed the L2 cache (DRAM) replies to the L1 cache (L2 cache) in either the 64b or the 256b sub-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 sub-network and vice-versa. Reply packets are L1/L2 cache line sized packets (1024b) and transmitting them over the 64b network increases their serialization latency. However, the 64b sub-network is relatively less congested when compared to the 256b sub-network (because of lower injection ratio of latency sensitive applications) and since the 64b sub-network is clocked at 3x frequency, the network latency in this sub-network is lower. Our analysis shows that, transmitting *all* the reply packets in the 256b network increases the system/application 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 latency sensitive packets, we always route them in the 64b high frequency sub-network.

**Comparison with prior works**: A previous work by Das et al. [5] proposed a ranking framework, called STC, that is based on criticality of a packet in the network. In this work, the authors use L1MPKI as a heuristic to estimate the criticality of a packet and based on this, propose a ranking framework which 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 batch boundaries, and communicates these information to each node. Apart from performance benefits, the authors also show that STC is better in terms of fairness when compared to the round-robin arbitration often employed in routers. Since, we also prioritize applications in the network, we compare our scheme with STC below. When comparing with STC for a single network design, we utilize a 2-level

ranking scheme when using our technique. The first level ranking prioritizes latency sensitive applications over bandwidth sensitive applications, and then among the latency and bandwidth sensitive applications, we use episode width and height to rank the applications (based on ranking in Figure 7).

Another recent work by Balfour and Dally [1] showed the effectiveness of load-distributing traffic *equally* over two parallel sub-networks. In this work, each of the sub-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. Since, we also propose parallel sub-networks (although our design shows heterogeneous networks are better than homogeneous), we compare our scheme with a similar load-balancing scheme proposed as by Balfour and Dally [1].

Figure 13 shows the results, where we compare the performance and fairness 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 STC work). The STC schemes are annotated as **-STC** with a given network design and the load-balancing schemes are annotated as **-LD-BAL** in the figures. The overall performance improvement with STC is 6%/3% (system/application) in a single 128b link monolithic network when compared to 1N-128. Compared to this, our 2-level ranking scheme shows 11%/8% system/application throughput improvement over 1N-128 design. Since STC uses L1MPKI to decide rankings, and as shown earlier, L1MPKI is not a very strong metric to decide the latency/bandwidth criticality of applications. Moreover, when using L1MPKI, STC does not take into account the *time* factor i.e how long in cycles does an application has this L1MPKI. Our proposed episode length captures this factor, and hence, can differentiate between two applications having similar episode height (L1MPKI in the context of STC) from each other. Based on this, our design ranks an application with shorter episode length higher than an application with longer episode length, hence capturing the *true* criticality of these packets. Even when comparing our scheme (2N-64x256-ST-RK(FS)) with that of STC in a two parallel network design (2N-64x256-ST-STC) (where applications are first steered into the appropriate network and then ranked using STC), we see an additional 12%/5% (system/application) benefit over the STC based design. Moreover, in terms of fairness (harmonic speedup results omitted for brevity), our scheme is 4% and 2% better than STC in a single and multiple parallel network design, respectively. Further, in our scheme the rankings are determined dynamically when the packet enters into each sub-network and there is no requirement of a dynamic co-ordination scheme to decide rankings as is required by STC scheme. So, our scheme not only has lower overhead, but also exhibits better performance and fairness compared to STC.

Since we propose heterogeneous sub-networks, when load balancing between two sub-networks we steer packets in the weighted-ratio of $\frac{256}{256+64}$ and $\frac{256}{256+64}$ between the 256b and the 64b sub-network. This scheme is annotated as **-W-LD-BAL** in the Figure 13. Our evaluations show that, steering packets with equal probability into each network leads to more congestion the 64b link sub-network and under-utilizes the 256b sub-network. We find that our proposal (2N-64x256-ST-RK(FS)) has an additional 18%/10% (system/application) throughput improvement over the weighted load-balancing scheme (2N-64x256-W-LD-BAL). Load balancing scheme is oblivious to the sensitivity or criticality of packets. With this scheme, a latency sensitive packet is steered into the bandwidth optimized network with a probability of 0.8 and bandwidth sensitive packet is steered into the latency sensitive network with a probability of 0.2 and, thus in both these cases performance either does not improve (for the former) or degrades (with the later). Further, with weighted load-balancing, there is negligible improvement in fairness whereas, in our scheme the fairness of the system improves by 19% over the baseline network. Overall, we believe that with heterogeneous sub-networks, load balancing is a sub-optimal scheme and that intelligently steering packets based on their sensitivity and criticality can lead to significant performance benefits.

**Sensitivity to distribution of bandwidth-latency applications in the workload:** All results shown till now had a multiprogram mix with equal percentage of latency and bandwidth sensitive applications. To analyze the sensitivity of our scheme across various application mixes, we varied the bandwidth sensitive application mix in a workload from 100%, 25%, 50%, 75% to 0%. Figure 14 shows the results of this analysis (the results have been normalized to 128b monolithic network since some of our 256b experiments with varying workload mixes are still running at the time of submission of this paper). We find that our proposal, in general, has higher system/application throughput across the entire spectrum of workload mix. However, the benefits are small (4%/9% system/application throughput improvement over 1N-128 design) when the system has 100% latency sensitive applications. When the application mix is skewed (i.e. system has *only* bandwidth *or* latency sensitive applications), we have assumed an oracle knowledge, and weighted-load balanced both the sub-networks. As such, with 100% latency sensitive applications in the workload mix, benefits arise only due to load distribution and the benefits are minimal in this case. Without this load balancing, the benefits of our proposal will only be because of ranking. We are currently working on a scheme that

can dynamically measure this skew and can then steer packets to the second sub-network.

# 7 Related Work

We have already qualitatively compared our scheme with Balfour and Dally's proposal [1] and showed that our scheme is significantly better than a load balancing (even weighted load balancing) scheme. Other works that have proposed multiple networks for NoCs include TRIPS [17], RAW [14], Tilera [3], and IBM cell [9]. 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) and 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. Even DASH multiprocessor [2] had multiple networks (request and reply meshes), but the design was meant to eliminate request-reply deadlocks. Tilera's iMesh network consists of five separate networks to handle memory access, streaming packet transfers, user data, cache misses, and interprocess communications. Among these five networks, there is only *one* network where the processor (user) gets to send data from cores to caches (and vice-versa). Each of the five networks are based on *sizes of packets from a source to a destination*. In contrast, our proposal sub-divides a network to customize for latency/bandwidth sensitive traffic and then rank traffic based on *criticality of packets*.

The concept of compute and non-compute epidodes has been used in ATLAS [11]. ATLAS defines *memory* episode length to be the duration when an application is waiting for at least one memory request (i.e. L2 miss). We use the concept of *network* episode which is the duration when the application is waiting for at least one L1 miss. ATLAS exploits the notion of memory episode to prioritize threads with least attained memory service time. On the contrary, we use network episodes to classify applications. Further, we use the notion of network episode *height* to capture applications MLP which no previous works have done.

# 8 Conclusions

Recently, design and analysis of NoCs has gathered significant momentum because of the criticality of the communication substrate in designing scalable, high performance and energy efficient multicore systems. However, most NoCs have been designed in a monolithic manner without considering the actual application requirements. We argue that such an approach is sub-optimal from both the performance and energy standpoints and propose an application driven approach to design NoCs. Based on the characterization of several applications, we observe that a heterogeneous NoC consisting of two separate networks, one optimized for bandwidth and other for latency, can cater to the applications' requirement more effectively.

We evaluate the effectiveness of this two-layer network over a range of monolithic designs. Evaluations with 36 benchmarks on a 64-core 2D architecture indicate that a two-layer heterogeneous network approach consisting of a 256b link (64b link) bandwidth (latency) optimized network provides 18%/12% system/application throughput improvement over a 256b link network and is 5%/3% better in weighted/instruction throughput, while consuming 47% lower energy compared to an iso-resource (320b link) network (59% lower energy when compared to a very high bandwidth 512b link network). In a combined performance-energy design space, the proposed application-driven NoC outperforms all competitive monolithic network designs. In conclusion, while multiple on-chip networks have been proposed in the literature, none of these are based on a systematic, application-driven approach like ours. Also, the proposed communication episode based classification and ranking schemes are significantly better than state-of-the-art NoC prioritization mechanisms.

# References

[1] J. Balfour and W. J. Dally. Design Tradeoffs for Tiled CMP On-Chip Networks. In *ICS*, 2006. 10, 14, 15

[2] D. Lenoski et al. The Stanford Dash multiprocessor. *Computer*, 1992. 15

[3] D. Wentzlaff et al. On-Chip Interconnection Architecture of the Tile Processor. *Micro, IEEE*, 2007. 10, 15

[4] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *DAC'01*. 10

[5] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Application-Aware Prioritization Mechanisms for On- Chip Networks. In *MICRO*, 2010. 4, 8, 13

[6] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Aergia: Exploiting Packet Latency Slack in On-Chip Networks. In *ISCA*, 2010. 4, 6, 8

[7] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. *Micro, IEEE*, 2008. 11

[8] B. Fields, S. Rubin, and R. Bodík. Focusing Processor Policies via Critical-Path Prediction. In *ISCA'01*. 5

[9] J. A. Kahle et al. Introduction to the Cell Multiprocessor. *IBM J. of Research and Development*, 2005. 15

[10] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *DATE*, 2009. 10

[11] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers. In *HPCA*, 2010. 15

[12] W. J. Krzanowski and Y. T. Lai. A Criterion for Determining the Number of Groups in a Data Set Using Sum-of-Squares Clustering. *Biometrics*, 44(1), 1988. 8

[13] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. K. Jha. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *ICCD*, 2007. 4

[14] M. B. Taylor et al. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, 2002. 15

[15] R. Mullins, A. West, and S. Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *31st ISCA*, 2004. 4

[16] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers. In *MICRO*, 2006. 8

[17] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore. Exploiting ILP, TLP, and DLP with The Polymorphous TRIPS Architecture. In *ISCA*, 2003. 10, 15

[18] A. Snavely and D. M. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor. In *ASPLOS*, 2000. 11

[19] S. Subramaniam, A. Bracy, H. Wang, and G. H. Loh. Criticality-Based Optimizations for Efficient Load Processing. In *HPCA*, 2009. 5

[20] R. Tibshirani, G. Walther, and T. Hastie. Estimating the Number of Clusters in a Data Set via the Gap Statistic. *Journal of the Royal Statistical Society.*, 63(2), 2001. 8

[21] V. G. Oklobdzija and R. K. Krishnamurthy. *Energy-Delay Characteristics of CMOS Adders (Chapter-6), High-Performance Energy-Efficient Microprocessor Design*. Springer, 2006. 8