Adaptive Cluster Throttling: Improving High-Load Performance in Bufferless On-Chip Networks

Rachata Ausavarungnirun* rausavar@ece.cmu.edu
Kevin Kai-Wei Chang* kevincha@andrew.cmu.edu
Chris Fallin* cfallin@cmu.edu
Onur Mutlu onur@cmu.edu

(* primary student authors)

Computer Architecture Lab (CALCM)
Carnegie Mellon University

SAFARI Technical Report No. 2011-006

September 6, 2011

Abstract

Higher core counts and increasing focus on energy efficiency in modern Chip Multiprocessors (CMP) have led to renewed interest in simple and energy-efficient Network-on-Chip (NoC) designs. Several recent proposed designs trade off network capacity for efficiency, based on the observation that traditional networks are overprovisioned for many workloads. Bufferless routing is one such example.

However, when the application workload requires high interconnect performance, the inefficiencies of bufferless interconnects can cause significant performance degradations. Previous work has tackled various issues with bufferless routing, but little work has been done to improve performance at high network load. Fundamental improvements in bufferless network performance at high load could extend the benefits of lower energy and smaller die area to a wider range of potential applications.

In this work, we propose *ACT* (Adaptive Cluster Throttling), a source-throttling mechanism that provides better system performance and fairness than the best current mechanisms on bufferless networks. By batching applications into clusters, and alternately throttling different clusters, ACT provides a chance for all applications to inject traffic into the network while maintaining control over total network load. We show 11.9% (10.2%) system performance gain on average with 14.5% (15.1%) improvement in fairness over 60 network-intensive workloads on a 4x4 (8x8) bufferless NoC. At high network load, ACT achieves nearly half the performance gain over a bufferless baseline that a conventional buffered network achieves, while reducing network power by 15.4% (5.4%).

1 Introduction

In recent years, significant work has examined *Networks-on-Chip* (NoCs) as a solution to scalability challenges in more traditional interconnect solutions, such as busses. As the number of cores grows into the hundreds, designs such as the 2D mesh allow for scalability with simple routing algorithms and low-radix router designs.

However, on-chip interconnect is typically limited by tight constraints on power consumption and die area. Because of this, some recent work has examined alternative designs that make different design decisions from traditional buffered virtual channel (VC) routers in order to gain more efficiency. One such design is *bufferless routing* [12, 14, 17, 27], in which the router buffers are completely eliminated, and router port contention is resolved by temporarily misrouting, or dropping and retransmitting, packets. This trade-off yields significant network power savings with minimal performance loss under low-to-medium network intensity workloads. However, bufferless networks perform significantly worse than a traditional buffered network with network-intensive workloads, because network inefficiency due to deflections and a lower in-flight capacity reduce the saturation bandwidth. Previous work [12, 27] notes that bufferless NoCs are a compelling design choice for low-to-medium network load because of significantly reduced power and die area. However, the drop in performance at high network load from a buffered to a bufferless network, which we call the *performance gap* in this paper, remains a hurdle for widespread adoption in general-purpose systems.

There are at least two general approaches to bridge this performance gap. The first possibility is to build a *hybrid network* that adaptively switches to a higher-capacity mode with buffers when the low-cost bufferless interconnect becomes saturated. Adaptive Flow Control [18] combines a bufferless deflection router with a traditional VC-buffered router, and allows each router to switch modes independently. While such an approach can attain the energy efficiency of bufferless design at low load, and the performance of buffered networks at high load, it does not fundamentally improve either underlying design. When running at high load, a hybrid system will spend a portion of its time in buffered mode, requiring higher power consumption and eliminating the full power reductions of bufferless design. Furthermore, while power gating can allow a hybrid scheme to enjoy the power and energy benefits of bufferless designs at low load, it imposes some overhead, and incorporating both bufferless and buffered designs side-by-side imposes an area penalty because buffers remain present.

An alternative approach is to improve the efficiency of bufferless deflection routing directly. One previous work [29] uses *source throttling* (limiting the network request rates of applications) in order to reduce in-network contention in bufferless networks, thereby reducing deflection rates and improving overall system throughput. Source throttling is an effective technique to enhance performance without switching to more expensive designs, such as buffered routers, under high load. Furthermore, it is orthogonal to hybrid designs, and may enable such a design to scale to higher loads on a cheap interconnect before beginning to use energy-consuming buffers.

In this paper, we propose a new throttling mechanism, *ACT* (Adaptive Cluster Throttling). Based on the observation that different applications impose different portions of total network load, and should be treated differently, ACT partitions applications into *clusters* (subsets) once per epoch. Clusters are *throttled* (limited in request rate) in a timesliced schedule: some clusters are throttled only during some timeslices throughout the epoch. Doing so grants the applications in these clusters some unthrottled access to the network, periodically making fast progress with less interference from the other clusters that are currently throttled. Furthermore, timeslicing splits network access among applications fairly without the need to adjust sensitive throttling rates individually on each application. ACT brings the performance of bufferless networks closer to buffered networks while actually reducing network power due to reduced network utilization. ACT addresses the performance gap by more efficiently making use of a bufferless NoC's capacity. We make the following contributions:

- ACT, a throttling mechanism for bufferless NoCs that groups applications into *clusters* according to application traits, and then alternately throttles clusters in timeslices.
- Evaluations showing improvement in system performance and fairness over a bufferless baseline, for a variety of workloads, with comparisons to one prior proposed work [29], a homogeneous throttling mechanism, and an adaptively-buffered network [18]. ACT closes 46.4% of the performance gap between bufferless and buffered 4x4-networks with an 11.9% system performance gain on average.

2 Background

2.1 Network-on-Chips

The network-on-chip (NoC) is the most commonly proposed solution to scale interconnect in chip multiprocessors (CMPs) beyond the limits of more traditional bus and crossbar interconnects [1, 7]. In a typical cache-coherent CMP, the NoC carries data requests and responses between private (L1) caches, shared (L2) caches, and memory controllers. It is thus on the critical path of every cache miss.

A significant body of research in the past decade has gone into mechanisms and new designs to improve the scalability of NoCs: for example, there has been work on improved topology [16, 20] and on congestion control [15]. Additional work has investigated how to build more energy-efficient routers by simplifying the microarchitecture [19] or removing buffers in the routers and misrouting [12, 24, 26, 27] or dropping [14, 17] traffic upon contention. Removing buffers in particular yields a large reduction in die area and network power [12, 27]. We will now examine this design direction in more detail.

2.2 Bufferless NoCs

Bufferless design is of particular interest because it optimizes the network for the common case when it is not heavily loaded, and leads to significant energy and die-area savings relative to more traditional designs that use in-router buffers. Bufferless NoCs have been proposed as an alternative option to the traditional virtual-channel (VC) buffered routers [4, 6]. Whereas these buffered routers must buffer packets at every router input until they win arbitration for a router output, deflection-based bufferless routers move packets continuously. The *key idea* of a such a design is to deflect, or misroute, some packets (or their individual components, *flits*) to alternate output ports when traffic contends. The packets will still eventually arrive at their destinations due to properties of the arbitration [27]. Thus, the network links serve as in-flight buffers, and local contention will cause traffic to naturally spread to nearby links. BLESS [27] motivates this scheme, bufferless deflection routing, by the energy and area savings obtained when removing buffers. CHIPPER [12] examined bufferless deflection routing at the router microarchitecture level in order to make implementation practical. Together, these works provide a compelling proposal for low-to-medium network load, where their performance degradation relative to a buffered network is minimal.

However, when network load rises above a certain point, bufferless networks begin to suffer in performance (and also in energy efficiency). This is due first to increased deflections in the network, leading to higher average latency, and second to a fundamentally lower network saturation point due to smaller in-flight traffic capacity. The lower saturation bandwidth leads to lower system performance when multiple network-intensive applications are running in the system. This is a case of *network congestion*. Moscibroda and Mutlu [27] claim that many workloads in a real system have network utilizations low enough that a bufferless network's capacity is sufficient. However, we observe that for real workloads that include network-intensive applications, there is often a significant *performance gap* between bufferless and buffered networks, as shown in §3. This performance gap, and the related problem of poor fairness (worst-case application slowdown), could hinder the adoption of bufferless networks as an efficient interconnect in general-purpose systems. We will now examine the problem in more detail before proposing a cluster-based throttling mechanism to tackle the issue in a new way.

3 Motivation: Performance and Fairness at High Load

As we have introduced, bufferless routing presents the opportunity for significant energy and area reductions at the cost of decreased network capacity. This decreased capacity can significantly degrade performance when network load is high. Figure 1 demonstrates the *performance-gap problem* quantitatively. Two network designs are evaluated: a traditional virtual-channel buffered NoC, and a bufferless NoC, FLIT-BLESS [27]. The top panel shows system performance, measured as normalized weighted speedup (methodology in §5). Workloads (formed from SPEC CPU2006 applications [32]) are split by network-intensity class, increasing in network intensity from left to right. While BLESS and a buffered network have nearly identical performance at low load, the gap grows with workload intensity, exceeding a 40% difference in the highest-intensity workloads. Previous work [27] motivated bufferless networks for low-to-medium load where the gap is small. Clearly, the bufferless network is unsuitable for the network-intensive workloads when performance is a concern.

However, the advantage of the bufferless-based approach is seen by examining *power*, as the bottom panel in Figure 1 shows. In the best case (at low load), bufferless networks consume less than a quarter of the power of buffered networks. Even at very high load, power consumption remains less than a buffered network. As previous work has noted [12, 27], removing the in-network buffers from NoC routers yields this significant power savings, as well as reduced die area. If the performance gap can be addressed, these cost-savings become more important: mechanisms that enhance the performance of bufferless networks at high load without reducing their power and area efficiency could lead to compelling energy-efficient interconnect designs for both network-light and network-intensive workloads.

In addition to system performance, network congestion and saturation in bufferless NoCs can lead to degradation of other important system properties. In this paper, we consider *fairness* as a first-class system metric alongside performance. Figure 1 also shows fairness, measured as maximum application slowdown [8, 22], across both buffered and bufferless baselines (lower is better). The bufferless network degrades fairness (increases max. slowdown) under high load: the reduced capacity leads to significant slowdowns because some applications starve (are denied adequate service) when the network is saturated by network-intensive applications. Many prior works [2, 10, 13, 25, 28] optimize for fairness as well as performance: this means to improve the worst case for a single application's performance degradation, which yields a more robust mechanism and can also improve system throughput.

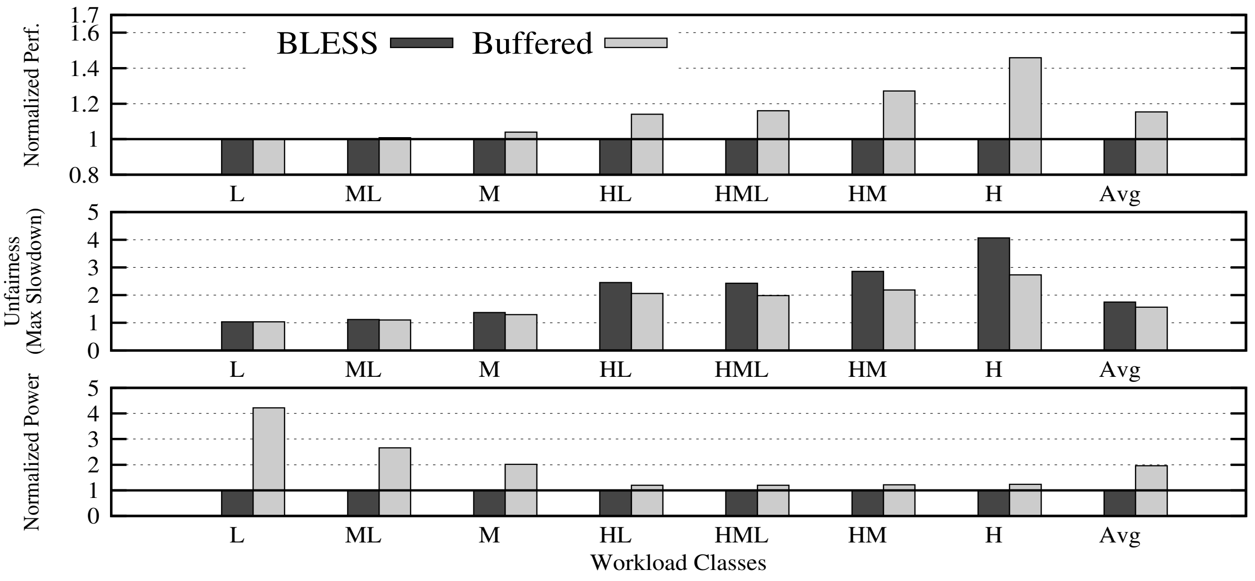


Figure 1: Buffered and bufferless network designs in a 4x4 (16-node) CMP. Details in §5.

Fundamentally, bufferless networks suffer the performance and fairness gaps because they become saturated at lower load. Higher network utilization leads to more deflections, which in turn reduces network throughput and system performance. However, past work [29] has shown that by limiting the network load with throttling (limiting request injection rate), deflections reduce, and the bufferless network attains higher throughput. Our goal in this paper is to develop a simple, easy-to-implement throttling-based mechanism for bufferless NoCs that further improves both performance and fairness in bufferless networks.

4 ACT: Adaptive Cluster Throttling

4.1 Throttling Strategy and Application Types

At a high level, ACT works by observing application network intensity, grouping applications into clusters based on that intensity, and then applying throttling to each cluster in a certain way.

What is Throttling? In broad terms, *throttling* in ACT is the blocking of new traffic injection into the network from a particular application in order to reduce load. We use a throttling technique that is *soft*: throttled nodes can still sometimes inject. We also assume that each node has separate injection queues for request packets and data response packets to other nodes, and that only a node's own request packets are throttled. For a throttling rate r, every time a node attempts to inject a request packet, it randomly chooses to block the injection with probability r. Throttling rates range from 0% (unthrottled) to 100% (fully blocked).

Overall Goal of Throttling: ACT applies throttling in order to reduce network load when the network is congested. This reduces interference and thus improves system performance and fairness. In order to be most effective at reducing load, ACT observes application network intensity and throttles those applications that will reduce network load the most.

Impact of Throttling on System Performance: Throttling a particular application both causes slowdown to that application (by reducing its access to the network), and causes performance gain in other applications (by reducing overall network load). The performance impact on the throttled application depends on application characteristics such as latency tolerance, which are difficult to measure directly. However, in a bufferless network, an application's impact on other applications correlates closely to its injection rate (network intensity): injection rate contributes to network utilization, and high utilization in a bufferless network causes a high deflection rate, which distributes load throughout the network and degrades performance for all applications [29]. Throttling an application with a high injection rate will reduce network load more than throttling an application with a low injection rate. Thus, ACT throttles applications according to network intensity, using the following principles.

Never Throttle Low-Intensity Applications: Low-intensity applications, or applications that have a low injection rate, contribute little to overall network load. Thus, throttling such applications will not usually benefit system performance, because throttling a low-intensity application does not reduce network load significantly. Hence, ACT *never throttles* low-intensity applications.
Sometimes Throttle Medium-Intensity Applications: Medium-intensity applications have higher injection rates than low-intensity applications, and thus have a non-negligible impact on total network load. Throttling medium-intensity applications reduces network load and improves the performance of the other applications. However, throttling such an application also impacts its own performance, because it is dependent on the network to make forward progress some of the time. Thus, throttling all medium-intensity applications inhibits any of them from taking advantage of the reduced congestion in the network. Hence, ACT builds clusters of medium-intensity applications that are *sometimes throttled*, and then throttles most of these clusters while unthrottling one at a time in fine-grained timeslices. This balances the impact of throttling across all medium-intensity applications, because each obtains unthrottled network access for some time period.
Always Throttle High-Intensity Applications: Finally, high-intensity applications have very high network injection rates. These applications are nearly always network-bound, stalling on outstanding requests; thus, they use as much network bandwidth as available. As long as there are lower-intensity applications in the network, ACT *always throttles* high-intensity applications, because unthrottled access allows them to saturate the network and degrade the performance of all other applications.
Throttling Strategy: In summary, ACT forms three types of clusters: one *never-throttled* cluster, multiple *sometimes-throttled* clusters, and one *always-throttled* cluster. Applications are placed in clusters based on measured intensity. Low-intensity applications are placed into the never-throttled cluster because they have little impact on network load. High-intensity applications significantly degrade all other applications' performance, and so are placed in the *always-throttled* cluster. Finally, medium-intensity applications are sensitive to throttling but also place load on the network, and so are split into many *sometimes-throttled* clusters. Most of these clusters are throttled, while one at a time is unthrottled. Figure 2 depicts this scheme.

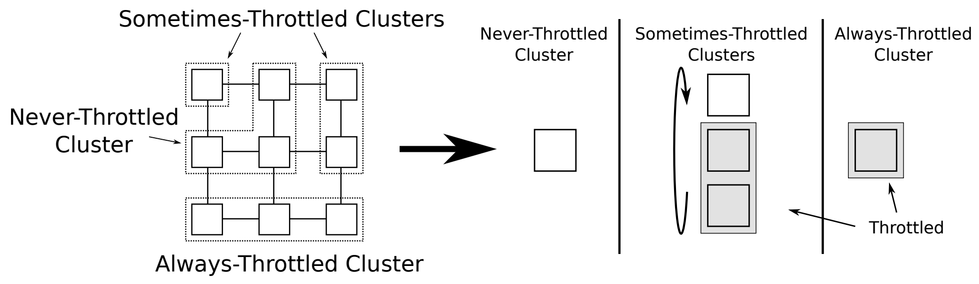


Figure 2: Cluster-based throttling groups applications into clusters and shares network capacity via timesliced throttling. Three types of clusters are formed: a never-throttled cluster, several sometimes-throttled clusters, which are unthrottled one at a time, and an always-throttled cluster.

4.2 Cluster Formation and Throttling

Measuring Application Intensity: ACT needs to quantitatively measure applications' network intensity in order to throttle each application appropriately. Because the network services cache misses, ACT uses cache miss rate, measured as L1 MPKI (misses per kilo-instruction), as a network intensity metric. This metric is stable (independent of throttling decisions) and easy to measure at each core. Other work [8] also uses L1 MPKI to measure application intensity.
Forming Clusters: Algorithm 1 shows ACT's per-epoch clustering procedure. The algorithm is *epoch-based*: it collects information about application network intensity (L1 MPKI) over a fixed-length epoch, and at the end of the epoch, decides how to throttle for the next epoch. The algorithm first sorts all applications by observed MPKI (misses per kilo-instruction) in the last epoch. It then iterates over all applications in this order (low to high MPKI) and builds clusters one at a time.

First, the never-throttled cluster is filled until the sum of MPKI values for all applications in the cluster exceeds the *never-throttled cluster capacity NeverCap*, a fixed algorithm parameter (we describe its effects in §6.4). Thus, the lowest-intensity applications are granted unthrottled access. By filling to a capacity, rather than using a per-application threshold, we handle the case where all applications are high-intensity: in this case, unthrottling the lowest-intensity of these applications benefits performance, because there are no lower-intensity applications that would benefit from an unsaturated network.

Once the never-throttled cluster is filled, sometimes-throttled clusters are created. These clusters are created one at a time, and applications are added to the most recent cluster until a new cluster is started. The algorithm fills each cluster to the *sometimes-throttled cluster capacity $SometimesCap$*, another algorithm parameter. $SometimesCap$ specifies the maximum sum of MPKI values for applications in each sometimes-throttled cluster. Together with the never-throttled cluster capacity, this constraint limits the maximum network load due to unthrottled applications, because at any given time, exactly one sometimes-throttled cluster is unthrottled. Finally, once applications' MPKI values become too large for a single sometimes-throttled cluster, the applications are added to the always-throttled cluster.

Algorithm 1 ACT: Clustering Algorithm

```
at the beginning of each epoch:
empty all clusters
sort $N$ applications by MPKI measurements $MPKI_i$
for sorted application $i$ in $N$ do
    if total MPKI of never-throttled cluster $+MPKI_i \leq NeverCap$ then
        Add application $i$ to the never-throttled cluster
    else if MPKI of the latest sometimes-throttled cluster $+MPKI_i \leq SometimesCap$ then
        Add application $i$ to the latest sometimes-throttled cluster
    else if $MPKI_i \leq SometimesCap$ then
        Create a new sometimes-throttled cluster
        Add application $i$ to the new sometimes-throttled cluster
    else
        Add application $i$ to the always-throttled cluster
    end if
end for
```

Throttling Clusters: The always-throttled cluster remains throttled continuously. The sometimes-throttled clusters are unthrottled one at a time, in round-robin order, while all other sometimes-throttled clusters are throttled. The throttling timeslice is a fixed interval that evenly divides into the epoch. Because the number of timeslices in an epoch may not be a multiple of the number of sometimes-throttled clusters, the choice of first unthrottled cluster in an epoch is randomized to avoid bias.

Controlling Throttling Rate: Every throttled cluster is throttled at a global rate. This rate is controlled dynamically as network load rises and falls. When the system is lightly loaded, no throttling is necessary. As load increases, the throttling mechanism becomes as aggressive as necessary to increase performance. This is accomplished with a *feedback-based throttling rate adjustment* based on network load.

To measure network load, we use *network utilization*, which is the average number of links occupied by traffic in the network at a given time over the total number of links. It ranges from 0% to 100%. Beyond a certain utilization, deflections cause performance degradation, and using throttling to reduce utilization increases system performance, as shown in prior work [29]. A certain range of network utilizations, fixed for a given network size and topology, will provide good performance for many workloads. This is because traffic in real network-intensive workloads tends to have a relatively homogeneous distribution, due to a large number of deflections. Because of this, we adjust throttling rates according to a *fixed network utilization target*. Sensitivity to the network utilization target is evaluated in §6.5.

Algorithm 2 shows the adjustment mechanism. At each epoch, this algorithm compares actual network utilization with a target utilization set statically for the network design, and adjusts the throttling rate as necessary. Per-epoch rate steps are given in Table 1: the rate moves with larger steps when it is lower, and smaller steps when it is higher, because most applications are more sensitive to throttling at higher throttling rates. The throttling rate has a maximum at 95%, based on empirical sensitivity sweeps that show a large performance drop beyond that point. Note that when the baseline network utilization is below the target, ACT will converge to a rate of 0%, effectively deactivating the mechanism.

Algorithm 2 ACT: Throttling-Rate Adjustment

```
initially: throttle rate $\leftarrow 0$
at the beginning of each epoch:
if $netUtil \geq targetNetUtil$ then
    if throttle rate $< max\_throttle\_rate$ then
        increase throttle rate for all clusters by throttling rate delta
    end if
else if $netUtil \leq targetNetUtil$ then
    if throttle rate $> 0$ then
        decrease throttle rate for all clusters by throttling rate delta
    end if
end if
```

Table 1: Throttle-Rate Adjustment

Current Throttling Rate	Throttling Rate Delta
0% – 70%	10%
70% – 90%	2%
90% – 100%	1%

Parameter	**Setting**
System topology	4x4 and 8x8 mesh; core and shared cache slice at every node
Core model	Out-of-order, 128-entry instruction window, 16 MSHRs
Private L1 cache	64 KB, 4-way associative, 32-byte block size
Shared L2 cache	perfect (always hits) shared, cache-block-interleaved mapping
Cache coherence	directory-based with cache-to-cache transfers, perfect striped directory
Interconnect Links	1-cycle latency, 128-bit width (1-flit request packets, 4 flit data packets)
Bufferless router	2-cycle latency, FLIT-BLESS [27] or CHIPPER [12] (§6.2)
Buffered router	2-cycle latency, 8 VCs, 8 flits per VC, oldest-first arbitration
Simulation parameters	5M cycle warmup, 25M cycle active execution
Multiprogrammed Workloads	60 in 4x4, 60 in 8x8, drawn from SPEC CPU2006 [32]
Multithreaded Workloads (§6.2)	4 from SPLASH-2 [36], 16 threads in a 4x4-CMP

Table 2: Simulation parameters.

5 Experimental Methodology

Simulator Model: We use an in-house cycle-accurate instruction-trace-driven simulator for our evaluation. We model a 4x4- or 8x8-mesh CMP. Table 2 shows the detailed configuration of our system. Note that we model *perfect shared caches*, as also evaluated for CHIPPER [12], BLESS [27] and previous work on throttling [29]. This configuration forces all shared-cache accesses to be hits, removing memory latency from consideration. Thus, the NoC becomes the system bottleneck. This configuration allows us to study high-load behavior, and is a realistic situation for cache-resident workloads.

Because only the L1 caches require warming with a perfect shared cache, 5M cycles of warmup is found to be sufficient. 25M cycles of execution gives stable results for the application phases simulated.

Workload Formation: We focus on network-intensive workloads for our main evaluations since the problem we are solving occurs only when the network is under high load. In a sensitivity study in §6.2, we will separately show behavior in network non-intensive workloads. In order to form suitable workloads for the main evaluations, we split the SPEC CPU2006 [32] benchmark set into three categories: High, Medium and Low. We perform cache-miss profiling over the instruction traces captured from representative sections of each benchmark, and split based on MPKI thresholds. In particular, High-intensity benchmarks have MPKI greater than 50, Medium-intensity benchmarks fall between 5 and 50 MPKI, and Low-intensity benchmarks form the remainder, as shown in Table 3.

Based on these categories, we form workload classes with predefined ratios of High, Medium and Low-intensity applications. We take four classes that contain high-intensity applications: H (all applications High), HM (High/Medium), HML (High/Medium/Low), and HL (High/Low). For each workload, we pick the intensity of each node's application randomly from the given categories, and then pick an application for each node randomly from the appropriate list. We evaluate 15 workloads per class.

We also evaluate multithreaded workloads in §6.2 using SPLASH-2 [36] applications. For these workloads, a 4x4 network runs 16 threads of a single application. Simulations are run for a fixed number of barriers (i.e., main loop iterations) and total execution times are compared.

Algorithm Parameters: Table 4 gives empirical values for our algorithm's parameters. We evaluate the effects of parameters and their sensitivity in §6.5. Cluster capacity parameters (*NeverCap* and *SometimesCap*) are treated specially in §6.4, as they control a performance-fairness tradeoff. Note that we define two variants of ACT, ACT-

Low-Intensity Apps	MPKI	Medium-Intensity Apps	MPKI	High-Intensity Apps	MPKI
perlbench	0.1	gromacs	7.4	libquantum	53.1
calculix	0.3	hmmer	8.1	GemsFDTD	55.0
sjeng	1.2	astar	10.9	lbm	57.1
namd	1.2	cactusADM	12.8	soplex	57.6
gcc	1.4	bzip2	19.6	mcf	122.4
wrf	1.4	omnetpp	22.4		
tonto	1.5	xalancbmk	27.2		
dealII	1.5	sphinx3	27.7		
gobmk	2.6	milc	27.9		
povray	2.9	leslie3D	42.4		
h264ref	4.2				

Table 3: L1 Cache Misses Per Kilo-Instruction (MPKI) of 26 SPEC 2006 benchmarks.

Perf and ACT-Fair, with different cluster capacity parameters. These variants are tuned for performance and fairness, respectively.

Algorithm Parameter	**Value**
Epoch Length	100K cycles
Timeslice Length	1000 cycles
Network Utilization Target	60% (4x4); 55% (8x8)
NeverCap never-throttled cluster capacity	150 (ACT-Perf), 50 (ACT-Fair)
SometimesCap sometimes-throttled cluster capacity	50 (ACT-Perf), 150 (ACT-Fair)

Table 4: Cluster-Throttling Algorithm Parameters

Metrics: To measure system performance, we use the well-known *Weighted Speedup* metric [31, 11] (Eqn. 1), which is shown to be equivalent to *System Throughput* [11], for our evaluation. All IPC_i^{alone} values are measured on the baseline bufferless network. To ensure no bias in selecting metrics to report our system performance, we also present our results in two more metrics: *System IPC* [11](Eqn. 2) and *Harmonic Speedup* [25, 11](Eqn. 3), which is inversely proportional to *Average Normalized Turnaround Time* [11]. Note that we mainly present results in weighted speedup since these metrics show similar trends, which we will show in §6.

$$WeightedSpeedup = \sum_{i=1}^{N} \frac{IPC_i^{shared}}{IPC_i^{alone}} \tag{1}$$

$$SystemIPC = \sum_{i=1}^{N} IPC_i^{shared} \tag{2}$$

$$HarmonicSpeedup = \frac{N}{\sum_{i=1}^{N} \frac{IPC_i^{alone}}{IPC_i^{shared}}} \tag{3}$$

Additionally, we report *fairness* – intuitively, equal treatment or progress for all applications – for all workloads. Unlike several previous works [2, 13, 25, 28, 31], which define fairness as the ratio of maximum to minimum application slowdown, we measure fairness as the maximum application slowdown (Eqn. 4) [8, 22, 23, 34], because it favors mechanisms that improve the slowest applications' performance explicitly (rather than equalize slowdowns at the cost of performance). For this metric, lower is better (more fair), hence it measures *unfairness*. Because of this inversion, we report the harmonic mean rather than arithmetic mean of fairness for workload classes and overall conclusions.

$$Unfairness = \max_i \frac{IPC_i^{alone}}{IPC_i^{shared}} \tag{4}$$

Performance and Fairness Gaps: We discuss the *performance gap* and *fairness gap* quantitatively in our evaluations. The performance gap is defined in terms of weighted speedup, and exists between the bufferless (BLESS) baseline and buffered baseline. For a mechanism to close 50% of the gap, it must achieve 50% of the improvement over bufferless that the buffered baseline achieves. The fairness gap is defined in terms of *minimum speedup*, or the inverse of maximum slowdown, because maximum slowdown is an inverse (lower-is-better) metric, but is otherwise defined analogously.

Comparisons to Other Mechanisms: We compare our mechanism to two other throttling techniques. The first is a homogeneous throttling mechanism. This uses network utilization as a measure of congestion, and implements the throttling rate adjustment in Algorithm 2. In contrast to ACT, *all* nodes are throttled at the same rate, without any clusters. The second comparison is to an implementation of prior work on throttling in bufferless networks in Nychis et al. [29], which we call "heterogeneous throttling."

Power Model: To model network power, we use a similar model for BLESS and buffered networks to that used in [12]. This model takes detailed event counts from cycle-accurate network simulations. The model is based on synthesized Verilog models for control logic and ORION [35] estimates for the router datapath, buffers (when applicable) and network links, assuming 65nm technology.

6 Experimental Evaluation

System Performance: The top panel of Figure 3 shows the normalized system performance (measured as weighted speedup) of bufferless and buffered baselines, AFC, simple throttling (homogeneous and heterogeneous), and two variants of our mechanism, ACT-Perf and ACT-Fair, optimized for performance and fairness respectively (§6.4). Results are split by workload intensity classes, as defined in §5. We focus on 4 network-intensive workload classes, each with 15 workloads.

First, the buffered network outperforms the bufferless baseline significantly. This is the *performance gap* motivated in §3. In a 4x4 CMP, the buffered network has 25.7% higher performance than the bufferless baseline on average in these network-intensive workloads. Adaptive Flow Control [18], which dynamically switches each router between

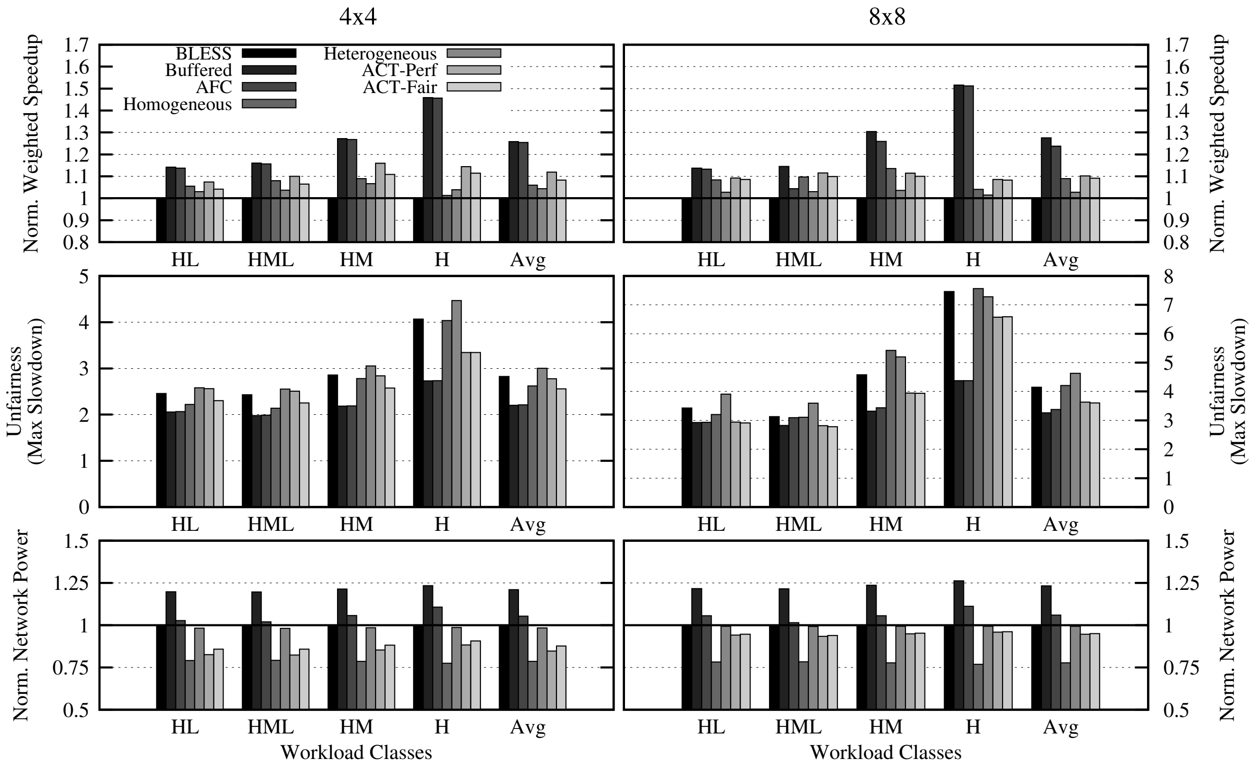


Figure 3: Performance, fairness, and power for 60 network-intensive workloads in 4x4 and 8x8 CMPs.

bufferless and buffered modes, provides nearly the same performance as a buffered network because it is able to enable its buffers when needed.

Both homogeneous and heterogeneous throttling improve performance over the bufferless baseline, at 5.9% and 4.3% respectively in a 4x4 network. Both variants of ACT, ACT-Perf and ACT-Fair (§6.4), outperform homogeneous and heterogeneous throttling, providing the best performance of the throttling techniques. ACT-Perf, optimized for performance, attains 11.9% performance improvement in 4x4 networks, and 10.2% in 8x8, closing 46.4% and 40.0% of the performance gaps, respectively.

Homogeneous throttling achieves performance gains simply by reducing network load, using ACT's throttling rate adjustment algorithm. However, homogeneous throttling has no clusters, and so cannot throttle applications differently according to their intensities, as ACT does (§4.1). ACT achieves an additional 6% performance gain over the baseline bufferless network due to this distinction. In addition, in the all-High class, homogeneous throttling has little performance benefit, because it applies the maximum throttling rate in an unsuccessful attempt to bring network utilization down to the target. In contrast, ACT's never-throttled cluster is able to capture several of the applications in the workload, because it fills to a capacity and no lower-intensity applications are present. ACT is able to extract a sizeable performance gain with this configuration.

ACT outperforms heterogeneous throttling in all cases for two reasons: it dynamically adjusts throttling rate to match the workload intensity, and it is able to make finer-grained and more accurate throttling distinctions. ACT uses a single global throttling rate controlled by a feedback loop that observes current network utilization. The mechanism controls the impact of throttling on each application by the way that it builds clusters and timeslices them. In contrast, heterogeneous throttling uses an open-loop throttling rate adjustment that applies a distinct throttling rate to each application based on its network intensity. Thus, heterogeneous throttling can sometimes over- or under-throttle the workload. In addition, its distribution of network access between applications is less precise than ACT's timeslicing because individual application performance can be very sensitive to throttling rates.

Figure 4 presents the system performance in weighted speedup, system IPC, and harmonic speedup without normalization to BLESS in both 4x4 and 8x8 networks. We report three separate performance metrics to show that our results are robust against choice of metric. ACT improves system IPC by 13.3% (10.7%) , and harmonic speedup by 9.2% (11.4%) relative to BLESS in a 4x4 (8x8) network. In comparison, a buffered network improves these metrics by 20.8% (19.1%) and 25.1% (27.3%) respectively. As shown, ACT closes 63.9% (56.0%) of the system IPC gap and 36.7% (41.7%) of the harmonic speedup gap. In general, the trends are shown to be similar for all the metrics, so our conclusions drawn from weighted speedup do not depend on any particular performance metric.

Measured using the harmonic speedup, which captures both system throughput and fairness [25, 11], ACT-Perf exhibits less performance improvement over homogeneous throttling than when using weighted speedup: 2.7% in a 4x4 network and 6.0% in a 8x8 network. This smaller improvement is because harmonic speedup incorporates fairness as well as throughput; as the middle panel of Figure 3 shows, ACT-Perf actually results in higher average system unfairness than homogeneous throttling in a 4x4 network. However, despite the impact that increased unfairness has on harmonic speedup, ACT-Perf still provides an absolute improvement over all other evaluated throttling mechanisms.
Fairness: Figure 3 (middle panel) shows fairness (measured as maximum slowdown) for all evaluated mechanisms. In these plots, a higher number indicates a higher maximum slowdown and thus worse fairness. The first conclusion is that a bufferless network has worse fairness on average, due to lower network bandwidth capacity and higher congestion, which starves applications of service and causes large slowdowns. The fairness gap is 24.8% (27.2%) on average in a 4x4 (8x8) network. Second, homogeneous throttling closes only a portion of this gap on average, because it treats all applications to be the same, whereas some applications are more sensitive to throttling than others. It can improve fairness more in the HL and HML classes because it closes a greater portion of the performance gap, and thus increases all applications' performance. In the HM and H classes, it is almost as unfair as the baseline bufferless case.

The heterogeneous throttling mechanism actually worsens fairness on average and in all workload classes in a 4x4 network, and on average and in all but one class in an 8x8 network. This is because the mechanism explicitly optimizes for system performance: it throttles applications that have lower *instructions-per-flit* (instructions retired per unit of network traffic) measurements, or those that lead to less system performance gain for a given amount of network throughput [29]. This can significantly impact individual applications' performance, because applications with lower instructions-per-flit measurements can be starved of service.

On average, ACT-Fair achieves better fairness than homogeneous and heterogeneous throttling, and shows 14.5% fairness improvement in 4x4 and 15.1% in 8x8 networks from the bufferless baseline. ACT-Fair closes 43.4% of the fairness gap with buffered networks in 4x4 networks and 55.6% in 8x8. ACT attains better fairness by treating each application according to its network intensity (§4.1). By throttling network-intensive applications at all times,

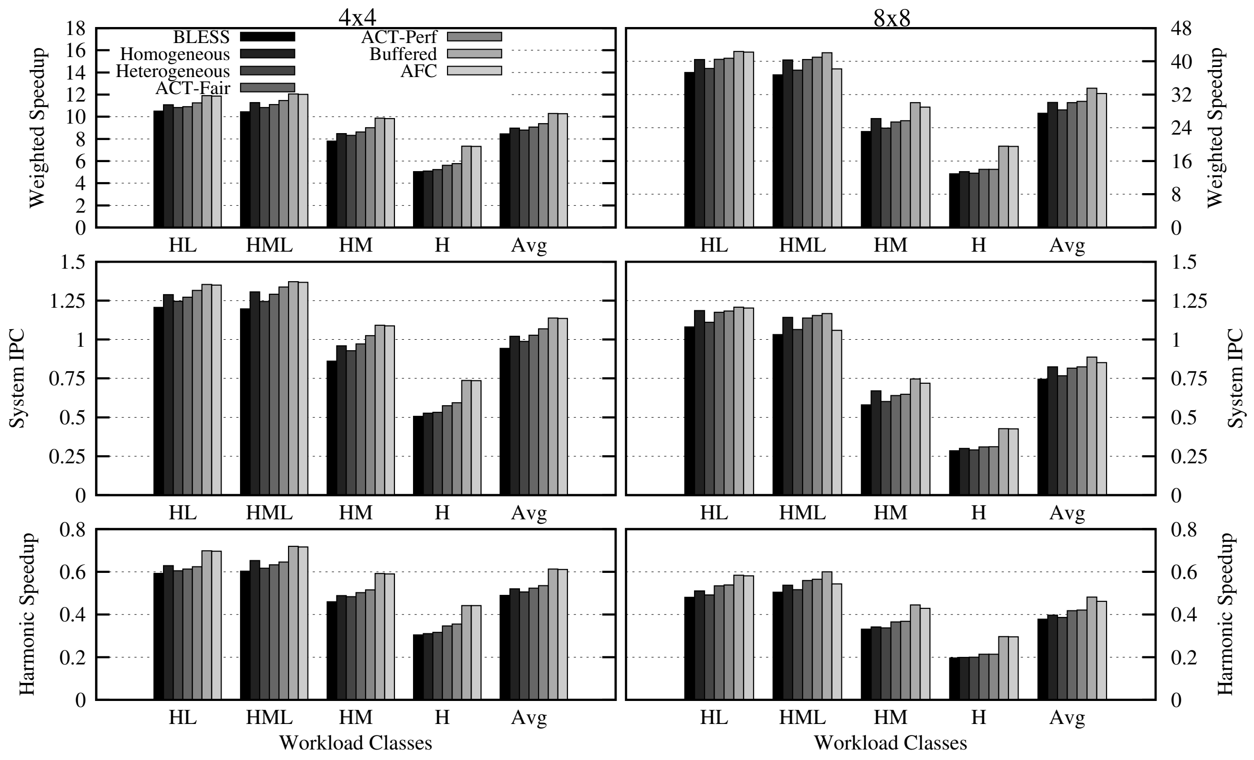


Figure 4: Performance results in Weighted Speedup, System IPC, and Harmonic Speedup for 60 network-intensive workloads in 4x4 and 8x8 CMPs.

ACT reduces network load and allows better network access for the other applications. Timesliced throttling of sometimes-throttled clusters allows many medium-intensity applications to have a chance at unthrottled injection, while still reducing network load overall. ACT's fairness improvement over homogeneous throttling demonstrates that this application-awareness yields significant gains.

Finally, a clear improvement in fairness is visible from ACT-Perf to ACT-Fair. In 4x4 networks, ACT-Perf has similar or slightly lower fairness to simple homogeneous throttling on average, whereas ACT-Fair achieves better fairness than any other throttling technique, at the cost of some system performance. This tradeoff occurs due to clustering thresholds, and we evaluate the effect explicitly in §6.4. In 8x8 networks, the reduction of general network congestion is a more dominant effect than the differences in cluster formation between ACT-Perf and ACT-Fair, and so both ACT variants improve fairness over other throttling techniques. Together, ACT-Perf and ACT-Fair show two extremes of a tradeoff spectrum that ACT allows between fairness and performance.

Network Power: Figure 3 (bottom panel) shows normalized network power. ACT reduces network power due to decreased network utilization: power reduces by 15.4% on average in 4x4 and 5.4% on average in 8x8 networks. Combined with the performance improvements, this yields reduced network and system energy overall. Homogeneous throttling achieves a slightly larger power reduction because it is more aggressive at reducing network utilization across all nodes. In exchange for this reduction, however, it achieves less performance and fairness. ACT thus presents a better tradeoff, especially when full-system power (in addition to network power) is considered: the reduced execution time due to improved system performance, coupled with reduced network and system power, can lead to significant energy reductions.

6.1 Effect on Other Bufferless Interconnects

CHIPPER: We show the effect of ACT on another low-cost bufferless interconnect, CHIPPER [12], which makes tradeoffs that sacrifice some performance in order to allow for a simple router design. ACT is also directly applicable to CHIPPER, because CHIPPER is a simplified deflection-based bufferless routing design.

Figure 5 shows performance and fairness for a CHIPPER baseline, and ACT-Perf on CHIPPER, in a 4x4 network.

Unmodified CHIPPER performs 1% worse on average across all high-intensity workloads than BLESS. ACT on CHIPPER attains 6.8% performance improvement over the CHIPPER baseline, or 5.8% over BLESS. Note that ACT has not been optimized for this configuration. Nonetheless, we conclude that ACT is also effective on other bufferless network designs.

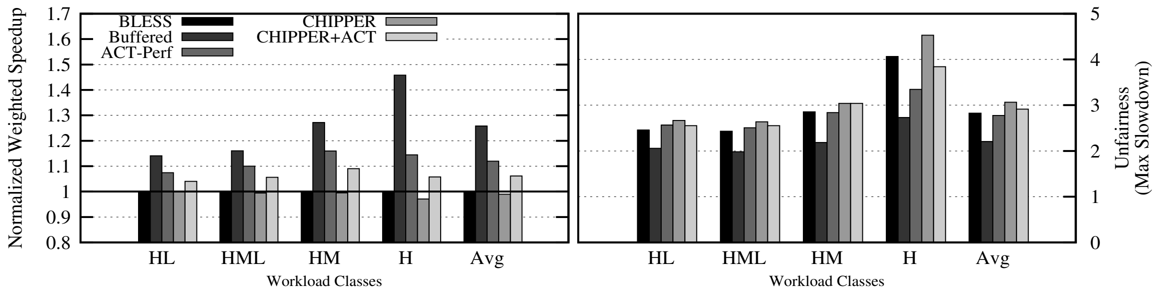


Figure 5: ACT-Perf on CHIPPER [12], a low-cost bufferless network, for a 4x4 network. Parameters are identical to ACT-Perf in our main evaluation, except for a network utilization target of 45%.

6.2 Behavior with Low-Intensity and Multithreaded Workloads

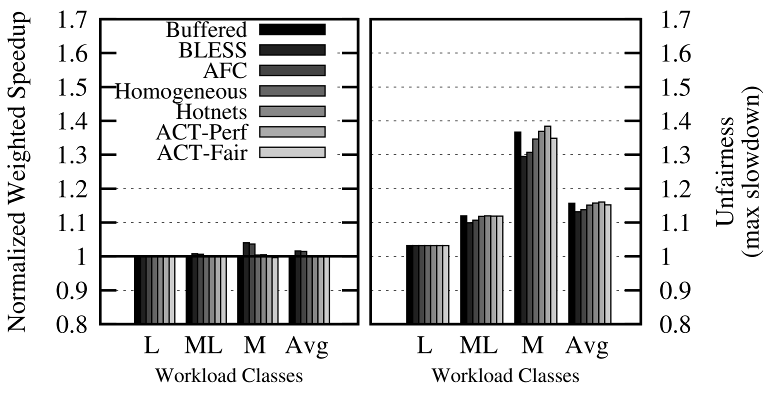
Low-intensity multiprogrammed workloads: We evaluate the same set of low-intensity workloads shown earlier in Figure 1 in §3 in order to show that ACT has no negative effects where little performance gap exists. Figure 6a shows performance and fairness for these workload categories in a 4x4 network. In all-Low and Medium-Low workloads, all systems perform essentially the same. In Medium-intensity workloads, the performance gap begins to emerge, showing a few percent in performance drop from buffered to bufferless networks. A small fairness gap also starts to emerge. ACT-Fair is able to recapture some of the fairness lost in bufferless networks even when the network has not yet reached high load.

Scientific multithreaded workloads: Although ACT as described in this work is not explicitly designed for multithreaded workloads, it can still provide benefits for such workloads by reducing network contention. We evaluate standard ACT on four 16-threaded workloads from SPLASH-2 [36] in a 4x4 network. Normalized speedups are shown in Figure 6b. In `luc` and `lun`, ACT closes most or all of the performance gap. The largest performance gap exists in `fft`, at 11.7%. However, ACT does not close this gap because it is not aware of cooperating threads. ACT does not close this gap because it is not aware of cooperating threads. In other words, because threads of a multithreaded program have dependencies between them, and slowing down one thread (by throttling it) might also impact the execution speed of other threads and the program as a whole, ACT can no longer throttle network nodes independently, as we have described so far. ACT could be easily extended to handle this case by treating all threads in a process as a single unit with one total MPKI, and assigning the threads atomically to one cluster. We do not evaluate such an extension but leave it for future work.

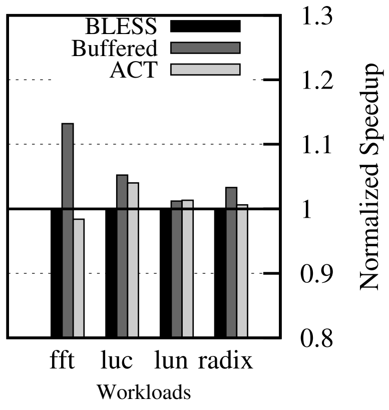
6.3 Implementation Cost

Our mechanism consists of three parts: measurement of MPKI and network utilization cluster and throttling-rate computation each epoch, and throttling and timeslicing at the routers.

First, MPKI and network utilization can be measured easily in hardware with simple integer counter structures reset every epoch. Second, throttling rates and timeslicing must be enforced at each router. Throttling can be implemented as a fixed duty cycle in which injection is allowed, or using a pseudorandom number generator. Timeslicing requires a timer that can be programmed with a repeating schedule to activate throttling. Note that no global coordination is required, since the NoC has a global clock domain. Finally, the cluster formation computation must be done in some central place. The algorithm consists of (i) a sort by MPKI and (ii) one pass over the application list. Although a detailed model is outside the scope of this work, such a computation should require at most several thousand cycles out of a 100K-cycle epoch, running on one central CPU node. This computation and communication overhead that occurs every epoch, is small given a 100K-cycle epoch. Assuming the computation takes 1000 cycles, then the overhead is minimal at 1% on one single core.



(a) Low-intensity workloads in a 4x4 network



(b) Multithreaded workloads

Figure 6: Several evaluations of ACT in workloads with low network intensity.

6.4 Cluster Sizes, Fairness and Performance

Two algorithm parameters, *NeverCap* (never-throttled cluster capacity) and *SometimesCap* (sometimes-throttled cluster capacity), control cluster formation and have important effects on the fairness and performance of ACT. Recall from §4.2 that the never-throttled cluster capacity defines the maximum sum of MPKI measurements for applications placed in the never-throttled cluster. Likewise, the sometimes-throttled cluster capacity defines the maximum MPKI sum for applications in any particular sometimes-throttled cluster, and applications with MPKI larger than this capacity are placed in the always-throttled cluster. Each parameter thus defines the total network intensity of a particular type of cluster.

Together, *NeverCap* and *SometimesCap* define the total network intensity of the applications that will be unthrottled at any given time (i.e., the never-throttled and one particular sometimes-throttled cluster): we call this the "unthrottled capacity." Thus, this sum also controls what portion of the nodes *are* throttled. If too few nodes are throttled, then ACT will be unable to bring the network utilization of a high-intensity workload down to the target, even if the throttling rate takes its maximum value. Such a situation will result in very unfair treatment of the few applications that are throttled (because of the high throttling rate) and small performance gains overall (because network utilization is lowered only slightly). Conversely, if too many nodes are throttled, ACT converges to simple homogeneous throttling, and loses the performance and fairness advantages shown in §6 over this scheme.

To examine performance sensitivity to the sum $NeverCap + SometimesCap$, we hold $SometimesCap = 0$ and sweep *NeverCap*. This removes sometimes-throttled clusters completely, leaving only the never-throttled and always-throttled clusters. The lowest-intensity applications fill the never-throttled cluster until the cluster's total MPKI reaches *NeverCap*, and the remaining applications are then placed in the always-throttled cluster. In this controlled sweep, $NeverCap = 0$ corresponds to homogeneous throttling, because all applications are placed in the always-throttled cluster; and as *NeverCap* exceeds the total MPKI of all applications in the network, the system converges to the baseline network, because all applications are placed in the never-throttled cluster. System performance for this sweep is shown in Figure 7. In 4x4 networks, the best performance occurs at a low-intensity cluster capacity of 150 MPKI. Note that our primary evaluations use a capacity of 200 rather than 150 MPKI in 4x4 networks, which performs nearly as well in this sweep. We conclude that the total unthrottled capacity has a significant impact on ACT's effectiveness.

Next, the distribution of the unthrottled capacity between the *NeverCap* and *SometimesCap* capacities trades off fairness and performance. Figure 8 shows five different configurations of ACT in 4x4 networks where $NeverCap + SometimesCap$ is held constant but the distribution is swept (points are notated $(NeverCap, SometimesCap$, ACT-Fair corresponds to $(50, 150)$, and ACT-Perf corresponds to $(150, 50)$). When more capacity is given to the never-throttled cluster, it captures a larger portion of the lowest-intensity applications in the workload (because it fills up to its capacity, and may contain several medium-intensity as well as low-intensity applications when it is large enough). Granting this capacity improves performance up to a point, because lower-intensity applications can make more forward progress with less load on the network. When this capacity is increased past a certain point, however, the sometimes-throttled clusters receive a much smaller share of the unthrottled capacity, and the performance of many medium-intensity

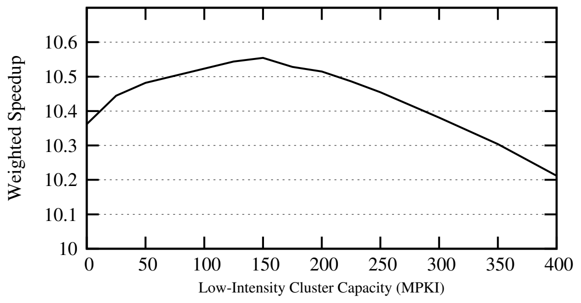


Figure 7: Performance as a function of unthrottled capacity.

applications is degraded because they are placed into the always-throttled cluster. When *SometimesCap* $= 0$, the sometimes-throttled clusters disappear completely.

On the other hand, when more capacity is given to sometimes-throttled clusters and less to the never-throttled cluster, the unthrottled capacity is distributed to more medium-intensity applications via timeslicing. Because many applications are unthrottled for short periods while the overall network load is reduced, the system is more fair because all of these applications can make forward progress. At the same time, performance reduces because less capacity is given to the never-throttled cluster, and applications that were previously in this cluster must wait for a turn for unthrottled access in the sometimes-throttled clusters. Beyond a certain point, this effect also reduces fairness, because too few low-intensity applications (which can make fast forward progress) obtain unthrottled network access.

Overall, we conclude that ACT shows a range of configurations that trade off fairness and performance based on the distribution of unthrottled cluster capacity.

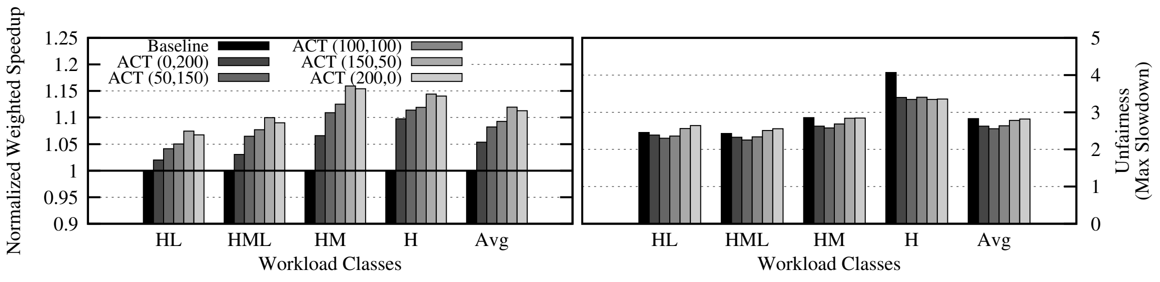


Figure 8: Fairness-performance tradeoff with never-throttled and sometimes-throttled cluster capacities.

6.5 Sensitivity to Algorithm Parameters

Epoch and timeslice length: Epoch length has little effect on performance and fairness improvements (less than 1% delta in performance from 20K to 1M cycles), due to stability of application behavior over that time period. Timeslice length has little effect once it is long enough; beyond 1K cycles, performance varies by less than 1%. Thus, we use 100K cycles and 1K cycles for epoch and timeslice length, respectively, allowing for up to 100 clusters in each epoch. The choice of epoch length trades off between responsiveness and overhead. A shorter epoch captures more fine-grained changes in application behavior, but with higher overhead and less stable measurements. A longer epoch gives more stable samples and amortizes recomputation over more time, but does not capture fine-grained changes in behavior.

Target network utilization: This parameter controls the throttling aggressiveness of ACT and thus its effectiveness. Figure 9, sweeping from 0% to 100% in ACT-Perf, shows that weighted speedup peaks at between 50% and 65% network utilization for a 4x4 network, with less than 1% variation at points within this range. In addition, Figure 9 shows that fairness is best in this range. Beyond 65%, performance drops significantly because throttling is not aggressive enough and the network remains congested. When the target is too low, throttling is too aggressive, and is capped at the maximum throttling rate (thus, performance remains stable and does not degrade if the target is too aggressive).

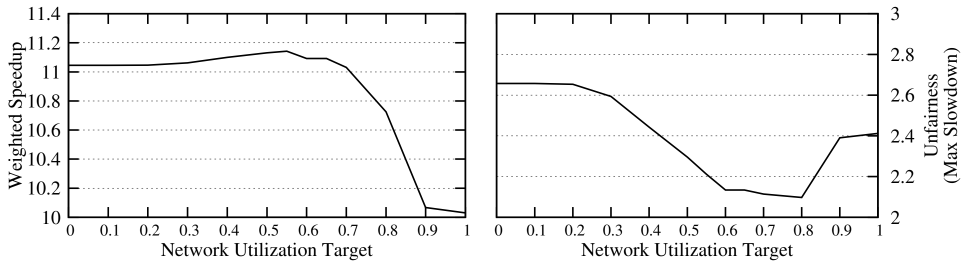


Figure 9: Performance and fairness sensitivity to network utilization target in a 4x4 network over 20 sweep workloads.

7 Related Work

To our knowledge, ACT is the first work to apply source throttling to a bufferless NoC with explicit goals of both performance and fairness. Other work studies throttling in the NoC and elsewhere, tackles NoC congestion, and targets the bufferless-buffered performance gap using various alternate approaches.

Congestion Control in Buffered NoCs: The congestion problem, and more generally the problem of optimizing network-intensive workloads, has been explored in many previous works [8, 9, 15, 29, 33]. Various approaches use available buffers [21], available virtual channels [5], output queue size [30] or a combination of these [15] to detect congestion. When congestion is detected, options include congestion-aware routing [15], throttling [33], or simply boosting the NoC operating frequency.

Thottethodi et al. [33] proposed a mechanism that applies source throttling and dynamically adjusts aggressiveness, as in ACT. However, ACT differs in that it splits applications into distinct clusters and throttles clusters heterogeneously, whereas [33] throttles all nodes homogeneously.

While these mechanisms tackle congestion in buffered networks, fundamental differences between buffered and bufferless networks limit applicability of prior work. Because deflection spreads traffic throughout the network, network congestion is a *global* problem, and it tends to affect all nodes. Controlling global network load is most important. In a buffered NoC, localized congestion leads to a focus on detecting hotspots. Furthermore, bufferless networks suffer from *starvation*, the inability to inject traffic, at high load [29]. Thus a significant portion of a packet's time is spent in injection queues, and source throttling can control congestion more easily than in-network prioritization or adaptive routing.

Congestion Control in Bufferless NoCs: Nychis et al. [29] proposed a throttling-based approach to reduce congestion in bufferless NoCs. We compare to this work as "heterogeneous throttling" in our main evaluations. Whereas heterogeneous throttling partitions applications into throttled and not-throttled categories every epoch, and then applies differing throttling rates to each throttled application, ACT uses timesliced sometimes-throttled clusters and applies the same throttling rate to all throttled applications. Doing so allows ACT to be much simpler, because it does not need to determine per-application throttling rates that are best for performance and fairness. ACT also adjusts throttling rate with a closed loop to adapt to different network loads. These two major differences lead to significant performance and fairness improvements, as demonstrated in §6.

Bridging the Performance Gap between Bufferless and Buffered Routing: Adaptive Flow Control [18] combines a bufferless and buffered router, and dynamically switches between the two modes depending on load. As shown in our evaluations, this technique closes the performance gap with buffered networks at high load. However, it loses the power/energy advantage of bufferless design at this point, because buffers are turned on. Additionally, it imposes an area penalty over both a buffered baseline (for additional control logic) and a bufferless baseline (for VC buffers). Note that ACT is orthogonal to AFC because ACT improves bufferless performance without switching to a buffered mode. AFC and ACT could be combined, by activating ACT in a congested bufferless network first, and switching to buffered mode with AFC only when the performance gain of ACT is insufficient. Delaying or eliminating a switch to buffered mode could improve the energy efficiency of a hybrid bufferless-buffered system.

Throttling-based Approaches in Other Parts of the System: Cheng et al. [3] proposed to reduce load in the memory system with a *Memory Task Limit*, which uses a timeslicing-based approach to schedule memory episodes of many threads to reduce interference. FST, proposed by Ebrahimi et al. [10], improves fairness in a shared memory system by source-throttling applications that interfere with others based on explicit measurements of application interference. Both MTL and FST are orthogonal to ACT, and ACT can be combined with mechanisms in other parts of the shared

memory hierarchy.
Cluster-Based Performance and Fairness Techniques: Thread Cluster Memory scheduling (TCM) [23] is a memory scheduler that targets performance and fairness. TCM groups threads into clusters for request prioritization, and performs a particular shuffling algorithm to ensure fairness. In contrast, ACT uses round-robin clusters as a simple way to turn throttling on and off for groups of applications. TCM and ACT work in different subsystems and solve orthogonal problems.

8 Conclusion

We presented Adaptive Cluster Throttling (ACT), an application-aware throttling mechanism that improves performance and fairness of a NoC-based CMP system. ACT partitions applications into *clusters* based on network intensity, and then alternately throttles some of the clusters in a timesliced way. ACT improves the average performance by 11.9% (10.2%) and provides a fairness improvement of 14.5% (15.1%) on average compared to traditional bufferless 4x4 (8x8) NoCs under high load. It retains the benefits of a simple bufferless router design in reduced power and area, while bridging nearly half the performance gap of such networks with traditional buffered networks at high load. Thus, a bufferless NoC with ACT could be a compelling option for interconnect in general-purpose CMPs.

9 Acknowledgments

We thank members of CALCM (Computer Architecture Lab at Carnegie Mellon) for their insightful feedback and contribution of ideas to this work. We gratefully acknowledge the support of NSF CAREER Award CCF-0953246, NSF Grant CCF-1147397, Gigascale Systems Research Center, Intel Corporation ARO Memory Hierarchy Program, and Carnegie Mellon CyLab. We also acknowledge equipment and gift support from Intel.

References

[1] S. Borkar. Thousand core chips: a technology perspective. *DAC-44*, 2007. 2

[2] F. Cazorla et al. QoS for high-performance SMT processors in embedded systems. *IEEE Micro*, 24(4):24 – 31, Jul-Aug 2004. 3, 8

[3] H. Cheng, C. Lin, J. Li, and C. Yang. Memory latency reduction via thread throttling. *MICRO-43*, 2010. 15

[4] W. Dally. Virtual-channel flow control. *IEEE Par. and Dist. Sys.*, 1992. 3

[5] W. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE TPDS*, 4(4):466 – 475, Apr 1993. 15

[6] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004. 3

[7] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *DAC-38*, 2001. 2

[8] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Application-aware prioritization mechanisms for on-chip networks. *MICRO-42*, 2009. 3, 5, 8, 15

[9] J. Duato et al. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. *HPCA-11*, 2005. 15

[10] E. Ebrahimi, C. Lee, O. Mutlu, and Y. Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. *ASPLOS*, 2010. 3, 15

[11] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *MICRO-41*, 2008. 8, 10

[12] C. Fallin, C. Craik, and O. Mutlu. CHIPPER: A low-complexity bufferless deflection router. *HPCA-17*, 2011. 1, 2, 3, 7, 9, 11, 12

[13] R. Gabor and S. W. adn A Mendelson. Fairness and throughput in switch on event multithreading. *MICRO-39*, 2006. 3, 8

[14] C. Gómez et al. Reducing packet dropping in a bufferless NoC. *EuroPar-14*, 2008. 1, 2

[15] P. Gratz, B. Grot, and S. W. Keckler. Regional congestion awareness for load balance in networks-on-chip. *HPCA-14*, 2008. 2, 15

[16] B. Grot, J. Hestness, S. Keckler, and O. Mutlu. Express cube topologies for on-chip interconnects. *HPCA-15*, 2009. 2

[17] M. Hayenga, N. E. Jerger, and M. Lipasti. Scarab: A single cycle adaptive routing and bufferless network. *MICRO-42*, 2009. 1, 2

[18] S. A. R. Jafri, Y.-J. Hong, M. Thottethodi, and T. Vijaykumar. Adaptive flow control for robust performance and energy. *MICRO-43*, 2010. 2, 9, 15

[19] J. Kim. Low-cost router microarchitecture for on-chip networks. *MICRO-42*, 2009. 2

[20] J. Kim and W. Dally. Flattened butterfly: A cost-efficient topology for high-radix networks. *ISCA-34*, 2007. 2

[21] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. Das. A low latency router supporting adaptivity for on-chip interconnects. *DAC-42*, 2005. 15

[22] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: a scalable and high-performance scheduling algorithm for multiple memory controllers. *HPCA-16*, 2010. 3, 8

[23] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread cluster memory scheduling. *MICRO-43*, 2010. 8, 16

[24] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of on-chip networks using deflection routing. *GLSVLSI-16*, 2006. 2

[25] K. Luo et al. Balancing thoughput and fairness in SMT processors. *ISPASS*, 2001. 3, 8, 10

[26] G. Michelogiannakis, D. Sanchez, W. Dally, and C. Kozyrakis. Evaluating bufferless flow-control for on-chip networks. *NOCS*, 2010. 2

[27] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. *ISCA-36*, 2009. 1, 2, 3, 7

[28] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. *MICRO-40*, 2007. 3, 8

[29] G. Nychis, C. Fallin, T. Moscibroda, and O. Mutlu. Next generation on-chip networks: What kind of congestion control do we need? *Hotnets-IX*, 2010. 2, 4, 6, 7, 9, 10, 15

[30] A. Singh, W. Dally, A. Gupta, and B. Towles. GOAL: a load-balanced adaptive routing algorithm for torus networks. *ISCA-30*, 2003. 15

[31] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. *ASPLOS-9*, 2000. 8

[32] Standard Performance Evaluation Corporation. SPEC CPU2006. `http://www.spec.org/cpu2006`. 3, 7

[33] M. Thottethodi, A. Lebeck, and S. Mukherjee. Self-tuned congestion control for multiprocessor networks. *HPCA-7*, 2001. 15

[34] H. Vandierendonck and A. Seznec. Fairness metrics for multi-threaded processors. *IEEE Computer Architecture Letters*, Feb 2011. 8

[35] H. Wang, X. Zhu, L. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. *MICRO-35*, 2002. 9

[36] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. *ISCA-22*, 1995. 7, 12