Investigating the Viability of Bufferless NoCs in Modern Chip Multi-processor Systems

Chris Craik craik@cmu.edu
Onur Mutlu onur@cmu.edu

Computer Architecture Lab (CALCM)
Carnegie Mellon University

SAFARI Technical Report No. 2011-004

August 29, 2011

Abstract

Chip Multi-Processors are quickly growing to dozens and potentially hundreds of cores, and as such the design of the interconnect for on chip resources has become an important field of study. Of the available topologies, tiled mesh networks are an appealing approach in tiled CMPs, as they are relatively simple and scale fairly well. The area has seen recent focus on optimizing network on chip routers for performance as well as power and area efficiency. One major cost of initial designs has been their power and area consumption, and recent research into bufferless routing has attempted to counter this by entirely removing the buffers in the routers, showing substantial decreases in NoC energy consumption.

However, this research has shown that at high network loads, the energy benefits of bufferless schemes are vastly outweighed by performance degradation. When evaluated with pessimistic traffic patterns, the proposed router designs significantly increase network delays in last level cache traffic, and can lower the throughput of the system significantly.

We evaluate these router designs as one component of the entire memory hierarchy design. They are evaluated alongside simple cache mapping mechanisms designed to reduce the need for cross-chip network traffic, as well as packet prioritization mechanisms proposed for high performance.

We conclude, based on our evaluations, that with intelligent, locality-aware mapping of data to on-chip cache slices, bufferless network performance can get very close to buffered network performance. Locality-aware data mapping also significantly increases the network power advantage of bufferless routers over buffered ones.

1 Introduction

The past few generations of Chip Multi-Processors(CMPs) have seen great leaps in core count, such as the 100-core Tilera[29], or the 48-core Single Chip Cloud Computer[13]. As the number of cores on chip grows[23, 1], the interconnection fabric that connects them has become a significant consumer of power and area. Some of these designs, such as [9] and [12] have shown the interconnect to be a major consumer of power and area.

The mesh network on chip has been an appealing approach to interconnection of resources[24], largely for its general simplicity and ease of implementation in a tiled CMP. We focus our evaluations on such a design, specifically an 8-by-8 mesh of symmetric nodes. As shown in Figure 1, each node contains a core, a private L1 cache, a shared L2 cache slice, and a simple router.

The on chip interconnect serves as the communication layer between tiles. Memory accesses that can't be served by the tile on which the application is running are forwarded to remote shared cache slices, and on to memory if needed.

The network on chip also serves cache coherence requests, forwarding state information in order to keep shared memory consistent between cores. Network transmissions are on the critical path for accesses that don't hit in a local cache, as well as communication between cores, and can thus be critical to system performance.

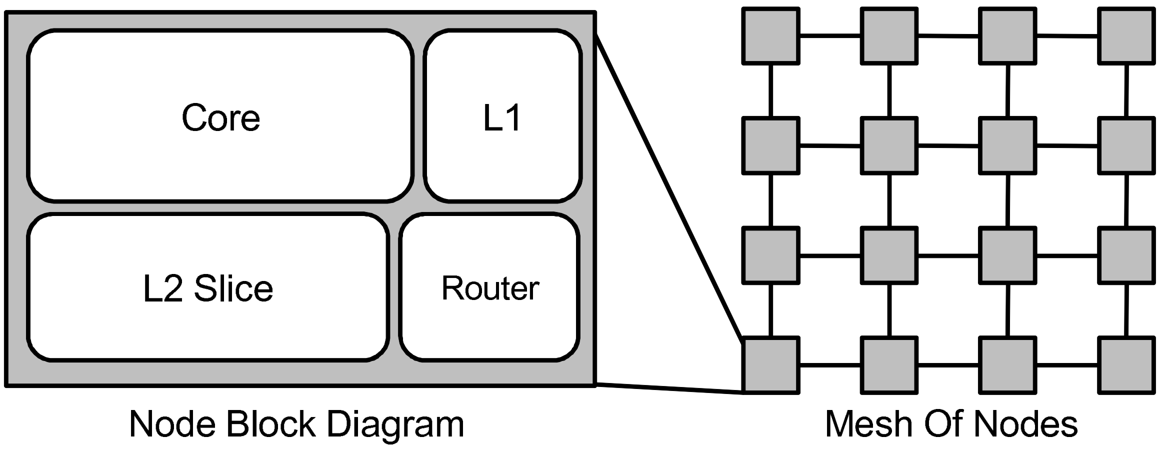

Figure 1: Tiled Mesh CMP Overview

The routers, though somewhat similar to those used in large scale networks, operate under very different conditions with very different design tradeoffs. They operate at very high frequency with very low latency, and complex implementation and buffering come at great expense because of on chip constraints. They forward packets of control and data information in units of flits - the smallest granularity of NoC packet routing.

In this work, we analyze the network on chip as a component of the memory system, and examine how both buffered and recent bufferless designs affect performance and energy. We begin with an overview of previous buffered and bufferless routing designs.

2 Background

The network on chip has the collective job of forwarding control and data packets between nodes in the system. Any access to a remote node for cache or memory state goes over this network. It is then the job of the network on chip routers to forward packets to connected routers so that packets may be ejected from the network at their destination node.

In our investigation, we focus on the mesh topology of interconnected routers. In the mesh topology, all nodes are laid out in a two dimensional grid, and are connected to all neighbors (up to 4). While other topologies have been investigated, the simple implementation and scalability of the mesh have made it the most popular topology for large modern CMPs.

2.1 Buffered Virtual Channel Routing

Much of the work on mesh network on chip routing has been based off the idea of virtual channel buffered routers, originally proposed in [4]. Because contention for a link may occur between two simultaneously arriving flits, a buffered router stores incoming packets into queues first. Then it may select from the heads of these queues in selecting outputs. This solves the problem of packets contending for outputs because a buffered router can simply stall the losers of the arbitration for a contended output port. If an input queue fills up, the router can exert backpressure on the router feeding it, telling it to stall packets destined to the full input buffer.

Virtual channels are frequently added to buffered routing architectures, in the form of multiple input queues. Having multiple queues per input limits the problem of *Head-of-line blocking*, where a packet in the front of the queue stalling forces all subsequent packets to stall as well. With multiple input queues, even if the first packet in one queue is stalled (due to backpressure or link contention), packets in other queues can route to other free inputs, and utilize the outgoing links.

2.2 BLESS

In BLESS[22], the authors suggest using bufferless routing to enable cheaper router design points. As buffers consume a lot of power and area, removing them from the router is an appealing possibility.

Whereas buffered routing will buffer or stall packets that can't make progress, bufferless routing has no internal buffers, and no flow control to create backpressure. Instead, bufferless routing relies on deflection of packets that don't make progress. If each packet cannot obtain a desired output port, it is rerouted over an undesired link, or 'deflected.'

Ideally these deflections are rare, because they create additional network traffic and congestion. Since the flits are never stored in the router, the buffers of the router are removed to save silicon and energy.

The authors point out that several problems in bufferless routing that are very different from those in buffered. Whereas buffered designs must avoid circular dependency deadlock, bufferless routers do not need to because flits remain in motion constantly, without backpressure. Bufferless routers can however suffer from flit live-lock - where flits can be indefinitely deflected. To solve this problem, the authors evaluated several prioritization schemes and eventually settle upon age based prioritization. This grants a total order of the priorities of flits in the network, but has the drawback of adding complexity.

The final proposed design includes a sort network to rank flits at a router by age, and then allocates ports in priority order. Figure 2 shows these components. On the left, flits are sorted by age and by a three stage sort network. The output of this stage is a sorted order of all incoming flits.

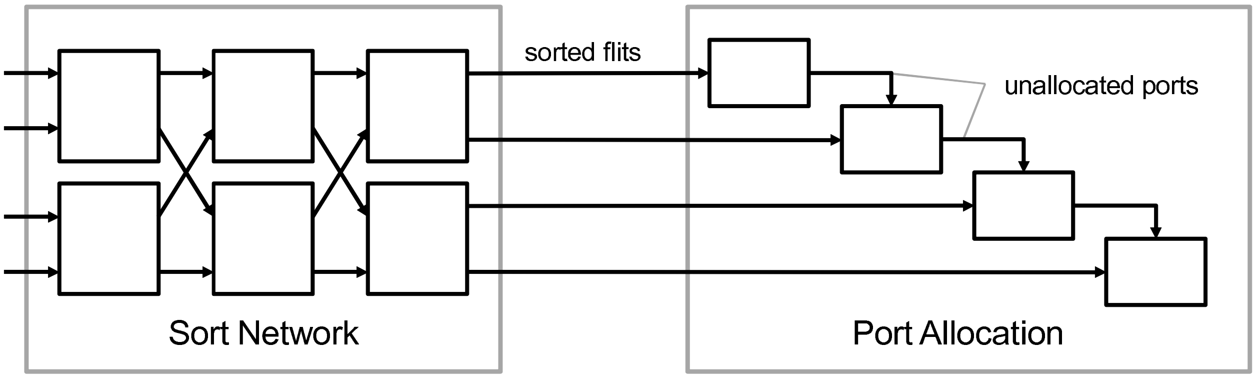

Figure 2: FLIT-BLESS Routing Components

Port allocation, the second component of BLESS-routing, assigns ports to flits in priority order. The first allocator block gives the oldest flit choice of any desired output link. The second flit is given choice over the remaining, and the fourth incoming flit (if present) is allocated the last remaining outgoing link. This directional information is passed finally to the crossbar (not shown) to direct the flits to neighboring routers. We don't mention the details of injection and ejection here, except to say that they are largely treated as just another pair of incoming and outgoing ports.

The authors evaluate their router's performance on an in-house simulator with multi-programmed workloads. They use a striped mapping of cache blocks to L2 slices, where blocks mapped to a last level cache slice by their least significant tag bits. They use memory without modeling the queueing of requests.

These design decisions put a significant strain on the network, especially with applications that miss frequently in the L1. The authors note that their router designs perform poorly in the high load case, when deflection rates ramp up dramatically.

2.3 CHIPPER

In CHIPPER[7], the authors identify the complexity, and resulting long critical path, make the original BLESS design difficult to implement. CHIPPER focuses on reducing complexity in order to further motivate the bufferless design point with reduced energy and area. While the authors also simplify reassembly buffers and provide an implicit token-based infrastructure for live-lock free deflection routing, that is not relevant to the discussion of bufferless performance in the high load case.

CHIPPER uses a permutation network composed of four two-input/two-output permuter blocks. Each block makes strictly local prioritization decisions, acting effectively as a 2x2 deflection router. The input flits are permuted in order

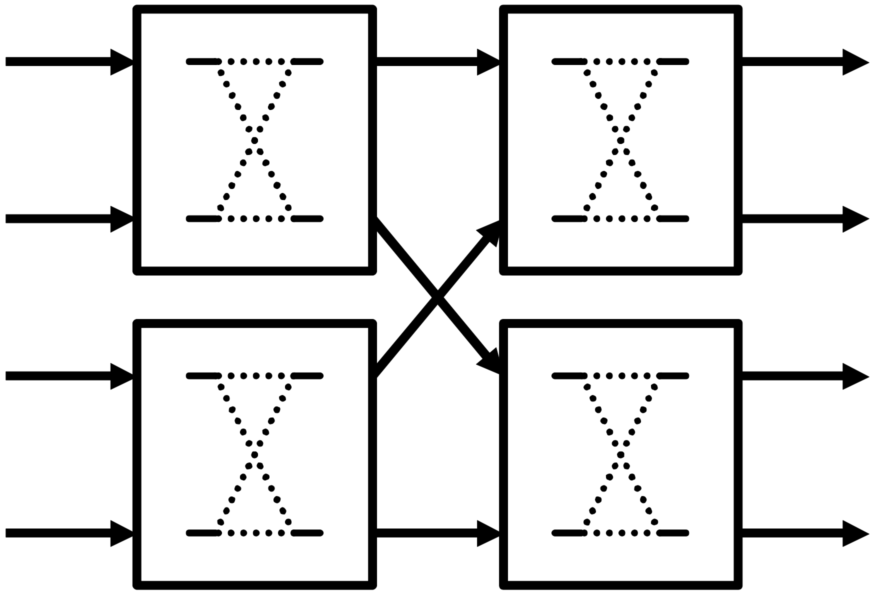
Figure 3: Partial Permutation Network

on the output links, with injection and ejection handled in a separate router stage. In this way, the lengthy priority sort and port allocation steps are replaced by much simpler operations, and flits are still routed where possible, or otherwise deflected.

One drawback of the simple permutation network routing is that it's only partially permuting. Due to the fixed layout of the available routes, contention for the links prevents certain permutations of the input, and creates deflections where a standard BLESS router would not. A simple example of this can be seen in the figure, where the top two links are occupied by flits desiring to travel straight across the router to the top two output links. One flit must be deflected down into the lower second stage permuter, deflected away from its desired links.

Because deflections occur where they would not in a BLESS router, deflection rate increases and performance is further sacrificed in the high load case. The paper argues that the high load case is not its focus, and not the only case of interest. At the very least, the further power savings offered by the simplified router architecture make it a design point worth considering. Due to space constraints however, we do not present results for CHIPPER-style routers in our evaluations.

2.4 Data Mapping

As the L2 cache is distributed amongst tiles, it can be used with different mapping mechanisms to advantage different patterns of access. Strictly private use of the L2 slices favor working sets that fit locally, and are simple - they are a natural extension of the private L1. Strictly homogeneously shared L2 slices are simple to use as well, but grant better sharing of data between tiles while sacrificing network latency.

Recent studies have investigated alternate designs[10, 27, 2, 3, 19] in between these extremes to find a good general solution. These studies, however, have not incorporated this investigation into the selection in another important memory system parameters, that of on chip network. We will evaluate bufferless routing with different simple data mapping mechanisms and show that high-load performance can be significantly improved with locality-aware data mapping. Our focus has been on simple mapping techniques; more sophisticated techniques can lead to even higher performance.

2.5 Prioritization

Traditionally, each packet and flit in the network is treated equally, no matter its purpose or destination. Recent studies[5, 6] have shown the affect of in-network prioritization of packets, so that packets that are more important to system performance are prioritized over others.

We examine the interaction of application-aware prioritization and different router architectures. We show that while prioritization can work for bufferless routers, its effect is significantly less pronounced than with the buffered baseline and the effects are further reduced when alongside more efficient mapping.

2.6 Other Related Work

Prior work has investigated different approaches to Bufferless routing in networks on chip. BPS[8] presents a drop-based bufferless network, that disposes of and resends packets to handle contention. SCARAB[11] builds upon this work by adding an additional network of circuit switched NACK lines to simplify retransmission. In [14], the authors propose a combined buffered/bufferless system that switches between the two modes of routing. This enables the network to avoid the inefficiency of bufferless networks in heavy traffic, but to take advantage of its energy savings(except potentially leakage energy) in the common case. Other work[15, 16] has investigated means of reducing the buffering requirements with alternate buffered designs. In [20], the authors evaluated buffer bypassing and show that it can improve energy efficiency of a buffered network, but buffer bypassing cannot for all network settings match leakage power and area reduction advantages of bufferless network designs. Due to the wide range of alternative designs proposed, and to space and time constraints, we do not present results for these alternate designs.

3 Evaluation Methodology and Baseline Results

The evaluation methodology used to compare different NoC alternatives greatly affects performance, and thus conclusions to be drawn from experimentation. We describe in detail the system we simulate, and discuss some drawbacks in simulation methodology that previous evaluations have used.

3.1 System Overview

For our evaluations, we use an in-house cycle-accurate CMP simulator to evaluate different design points. The system consists of a 8x8 mesh of 64 cores, with a processor, L1 cache, router, and L2 cache slice at each node. There are 4 memory controllers, one at each corner, with DDR3-1866 timing (933 MHz clock)[21] and accurate FRFCFS[25, 17] scheduling and queueing modelled.

We model the network with a single cycle link delay, coupled with a two cycle router pipeline for all architectures. All simulations presented herein are warmed up for 50 million instructions, and evaluated over 20 million cycles. We use 20 million cycles to reduce simulation time and find that our results are representative of much longer intervals on the order of billions of cycles (verified using a representative subset of workloads).

Our simulations are driven by representative instruction traces, sampled from benchmark applications with PinPoints[26]. For single threaded applications with deterministic behavior, recording and playing back the stream of instructions provides equivalent control and data flow to full simulation with a considerable decrease in simulator complexity. Including annotations for each memory address and simulating the memory system accurately allow instruction driven traces to be cycle accurate.

Parameter	Setting
System Topology	8x8 mesh, Core and L2 Cache Slice at each node
Core Model	Out-of-order x86, 128-entry instruction window
Private L1 Cache	32KB, 4-way associative, 64-byte block size, 16MSHRs
Distributed L2 Cache	1MB, 16-way associative per slice, 16MSHRs
Main Memory	4 MCs, one per corner, DDR3-1866 Timing[21], FRFCFS Scheduling[25, 17]
Coherence Protocol	Simple directory, based on SGI Origin[18]
Interconnect Links	1 cycle latency, 8 flits per cache block
Baseline Buffered Router	2-cycle latency, 4VCs/Channel, 8 flits/VC
Aggressive Bufferless Router	2-cycle latency, FLIT-BLESS[22] Sequential Allocating Bufferless Router
Lightweight Bufferless Router	2-cycle latency, Partial Permutation Router[7]

Table 1: System Parameters

3.2 Synthetic NoC Evaluations

Synthetic evaluations give two major benefits over execution and trace driven simulation. First is the decrease in programming complexity and corresponding increase in simulation speed. The second is simplicity of evaluation, because sweeping across synthetic design points theoretically offers complete experimentation of the problem space. For the second of these reasons, we present a synthetic comparison of a bufferless and baseline buffered router. Because the first is not motivation enough to decrease simulation fidelity, we evaluate with instruction trace driven simulations in other sections.

Some synthetic evaluations use open-loop requests without a simulated core or instruction window. They show latency growing to infinity as injection rates increase, as the network at some point cannot keep up with the synthetic injection rate. However, the closed-loop nature of the memory hierarchy prevents this in a real world system. In our simulations, we use a self throttling network, i.e. a processor stalls when its outstanding request queue is full and a new remote request is attempted. This is an important factor, as any real system will not have infinite queueing capabilities.

In a self-throttling system, latencies are quite finite - as our evaluations with 1000 L1 MPKI illustrate. Open-loop latency curves are sometimes used to compare routers, but in reality are just used as proxies for performance differences. For this reason, we choose to evaluate synthetic traffic in a closed loop system, and present real performance results.

In Figure 4 we compare the buffered baseline router against the BLESS router, using synthetic memory requests. The processor in each node generates a trace of random memory accesses, with each being served by a remote L2 slice. We sweep across different request frequencies, from nodes not accessing remote memory to every instruction being an off-node request. Network transaction latency, as shown, is the time between L1 MSHR allocation and the writeback of data into the MSHR, including all time spent in and waiting to enter the network.

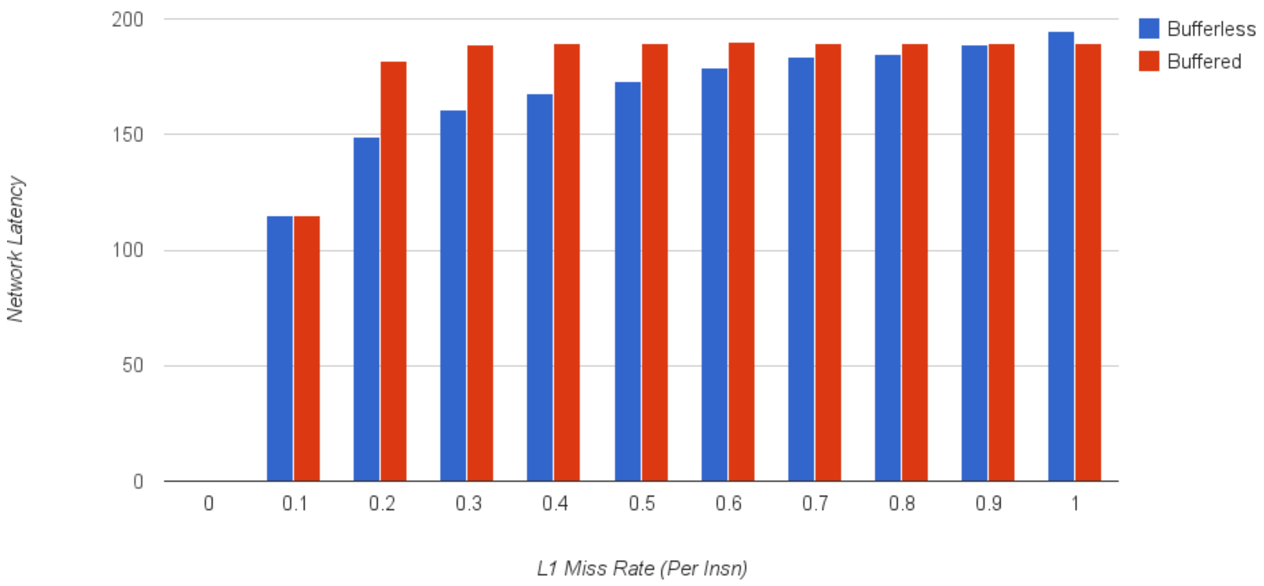

Figure 4: Synthetic Latency

At first glance, the lower latency despite deflections in the bufferless network seems illogical. The bufferless network despite its inherent deflections shows lower latency in all but the most network intensive case. Even if 70% of instructions are network requests, the latency of bufferless is lower than that of buffered.

This makes sense however, when we consider that while bandwidth of the network is lowered, the number of active transactions decreases as well. Deflections lower the bandwidth of the network as links are occupied by unproductive flits. This doesn't necessarily increase the latency of a single transaction though. In the baseline buffered network, latencies aren't necessarily lower - packets are often stalled waiting in virtual channels.

The main difference of the bufferless network is that fewer transactions tend to happen in parallel. In buffered routers, arbitration has multiple virtual channels per input port to select a packet from, and thus more options to use the links wisely. With this, the network is able to handle larger numbers of outstanding packets and even if each individual transaction is slower, the aggregate throughput is still greater.

This is somewhat like two alternate designs for a grocery store checkout. Four clerks with lines of length N are on the whole always more effective than three clerks with lines of length 2N. From the store owner's perspective, if each register has a line, (remember that we're considering a fairly contended resource) more customers are getting through. Latency is often a poor proxy for throughput or in the case of NoCs, system performance.

As in a real system, the closed loop of memory requests forces the processor to stall when the network can't fulfill requests. Because fewer requests are in the network simultaneously, the latency of each network transaction is in most cases less than buffered.

In Figure 5, the performance comparison of the experiment, this is shown to be the case. A 10% request rate, or an L1 miss rate of 100 MPKI is actually quite intense, saturating the bufferless network, but with fewer requests stalled waiting for network injection to drive up the average latency. The graph of relative network throughput is very similar, and thus not shown.

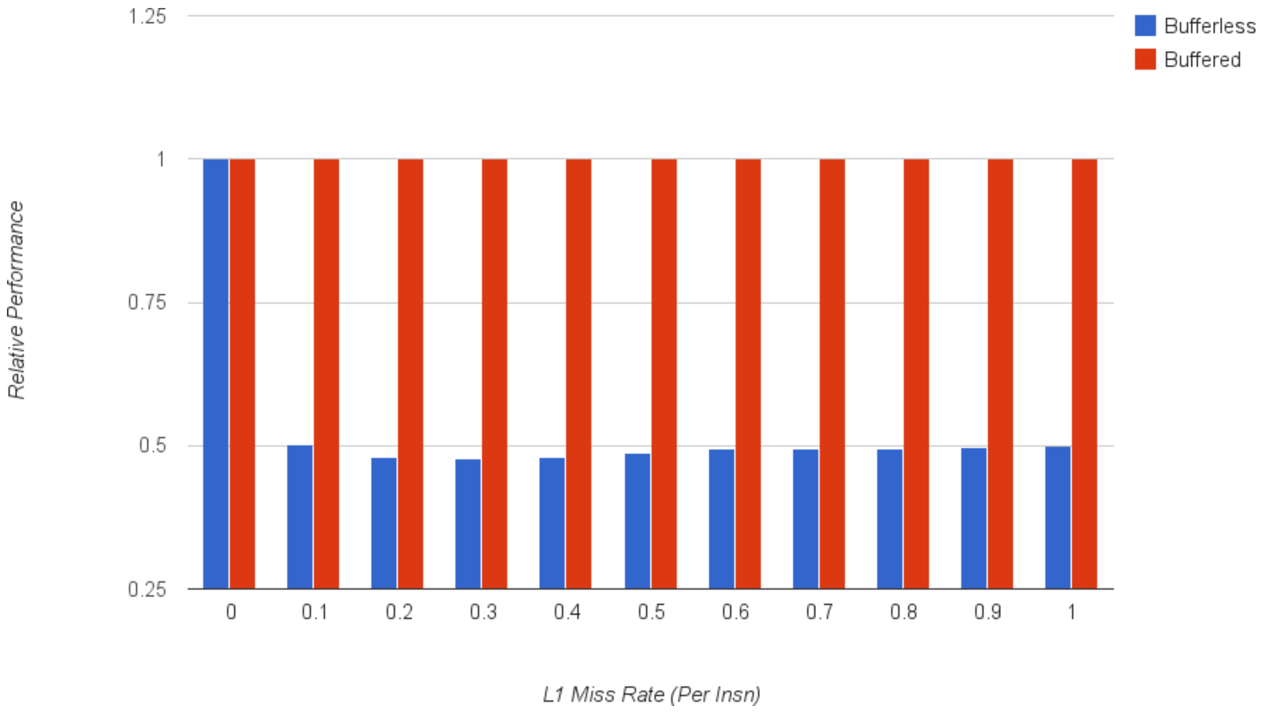

Figure 5: Synthetic Performance

3.3 NoCs in the Memory Hierarchy

The network on chip serves an important duty in the memory hierarchy of a CMP. It ferries requests and data across different nodes, and allows access of remote memory - both on-chip and off. It is thus important that NoCs be evaluated in context, in a simulated real system.

However, previous evaluations of bufferless networks have sometimes neglected fidelity in the modelling of other components of the memory hierarchy. BLESS[22] used a simple memory approximation, with no queueing and no in-network representation of memory controller interfaces. This artificially prevents memory intensive workloads from stalling one another in DRAM, in turn putting artificial strain on the network. CHIPPER[7] went even further by removing memory entirely in its evaluation.

Both of these papers had motivation for evaluating a system without memory contention - they evaluated their proposed designs in a network-intense system to bound the worst case. They showed that there was potential for lightweight router designs; that there are many workloads for which the baseline buffered NoC was overprovisioned even with aggressive network usage.

The approach of this work, by contrast, is to look at the common case. We explore the differences between the proposed routers in a more realistic system, simulating memory controllers and queues.

3.4 Workload Selection

In selecting a set of workloads, we aim to investigate the effect of network design decisions on different network loads. To help isolate the variety of intensities applications can have in the network, we classified benchmarks as either high or low network intensity. Benchmarks were classified based on their observed flit traversals per cycle, a metric to show how much traffic they created in the network into two equal groups. Each benchmark was duplicated across all 64 cores in the system in a classification run to ensure all system resources were taxed realistically. While the metric we used shows some variation with system and network parameters, its general trend in classification of the benchmarks remains fairly constant.

We then built five groups of eight workloads each, for a total of forty workloads. Each workload in a group shared the percentage of high intensity benchmarks, e.g. a 50% intense workload would contain 32 network intense benchmark instances, and 32 non-network intense benchmarks. Groups were created for 0%, 25%, 50%, 75%, and 100% intensity.

We then populated both the low and high intensity portions of each workload randomly from the available benchmarks. We expect that, as other studies have demonstrated, the workloads with low relative intensity will not be strongly affected by network design decisions, as they are not network bound. Nevertheless, we present data from all five classifications where possible, and averaged elsewhere.

4 Mapping

The mapping of data to last level cache slices in tiled CMPs has been an active research area [10, 19, 2]. The growing consensus is that mechanisms that dynamically control the placement of data significantly outperform simple private caches or naive fully shared caches.

4.1 Private

One of the simplest methods for cache mapping is to use a purely local private last level cache. Each time an application misses in the L1, it searches for the block in the local L2. Because this operation is entirely local, no network transactions are required unless the request then misses in the local L2.

Private last level caches can suffer from a space overhead of holding data in multiple places. If data is shared between different threads in an application, it takes up space in each of their caches. This can decrease the number of unique resident cache lines available on chip at any time. If multiple threads share a large working set, private caches can increase the last level cache hit rate significantly, reducing throughput.

In addition, private last level caches statically partition cache space evenly between concurrently running threads. This may be a very bad decision if an L1 resident application effectively doesn't use its cache slice. Another nearby application with more data to fit in its local slice might have benefitted greatly from sharing that space. Every thread, regardless of its reuse pattern and working set size, is given the same amount of last level cache space.

4.2 Globally shared

In a naively striped, globally shared cache, each cache line is stored in a node designated by the last bits in the line number. This is 6 in the case of our 64 node system. Each contiguous set of 64 lines in memory would be stored with one line cached on every node in the system. Globally shared caches don't suffer from unnecessary duplication of data, as each block can only reside in one set in the entire distributed cache. Space is shared cooperatively between threads automatically; threads with large working sets use as much cache space as possible, while L1 resident threads don't occupy much L2 space.

However, naive CMP shared caches places data without awareness of topological locality. A block of data used exclusively by the top left core in the mesh is just as likely to reside in its user's slice as it is in the bottom right. This locality-unawareness causes most L1 cache misses to require a network transaction.

The latency of these network transactions is non-trivial, and can delay the processor for two reasons. First, the slice to which a cache line is mapped may be several hops away. Each packet transmission requires time proportional to this distance even in the best case. Because of this, the best-case latency of the network transaction will scale with the size of the network.

Benchmark Name	Traversals Per Cycle	Net Intensity	L1MPKI	L2MPKI
400.perlbench	0.019849	Low	0.288718	0.099737
454.calculix	0.023249	Low	0.706090	0.050428
481.wrf	0.023345	Low	0.578421	0.000429
447.dealII	0.038359	Low	1.141246	0.093626
crafty.5Bskip.100M	0.081927	Low	3.626324	0.062634
465.tonto	0.097925	Low	3.479114	0.011835
458.sjeng	0.101914	Low	1.861570	0.344768
464.h264ref	0.111647	Low	2.439626	0.428448
435.gromacs	0.161187	Low	5.706131	0.173472
444.namd	0.163496	Low	5.507062	0.067003
shpoint91s	0.188706	Low	5.410433	0.730275
shpoint9ebs	0.189209	Low	5.472909	0.728529
445.gobmk	0.199498	Low	4.299901	0.509944
403.gcc	0.234362	Low	11.389455	0.212362
453.povray	0.258553	Low	12.402468	0.001758
401.bzip2	0.419965	Low	11.487220	0.891355
vpr.60Bskip.100Mcomplete	0.421227	Low	17.706930	0.738427
436.cactusADM	0.464606	Low	7.305476	4.812732
456.hmmer	0.484417	Low	5.300028	2.729838
482.sphinx3	0.553152	High	16.235336	13.155198
471.omnetpp	0.561466	High	17.643306	16.776877
483.xalancbmk	0.569268	High	22.416194	16.545166
433.milc	0.604465	High	13.861983	13.859931
429.mcf	0.628157	High	101.490791	72.263515
ms_1_15_3	0.636264	High	9.972153	2.395367
459.GemsFDTD	0.648010	High	27.217091	27.102799
437.leslie3d	0.683358	High	20.432670	5.244858
tpcc	0.707561	High	28.604974	1.441570
473.astar	0.715480	High	7.403750	5.576512
450.soplex	0.718929	High	34.167689	26.794749
stream.100M	0.723571	High	35.717634	35.716793
xml_trace	0.725859	High	31.989659	29.334220
462.libquantum	0.728699	High	32.274022	32.273665
ms_1_15_50	0.757180	High	19.429075	3.886776
470.lbm	0.853376	High	44.903540	27.191799
health.50Mskip.100M	0.854957	High	56.863482	50.829364
art.ref.train_match.100M	0.868367	High	79.374168	15.745927
matlab3	0.895579	High	253.311321	112.464095
mcf.24Mskip.100M	0.914853	High	57.529701	5.930129

Table 2: Benchmark Classification

Second, if the network is highly contended, many network transactions may not see best case latency. As many packets are traversing large distances across the network, they contend for links. Only one flit can traverse a particular link at once; any other flits are blocked (given buffered routing) or deflected (bufferless routing), adding delays to their round-trip time. The network overhead of this simple striping mechanism motivates the investigation of locality-aware mechanisms.

4.3 Sharing Regions

Because striping blocks across all nodes in the network isn't efficient for large network sizes, we also investigated smaller regions of striping. Smaller regions benefit from shorter network distances, while still providing the ability for applications to share space. By striping at a fixed size region, last level cache traffic becomes immune to the distance-scaling concerns of global striping.

This method applies, in our system, to two different sub-region sizes, 2x2 and 4x4. These can be seen as midpoints between the 8x8 global striping approach, and private caches (i.e. 1x1 striping). For brevity, however, we only present data for 2x2 striping, which outperforms 4x4 in our evaluations.

These mapping regions can be achieved one of two ways - 1) different regions cache discrete portions of physical memory, or 2) regions can exist as separate coherence domains. In 1), the OS allocates pages to each thread striped across its home region. Discrete regions of the physical address space are mapped to each region, so they are automatically coherent at the L2 level, as with a full striped mapping. In 2), regions can be thought of as separate caches each able to contain its own copy of a cache block. For this reason, they would require a coherence substrate between them.

We choose method 1) for its simplicity, and depict this visually in Figure 6. Using this page based mechanism, private caches are simply implemented by allocating from *N* free lists, one per node.

Th tradeoff between private and globally shared caches is highlighted in these intermediate approaches. Smaller sharing grids offer less communication and less latency on a hit, but don't exploit heterogeneous access patterns. Smaller sharing regions can reduce hit rate if threads/applications that don't well utilize their full local L2 cache space can't share that space to other cores.

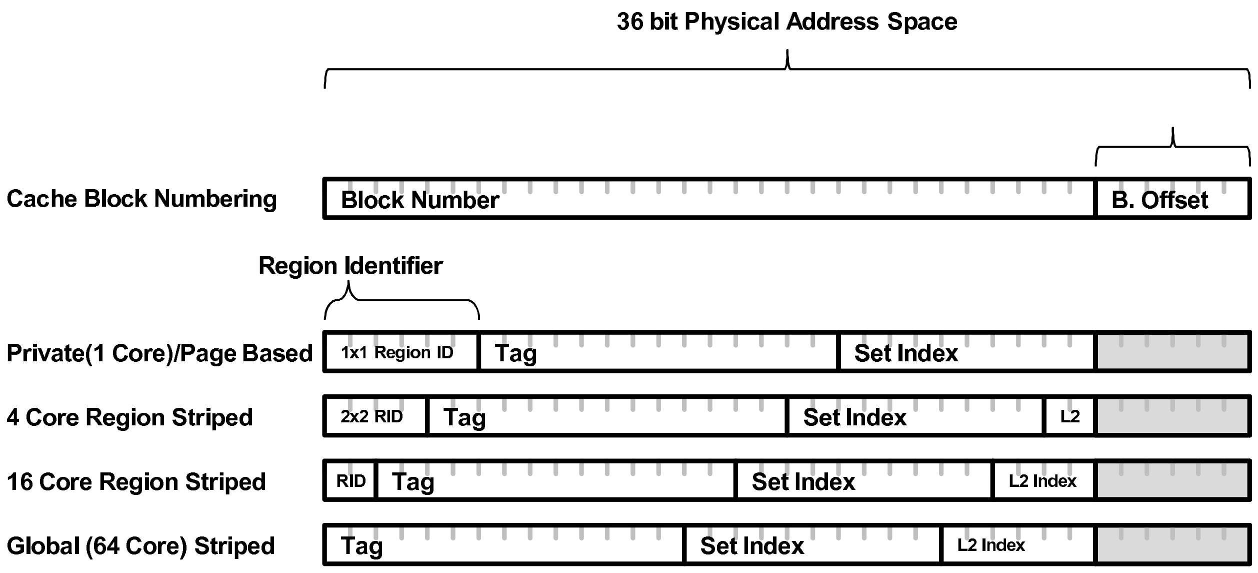

Figure 6: Mapping of Physical Address Space to L2 Cache Slices

4.4 OS Level Page Allocation

OS-driven cache mapping was proposed in [3], where the operating system would allocate physical pages to an application based upon the cache mapping of those pages. The operating system keeps a separate free page list of pages per node, where each page's physical address maps to the shared L2 space at the node. When allocating a region of memory for a node, the OS uses the free page list associated with nearby nodes, ensuring that some data is placed in shared space in nearby nodes.

This method benefits from simplicity, in that it doesn't require any more hardware than that of neighborhood sharing. However, the page-granularity allocations don't necessarily track with the regional usage. We found difficulty in evaluating this method in our simulations, as the time granularity we were using - millions of cycles - is not enough to capture full application allocation behavior. We provide this method for comparison, though its results should not be seen as best case.

4.5 Dynamic Spill Receive

Many of the previous mechanisms allow applications to share L2 space, but don't provide any means for partitioning it. The replacement policy is, in a sense, entirely in control of the shared space, and which application has what portion.

While the OS Level Page Mapping approach favors local placement of data, it still places applications equally. It is still ignorant of the differences in cache utility different applications may have.

In [27], the authors propose Dynamic Spill Receive (DSR) as a method for distributing data amongst shared caches. The mechanism is based on the idea of spiller and receiver caches. Some applications don't fit in their private slices, and would prefer to spill entries to other caches. Other applications don't need their full local slices, and could improve system performance by receiving the spills of others. Caches that spill store their evicted blocks in a receiver cache. Spiller caches, because of this, will pollute the remote caches, but do so with blocks that are likely to receive hits later.

The paper details a mechanism that decides whether each individual L2 cache slice should be a spiller or receiver. DSR does this by using specially mapped regions of the L2 cache that are statically allocated as spillers, and others as receivers. These are called the *training sets*. Each cache keeps track of which of these regions - the spiller or receiver regions - has a higher hit rate. Whichever has the higher hit rate determines the choice between spilling or receiving for the rest of the cache. This is similar to *set dueling* as seen in other caching mechanisms[28], often for selecting a replacement policy best suited to a particular workload.

The described implementation of DSR was designed for a set of caches sharing a bus to main memory. Cache blocks that are spilled from one cache are randomly received by any available receiving cache. This means that a spiller application could have its blocks in any other cache in the system. The shared system bus then made retrieving a spilled block simple, since a broadcast for the block was just a single bus operation, which would have been required in going to memory anyway.

4.6 Mesh Dynamic Spill Receive

In order to adapt this mechanism to a mesh network, we modified the method of selecting a receiver and finding a spilled cache line greatly. Because broadcasts are costly both in terms of latency and network utilization, it is very inefficient to have a broadcast for every miss in a spiller cache. By modifying the storing and searching policy, we were able to greatly simplify the task of selecting a receiver node to store a block and to search through receivers.

First, we partitioned the network into regions, based on the simplifying assumption that imbalance between the regions in utility would be outweighed by the benefits of the short intra-region distances. We evaluated region sizes of 2x2, 4x4, and 8x8, the full network. Unlike in the bus-based system, where querying memory and all receivers can be done simultaneously without additional complexity or communication cost, we optimized our design for shorter communication paths.

We found empirically that 2x2 sub-regions were large enough to usually have a receiver node under-utilizing its L2 cache. The major benefit of the 2x2 region was the shortening of critical path latencies. In our implementation, a miss in a spiller cache must query a receiver cache before requesting the data from memory. While we could have implemented a parallel request to memory and a receiver, it was our feeling that the additional complexity would be prohibitive, and results were promising without this additional optimization. As one block maximum could be considered a receiver for a spiller's block, it adds one network hop to a single L2 cache slice for the evictions and accesses of a spiller, illustrated in Figure 7.

We then made the decision that a spiller cache should only have to query one receiver for a given block to determine if the block is still on-chip. We greatly simplify the problem of communication with this assumption, but must additionally provide methods of predictably mapping a block to a given receiver, and balancing load between receivers.

We kept a cache of the receiver/spiller state of each region member in each cache, so that a given spiller node could know with reasonable accuracy which of the other caches in the region were receivers. The spiller, when evicting a block, would then iterate through the block number, trying to match a subsequence of the bits to a receiver's member ID. This process is illustrated in Figure 8. This mapping of block number to receiver is also applied when searching for a block on a local L2 miss. The ramifications in coherence design are out of the scope of this report, though they can be summarized by saying that the blocks stored in remote caches are always clean with respect to the rest of the memory hierarchy. Writebacks happen upon spilling a line, so that the line may be evicted silently by a receiver.

4.7 Mapping Comparison

Mapping has two major effects in the system which shape its effect on performance. First, it affects the network intensity of a workload by defining the distance travelled when sharing. Second, it defining the sharing patterns of applications and thus affects L2 hit ratios. In Figures 9 and 10, we show the effects of mapping on workload IPC. The mapping mechanisms show the same general trends with either baseline buffered or Bless routers. As mentioned

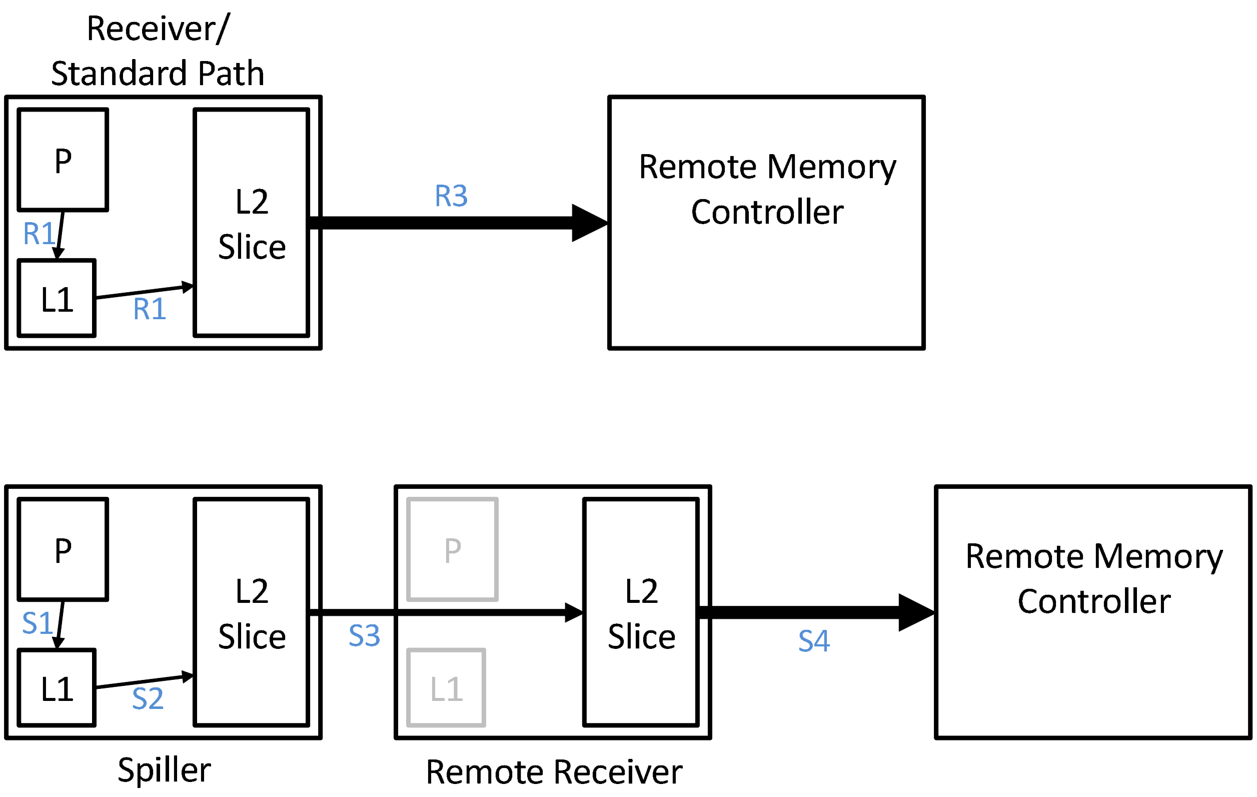

Figure 7: Path of Memory Access for DSR Nodes

previously, the workloads are sorted in groups of eight, with the leftmost being least network intensive, and the right being the most. Aggregate performance data is shown in Figure 11. The 0% point in this figure corresponds to the baseline buffered router, with striped mapping. All other points are the average IPC improvements of other evaluated schemes compared to baseline buffered routers with striped mapping across all workloads.

The naive striped mapping offers the worst performance, falling at least 7.7% in performance behind the other mechanisms in buffered. In the bufferless network, with its lower network throughput, the effect is more pronounced with the striped mapping falling 13.6% below the next lowest performer. We present the average network utilization, which we define as the average portion of active links, in Figure 12.

The cross-chip requests inherent in striped mapping require much larger numbers of traversals. This increase in traffic strains the network, running it at much closer to maximum capacity than any other mapping, a problem exacerbated by the deflections of the Bless router. The gap between the routers is widest here because the Bless network's deflections are worse in congested networks. It is worth noting that although the networks appear to be running far from maximum utilization, this does not account for heterogeneity in link utilization. We have observed that the central links in both the buffered and bufferless networks are significantly more congested than the average. As shown in Figure 13, the L2 hit ratio in the striped case is better than most others, but this does not overcome the network as a bottleneck.

The private mapping has very similar performance in both the Buffered and Bless, largely due to two things. First its aggregate hit ratio is worse than any other mapping, shown in Figure 13. This comes from the lack of sharing as discussed previously in this section. The low hit rate, in turn, causes the private mapped system to experience more memory access and queueing delays. The other major component to the similarity in performance is the lack of remote L2 accesses. Even though many requests must traverse the network to access a memory controller, the memory access delays far outweigh the networking delays, and the requests spend a smaller portion of time in the network, congesting it less than in other mechanisms. No requests that hit in the L2 utilize the network at all. The result of this low network utilization is that the routers are very lightly loaded, and not nearly the bottleneck they are with other mappings. The Bless network comes within 0.3% of the performance of the Buffered network with the private L2 cache mapping.

2x2 region mapping performs well, but shows significant differences between buffered and bufferless networks.

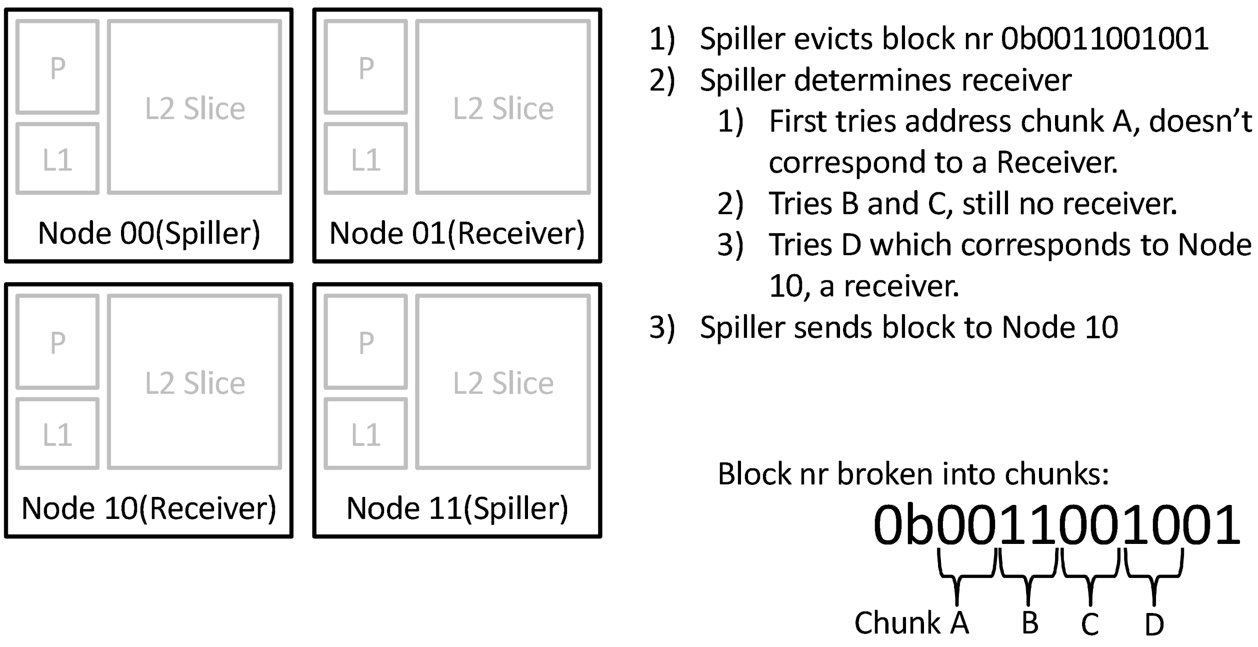
P
L1
L2 Slice
Node 00(Spiller)
Node 01(Receiver)
Node 10(Receiver)
Node 11(Spiller)
1) Spiller evicts block nr 0b0011001001
2) Spiller determines receiver
1) First tries address chunk A, doesn't correspond to a Receiver.
2) Tries B and C, still no receiver.
3) Tries D which corresponds to Node 10, a receiver.
3) Spiller sends block to Node 10
Block nr broken into chunks:
0b0011001001
Chunk A
B
C
D

Figure 8: Selection of a Receiver Node

Though the network utilization rates are shown to be significantly lower than the striped mapping, they are still highest amongst the remaining mapping scheme. On top of this, our region mapping scheme puts a much heavier use on the links used for intra-region communication. Deflection rates show that although the network utilization is roughly half of the striped mapping, the deflection in the bufferless implementation is much closer. Because the network intensity on these intra-region links is high, the bufferless router suffers 2.5% performance loss compared to the buffered baseline.

The OS Page Spilling mechanism provides lower performance in our evaluations than most other mechanisms, including the private mapping. We believe that this may arise from a discrepancy between the aggressiveness of the memory and network compared to the implementation and system devised in [3], as well as the duration of our simulations. Unfortunately, the tens of millions of cycles we use in our simulations are not enough to show real, long term allocation behavior in an application. We omit the OS Page Spilling results in further sections of our study for this reason.

DSR provides the best performance in both the buffered and bufferless cases, showing a larger performance improvement relative to striped mapping in the bufferless case. In buffered, its network utilization is almost exactly as low as that of private mapping, 13.4%. In the bufferless network, the network utilization is higher than private, but does not suffer as much in the Aggregate L2 Hit Ratio. Though spillers often need to query region members for blocks, the increase in L2 Hit Ratio that this affords compensates for the additional traffic. The DSR mapping provides 13.1% improvement with a buffered network, and 10.3% improvement with a Bless network compared to the baseline Buffered with Striped mapping. Thus, with DSR mapping, the performance gap between the bufferless and buffered network are significantly smaller than that with a naive striped mapping. We conclude that intelligent data mapping strategies in caches and memory controllers can lead to the network to become less of a bottleneck and can more easily enable cheaper, lower-throughput network implementations.

For the next sections, we continue our evaluations with three mapping mechanisms. We include the striped mapping to serve as a baseline, as it is the mapping used by much of previous work. We include the newly proposed DSR mapping as it performs the best of all evaluated mappings. We also include the 2x2 region mapping, which performs second best on the baseline Buffered network.

From the reduction in network strain afforded by the more efficient mapping mechanisms, we have shown the performance improvement of both the Buffered and Bufferless systems. With the baseline striped mapping, the system with the Bless network was running at 94.0% the performance of the Buffered system. With the best mapping mechanism applied, and even though the system with the Buffered network improved greatly, that gap narrowed. The Bless router, with DSR in 2x2 regions mapping performs at 97.5% the performance of the Buffered router using the same mapping. The DSR mapping reduces the network strain which eases the job of both routers, and allows the Bless

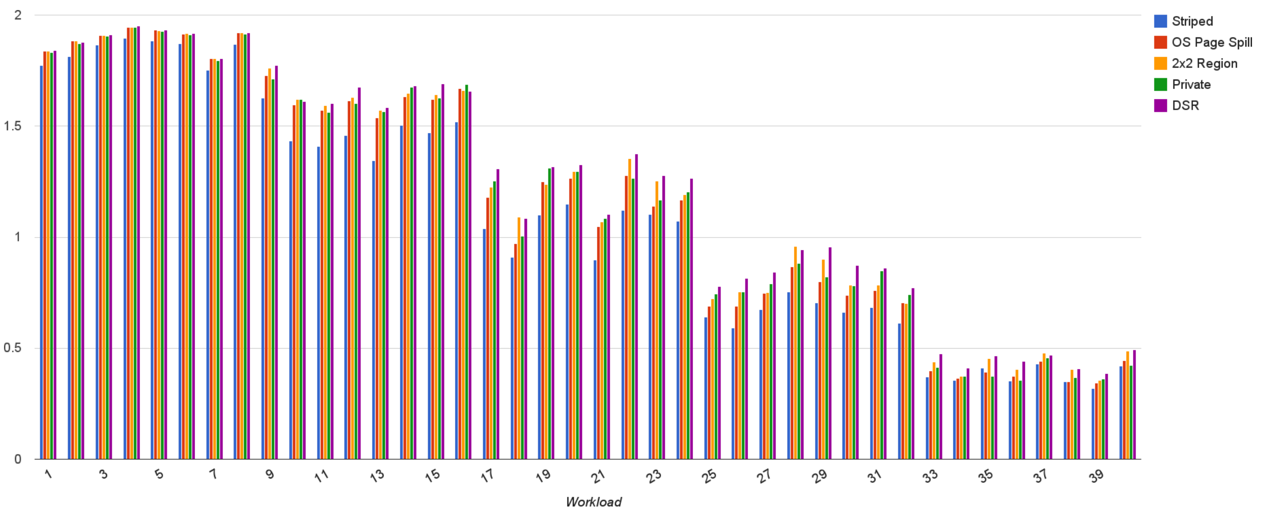

Figure 9: Mapping Effect on Per-Node IPC in Buffered Network

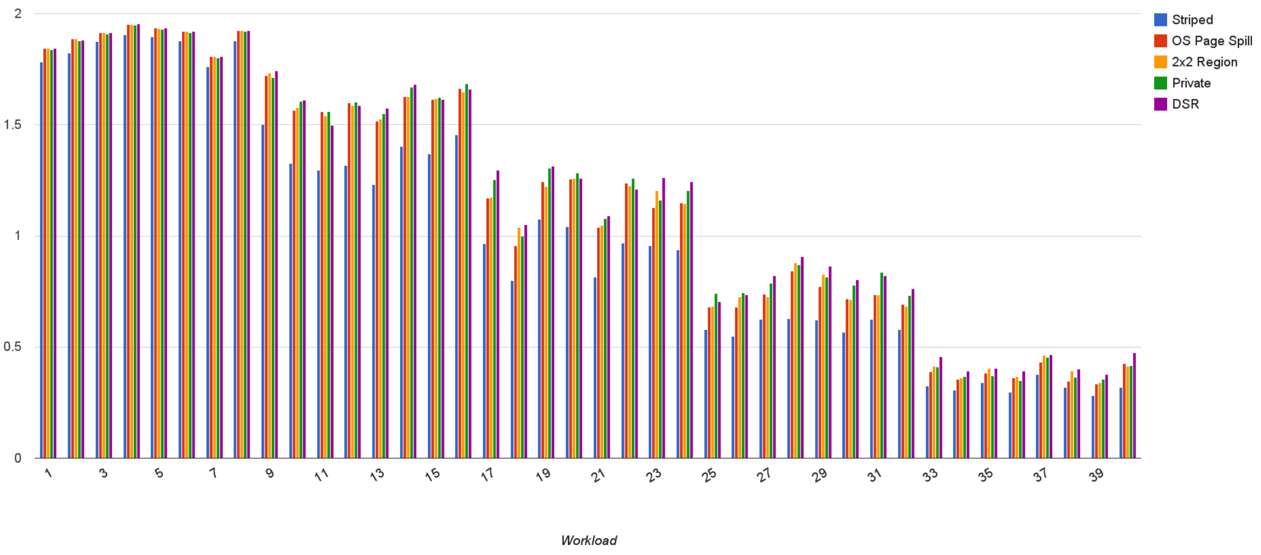

Figure 10: Mapping Effect on Per-Node IPC in Bufferless Network

router to perform very close to the buffered router. We conclude that the performance of bufferless routers become very competitive with that of buffered routers with intelligent data mapping mechanisms.

5 Prioritization

Because the links in a network are a contended resource, routers must make decisions about which packet traverses which link.

When multiple packets in a router contend in a given cycle, the router chooses which packet to prefer, or prioritize. In buffered routers, this is most often done through round-robin assignment of packets to output ports. Each input buffer takes a turn at reserving the crossbar in sequence. All input buffers and packets are treated equally by this arbitration, even though imbalance in the output and input queueing and blocking may occur.

Prioritization is the use of preference in packet/flit arbitration. Simple prioritization mechanisms would be similar to Oldest first - prioritizing the packet that has been in the network longest, or Closest First - preferring the packets that are closest to their destination. These simple modifications to packet routing can have major effects on network performance. Different types of packets may be treated differently, or certain applications can be prioritized over

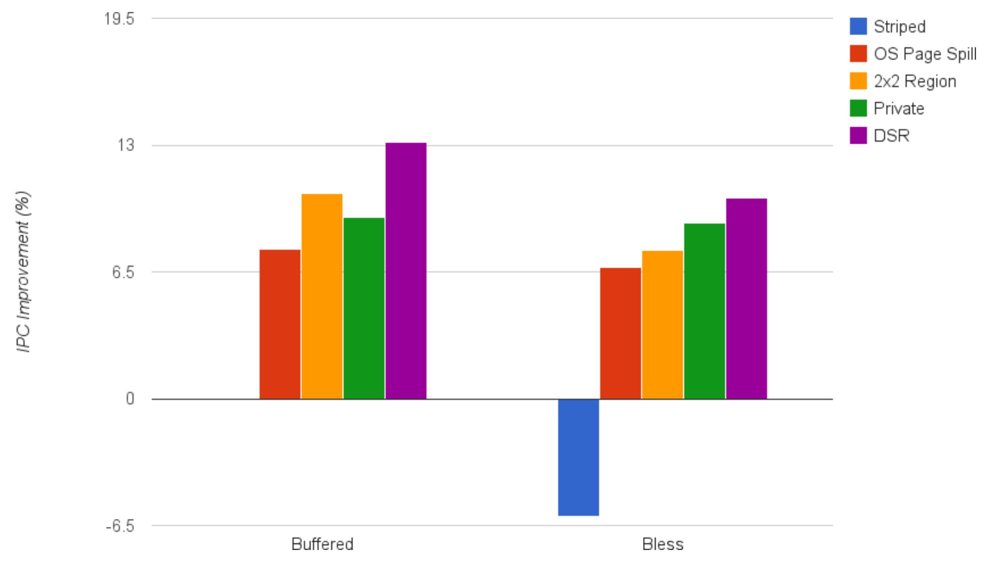

Figure 11: Mapping Performance (Normalized to Buffered Striped)

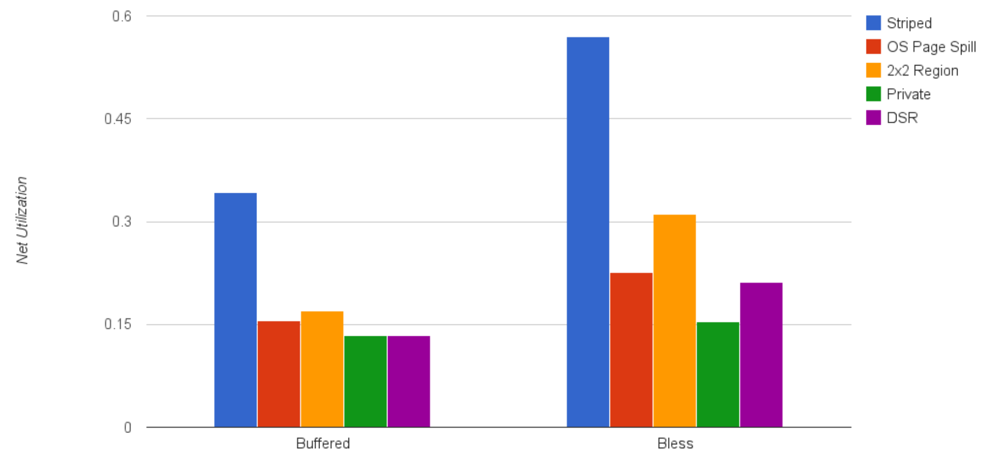

Figure 12: Mapping Network Utilization

others.

5.1 Bufferless Arbitration

Bufferless deflection networks suffer from a potential failure known as Livelock. While buffered routers must be designed to avoid deadlock through circular backpressure dependencies, bufferless routers don't exert backpressure, and instead allow packets to greedily move about the network where space is available. The problem lies in the fact that packets may be deflected, and move in a direction that is counter productive. In a heavily contended bufferless network, these deflections can lead to high variance in latency. Whereas round robin coupled with backpressure guarantees that each packet will eventually get the opportunity to move, bufferless routing must provide other ways to make this guarantee.

In BLESS, the authors experiment with different prioritization mechanisms such as Oldest First, Closest First, Furthest First, and others. One of the reasons that the authors finally decided upon Oldest First was that it guaranteed a total priority ordering of all packets in the network. On the entire network scale, the highest priority packet would be naturally directed quickly to its desired point of ejection. At the same time, each packet would eventually become the oldest, as each packet older would be prioritized, and leave the network. The authors used this inductive proof to guarantee that Livelock was impossible in the Bless network.

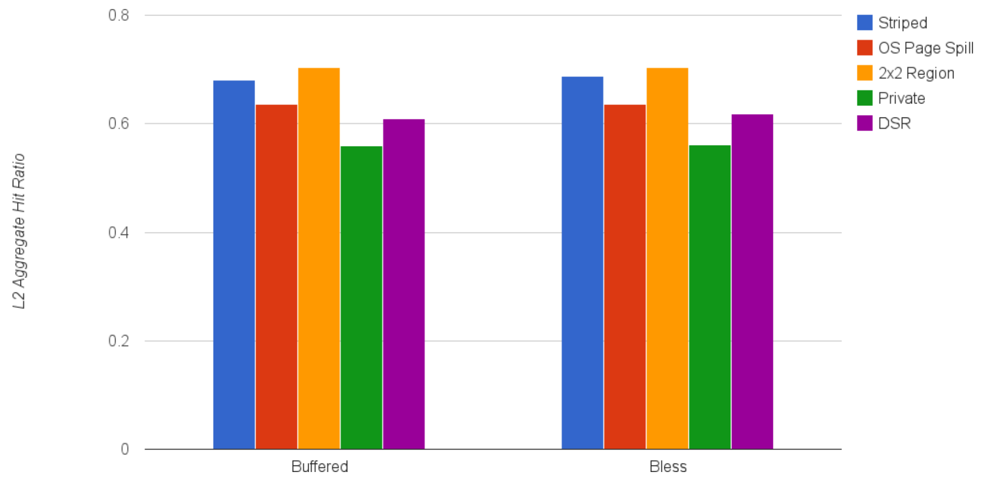

Figure 13: Mapping Aggregate L2 Hit Ratio

The other arbitration/prioritization mechanisms tried suffered from adding too much variance to arrival times, and ended up not offering performance improvements. CHIPPER attempted to go a different route, sacrificing the costly sorting and arbitration logic to provide minimalist Livelock freedom with Golden Packet. Performance suffered further, however, as the latency distribution narrowing effects of age-based arbitration were gone.

5.2 Prioritization for Performance

Recently, mechanisms have been proposed for network prioritization with the goal of improving system performance[5, 6]. In network port arbitration, different packets can be treated differently to try and increase the overall throughput of the system. Memory (or network) level parallelism inherent in modern systems means delays for different requests may affect execution time differently.

The delay of some request by a cycle may not stall it's application for a cycle, as it may not be on the critical path. Similarly, prioritizing a packet to make it arrive faster will only benefit the system if it was going to stall the processor. It is simple to know in a simulator which packet is stalling the processor, but in the real world CMPs are naturally distributed systems. Knowing the optimal packet prioritization is difficult because it depends strongly on workload and benchmark behaviors, as well as cache, memory, and network state. As a general rule though, known latency sensitive packets that are likely to directly affect the execution time of an application should be prioritized.

5.3 Application Awareness

In [5], the authors propose Stall Time Criticality (STC), where individual applications are prioritized based on the utility they get from the network. Packets that are most likely to stall their core if delayed are prioritized foremost. This has the affect of deprioritizing intensive, memory-parallel applications. Those applications that most frequently stall on a given memory request are given free reign on the network bandwidth. Because they, in effect, no longer contend with intense applications, their packets move through the network faster, stalling less frequently.

In their evaluation, the authors of [5] evaluate one network configuration. It has buffered, virtual channel routers as in our baseline, as well as a striped L2 cache.

We built a version of the STC prioritization mechanism in our simulator that works with Bufferless as well as Buffered routers. This requires swapping out the Oldest First ranking logic for STC based ranking, but does not require anything beyond the Buffered implementation to work in a Bless router. We use 8 application ranks, with 8 injection buffers as the original evaluation did.

5.4 Prioritization Comparison

In Figure 14, we present the performance results of our evaluation of STC in both Buffered and Bless routers, with the baseline Striped mapping. We do this to validate our implementation against that of the original STC proposal. We observe, however, that our system sees less benefit from the STC ranking mechanism than the results reported in [5], only 2.3% in Buffered and 4.4% in Bless. The performance aggregate values are shown in Figure 15.

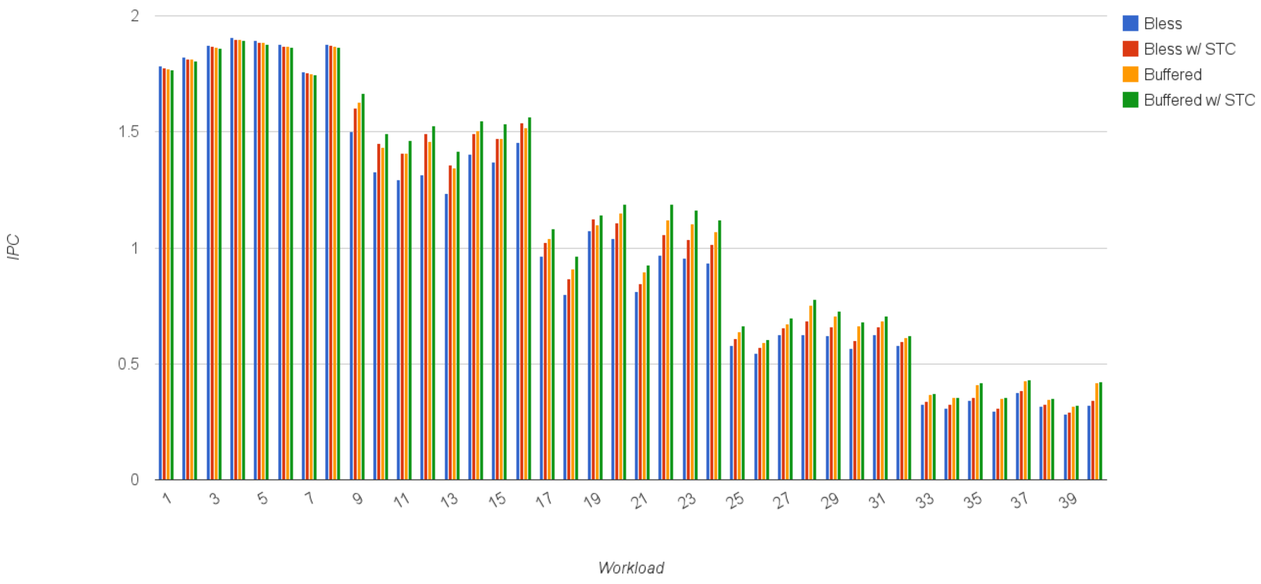

Figure 14: Prioritization in Buffered and Bufferless Networks

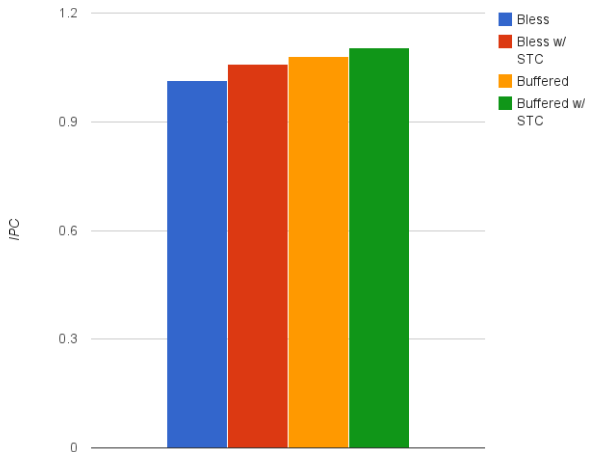

Figure 15: Overall Performance of Prioritization

In the 50% network intense workloads, these numbers increase to 6.9% and 4.7%, but are still lower than expected. We believe that this is largely a factor of workload selection and system and memory configuration. It can be seen in Figure 16 that STC does decrease the network intensity of workloads on average, which is part of what gives it the increased performance. This comes from prioritizing the applications with infrequent, critical requests, effectively throttling back the more intensive traffic composed of less-critical packets.

6 Combined Evaluation with Mapping and Prioritization

In Figure 17, we present performance results of the system with network prioritization, comparing the effects of router design and mapping. Each of the systems shown uses the STC implementation discussed in the last section, with different mappings. The results are given as percent improvement over the baseline system - buffered Routing with naive L2 cache Striping.

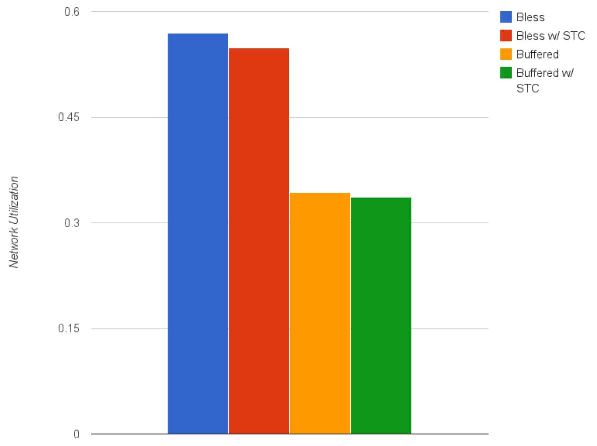

Figure 16: Effect of Prioritization on Network Utilization

The DSR mapping produced the best results again, now alongside STC prioritization. The two mechanisms complimented each other poorly in the buffered network, where the 13.1% gain of DSR is complemented only slightly by STC to a 13.3% improvement over baseline. The bufferless network similarly shows negligible improvement by the combination of the two methods. The DSR mapping alone gives the bufferless network a 10.3% increase over the buffered, Striped baseline. With STC as well, it only rises to 10.4%. This does not strongly motivate the use of STC for this system, since it comes with additional hardware costs such as ranking logic and injection port separation. The reduction of network traffic and congestion from the new mapping mechanism gives STC few opportunities to push critical packets ahead of others, and thus doesn't allow it to contribute greatly to performance. The rest of our results are from evaluations without STC, as we found that the negligible benefits did not overcome the additional complexity required.

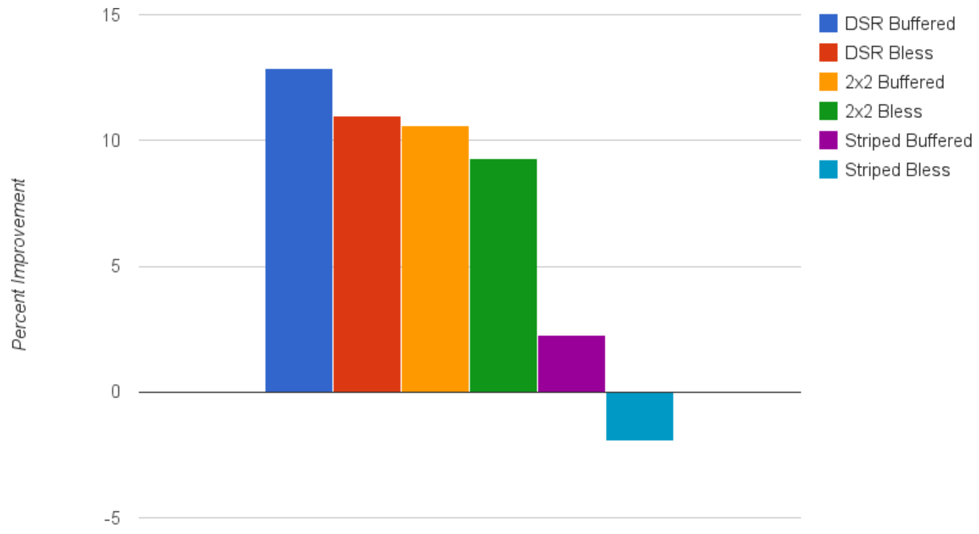

Figure 17: Performance of Mapping Mechanisms on Top of STC Prioritization

6.1 Power and Energy Efficiency

To compare the alternate designs on the merits of energy efficiency, we use the power and area modeling infrastructure from [7]. This infrastructure uses ORION[30] to provide data for the baseline buffered system, and custom RTL (Verilog) models of the Bless router. We use event counters in our simulator for traversals, buffer loads and stores, and similar relevant network events. We use values from these counters along with estimations of per-event energy consumption from our models to generate the final power numbers.

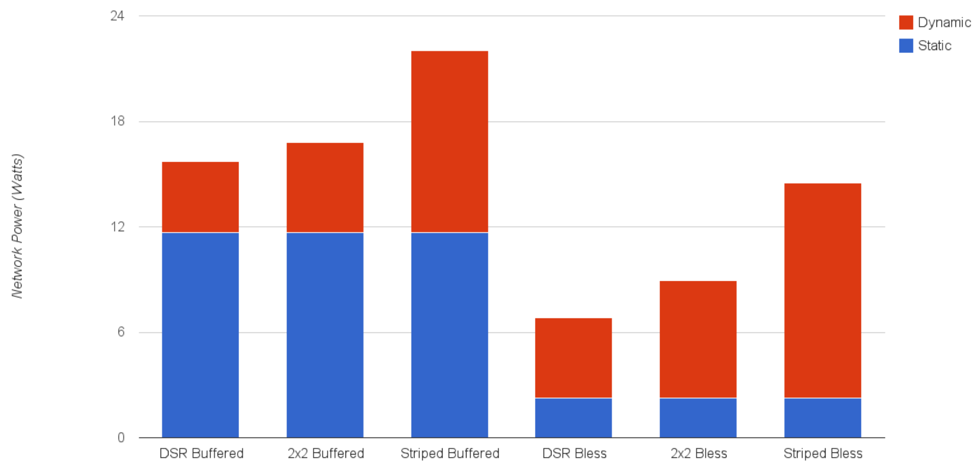

Figure 18: System Power Consumption

Figure 18 shows the power consumption of the six final configurations. When confined to the baseline Striped mapping, we see a 34.2% decrease in network power from the Bless router. This is fairly close to the 40% power reduction presented in [22]. When DSR mapping is used with both routers, that power gap increases to a 56.6% decrease in power consumption from the bufferless system.

This is because DSR reduces the network load by requires fewer communications, and fewer hops per transaction on average. While this significantly reduces the dynamic power consumed in both the buffered and Bless cases, the Static power is unaffected. Static power consumption is reduced by 80% in the Bless design through removal of buffers, and thus defines the minimum power consumption to be much lower. We conclude that Bufferless routing becomes significantly more effective in reducing network power when locality-aware data mapping mechanisms are used.

7 Conclusion

In this work, we have presented an evaluation of L2 Cache mapping mechanisms, NoC prioritization, and the tradeoffs involved between buffered and bufferless routing. We described an extension to the Dynamic Spill Receive mechanism [27] for mesh networks on chip, and found that this simple extension allowed for 13.1% improvements to performance in buffered networks and 10.3% in bufferless, compared to a baseline buffered network with naive striped data mapping of cache blocks to cache slices. We also showed that our adaptation of DSR enables the bufferless router to reduce network power consumption by 56.6% compared to a buffered router, greater than what can be achieved with the naive striped mapping. Evaluating prioritization alongside mapping showed that the decreased network intensity left little opportunity for the evaluated prioritization mechanism to improve performance.

Our key conclusions are as follows:

1. With intelligent, locality aware mapping of data to on-chip cache slices, bufferless network performance can get very close to buffered network performance. This is because locality aware data mapping reduces network utilization, making lower-thoughput networks potentially more effective, and reduces the negative effects deflections can have on performance in bufferless networks.

2. Intelligent, locality aware data mapping also significantly increases the network power advantage of bufferless routers over buffered ones, for the same reasons as above.

3. Intelligent packet prioritization mechanisms can improve the performance of both buffered and bufferless networks, but their benefit is not as pronounced when intelligent locality-aware data mapping is employed because contention reduces with such mapping.

8 Acknowledgements

We gratefully acknowledge members of the SAFARI research group, in particular Chris Fallin, Rachata Ausavarungnirun, and Kevin Chang, for many insightful discussions on this and related work. We also acknowledge the financial and equipment support of Gigascale Systems Research Center, NSF, AMD, Intel, and CMU CyLab.

References

[1] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, New York, NY, USA, 2007. ACM.

[2] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. *International Symposium on Computer Architecture*, 0:264–276, 2006.

[3] S. Cho and L. Jin. Managing distributed, shared l2 caches through os-level page allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 455–468, Washington, DC, USA, 2006. IEEE Computer Society.

[4] W. J. Dally. Virtual-channel flow control. In *Proceedings of the 17th annual International Symposium on Computer Architecture*, ISCA '90, pages 60–68, New York, NY, USA, 1990. ACM.

[5] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Application-aware prioritization mechanisms for on-chip networks. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, New York, NY, USA, 2009. ACM.

[6] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Aergia: exploiting packet latency slack in on-chip networks. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, New York, NY, USA, 2010. ACM.

[7] C. Fallin, C. Craik, and O. Mutlu. CHIPPER: A low-complexity bufferless deflection router. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, February 2011.

[8] C. Gomez, M. Gomez, P. Lopez, and J. Duato. Reducing packet dropping in a bufferless noc. In E. Luque, T. Margalef, and D. Benitez, editors, *Euro-Par 2008 Parallel Processing*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008.

[9] P. Gratz, C. Kim, R. Mcdonald, S. W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. In *in Proc. Int. Conf. Comput. Des*, 2006.

[10] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive nuca: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 184–195, New York, NY, USA, 2009. ACM.

[11] M. Hayenga, N. E. Jerger, and M. Lipasti. Scarab: a single cycle adaptive routing and bufferless network. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, New York, NY, USA, 2009. ACM.

[12] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27, September 2007.

[13] Intel Corporation. The single-chip cloud computer. 2010.

[14] S. A. R. Jafri, Y.-J. Hong, M. Thottethodi, and T. N. Vijaykumar. Adaptive flow control for robust performance and energy. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, Washington, DC, USA, 2010. IEEE Computer Society.

[15] J. Kim. Low-cost router microarchitecture for on-chip networks. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 255–266, New York, NY, USA, 2009. ACM.

[16] A. K. Kodi, A. Sarathy, and A. Louri. ideal: Inter-router dual-function energy and area-efficient links for network-on-chip (noc) architectures. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 241–250, Washington, DC, USA, 2008. IEEE Computer Society.

[17] W. K.Zuravleff and T. Robinson. Controller for a synchronous dram that maximizes throughput by allowing memory requests and commands to be issued out of order, May 1997.

[18] J. Laudon and D. Lenoski. The SGI Origin: a ccNUMA Highly Scalable Server. In *Proceedings of the 24th annual International Symposium on Computer Architecture*, ISCA, New York, NY, USA, 1997. ACM.

[19] H. Lee, S. Cho, and B. Childers. Cloudcache: Expanding and shrinking private caches. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011.

[20] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis. Evaluating bufferless flow control for on-chip networks. volume 0, pages 9–16, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[21] Micron. *DDR3 SDRAM MT41J256M Data Sheet*. [Online]. Available: http://www.micron.com/get-document/?documentId=425&file=1Gb_DDR3_SDRAM.pdf.

[22] T. Moscibroda and O. Mutlu. A Case for Bufferless Routing in On-Chip Networks. In *Proceedings of the 36th annual International Symposium on Computer Architecture*, ISCA-36, New York, NY, USA, June 2009. ACM.

[23] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, ASPLOS-VII, pages 2–11, New York, NY, USA, 1996. ACM.

[24] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27, September 2007.

[25] J. D. Owens, P. Mattson, U. J. Kapasi, W. J. Dally, and S. Rixner. Memory access scheduling. volume 0, page 128, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[26] H. Patil, R. Cohn, M. Charney, R. Kapoor, and A. Sun. Pinpointing representative portions of large intel itanium programs with dynamic instrumentation. In *In International Symposium on Microarchitecture*, pages 81–92. IEEE Computer Society, 2004.

[27] M. Qureshi. Adaptive spill-receive for robust high-performance caching in cmps. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, feb. 2009.

[28] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th annual International Symposium on Computer Architecture*, ISCA '07, New York, NY, USA, 2007. ACM.

[29] Tilera Corporation. *Tile Processor Architecture Overview*, 2009.

[30] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. *Microarchitecture, IEEE/ACM International Symposium on*, 0:294, 2002.