History-Based Arbitration for Fairness in Processor-Interconnect of NUMA Servers

Wonjun Song[†], Gwangsun Kim[††], Hyungjoon Jung[†], Jongwook Chung[‡], Jung Ho Ahn[‡], Jae W. Lee[‡], John Kim[†]

[†]KAIST, [††]ARM, [‡]Seoul National University

[†]{iamwonjunsong, hans7taiji, jjk12}@kaist.ac.kr, [††]gwangsun.kim@arm.com, [‡]{jwc815, gajh, jaewlee}@snu.ac.kr

Abstract

NUMA (non-uniform memory access) servers are commonly used in high-performance computing and datacenters. Within each server, a processor-interconnect (e.g., Intel QPI, AMD HyperTransport) is used to communicate between the different sockets or nodes. In this work, we explore the impact of the processor-interconnect on overall performance – in particular, the performance unfairness caused by processor-interconnect arbitration. It is well known that locally-fair arbitration does not guarantee globally-fair bandwidth sharing as closer nodes receive more bandwidth in a multi-hop network. However, this work demonstrates that the opposite can occur in a commodity NUMA server where remote nodes receive higher bandwidth (and perform better). We analyze this problem and identify that this occurs because of *external* concentration used in router micro-architectures for processor-interconnects without globally-aware arbitration. While accessing remote memory can occur in any NUMA system, performance unfairness (or performance variation) is more critical in cloud computing and virtual machines with shared resources. We demonstrate how this unfairness creates significant performance variation when a workload is executed on the Xen virtualization platform. We then provide analysis using synthetic workloads to better understand the source of unfairness and eliminate the impact of other shared resources, including the shared last-level cache and main memory. To provide fairness, we propose a novel, *history-based arbitration* that tracks the history of arbitration grants made in the previous history window. A weighted arbitration is done based on the history to provide global fairness. Through simulations, we show our proposed history-based arbitration can provide global fairness and minimize the processor-interconnect performance unfairness at low cost.

ASPLOS '17, April 08-12, 2017, Xi'an, China
© 2017 ACM. ISBN 978-1-4503-4465-4/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3037697.3037753

CCS Concepts **• Computer systems organization** → *Interconnection architectures*

Keywords Processor-interconnect; NUMA servers; arbitration; router concentration

1. Introduction

Interconnection networks can have a large impact on overall system performance and cost. There has been a significant amount of research done on large-scale networks, including networks found in supercomputers and high-performance computing systems. Recently over the past 10 to 15 years, research on interconnection network has extended to on-chip networks [7]. In this work, we focus on a different type of interconnection networks that is found in NUMA servers to interconnect the sockets or nodes [1] within a server system – which we refer to as *processor-interconnect*. AMD HyperTransport [29] and Intel QPI (QuickPath Interconnect) [24] are two examples of processor-interconnect that are commonly used. In particular, we focus on arbitration and global fairness within a processor-interconnect.

A high-level block diagram of a processor-interconnect found in servers today is shown in Figure 1 for a 4-socket server. Each socket consists of multiple memory modules, memory controllers, and a router, often located within the on-chip Northbridge that connects to other sockets. Most of the server systems used today are 2- or 4-socket servers while some 8-socket servers are available. The size of the processor-interconnect system is relatively small compared with large-scale networks with thousand of nodes or an on-chip network with tens to hundreds of components. However, we show how the performance of the processor-interconnect can have a significant impact on overall system performance. In particular, the arbitration and the router

[1] We use the term or nodes interchangeably in this work. For some systems with multi-chip packaging, a single socket can consider of two "nodes".

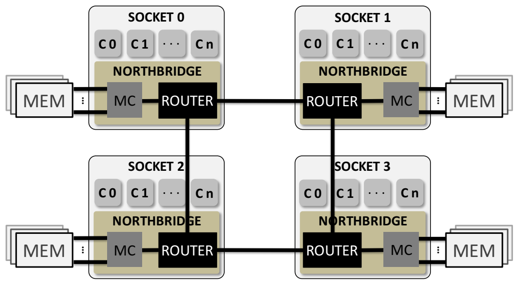


Figure 1. Block diagram of a 4-socket server system

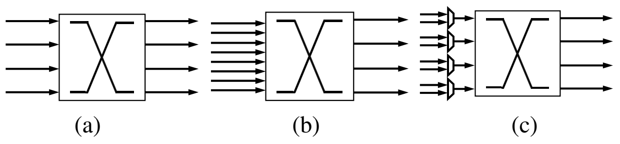


Figure 2. Block diagram of (a) conventional 4-port router, (b) integrated 2-way concentration, and (c) external 2-way concentration router.

concentration microarchitecture can impact overall global fairness.

Fairness has been widely studied in interconnection networks, and a common problem has been global fairness – with only local fairness in arbitration, remote nodes can be at a disadvantage. However, in this work, we show that the opposite can occur in a processor-interconnect because of the way the router micro-architecture and concentration is implemented. Concentration is a common technique used to share network resources [8] (such as channel bandwidth) and the first stage of a processor-interconnect often leverages external concentration (or a multiplexer) where multiple on-chip cores share the off-chip processor-interconnect bandwidth. As a result, we show how the external concentration can result in a remote node obtaining more memory bandwidth than local nodes, causing remote nodes to have *higher* performance than local nodes. We demonstrate how this unfairness can cause significant performance variation for virtual machines (VMs) as workloads allocated to remote nodes perform better than local nodes.

To provide fairness within the processor-interconnect, we propose a novel low-cost, history-based arbitration to achieve global fairness between cores across different sockets. The arbitration maintains a history queue that contains the source ID of the packets that have received arbitration *grants* and have used the bandwidth. By tracking history and performing weighted arbitration, we show how fairness between the sockets can be provided with minimal overhead. We also demonstrate how such interconnect fairness can have a significant impact on the memory system fairness as well. There has been a significant number of studies done on providing fairness in interconnection networks [1, 12, 19, 20], but solutions are not necessarily appropriate, given the different packaging constraints of processor-interconnects

In particular, the contributions of this paper include the following:

- We demonstrate how a remote node can obtain more memory bandwidth, and thus achieve higher performance than a local node in modern multi-socket servers.
- We provide a thorough evaluation to isolate the impact of other system components and evaluate unfairness across different systems. We also provide an analysis of the non-intuitive unfairness behavior of a system using a simple model.
- We propose a novel low-cost, history-based arbitration for processor-interconnects that enables global fairness between the remote and local nodes by keeping track of the history of arbitration grants.

2. Background

In this section, we provide background on concentration in the design of an interconnection network and the traditional parking lot problem. In the following section, we describe how the parking lot problem occurs in the processor-interconnect.

2.1 Concentration in Interconnection Networks

A common technique used in interconnection network is *concentration* [8] where network resources (e.g., network channels) are shared between different terminal or end-point nodes. Concentration has been used in large-scale networks [15] and for on-chip networks [3, 28]. In addition, it is also used in processor-interconnect as multiple cores within a single socket share a single router for communication with remote sockets. In this work, we explore the impact of concentration in such processor-interconnect and the global unfairness that it can cause.

Concentration can be classified as either *integrated* or *external* [18]. A conventional 4-port router is shown in Figure 2(a). Integrated concentration (Figure 2(b)) can provide higher performance as more bandwidth into the router is provided by increasing the router radix or the number of router ports. However, this approach results in increased cost as the router complexity increases. By contrast, external concentration (Figure 2(c)) simplifies the router micro-architecture as the router input bandwidth is shared through an external multiplexer and reduces the network cost but can result in performance degradation since router input bandwidth is shared. In this work, we explore how external concentration can cause global unfairness and result in non-intuitive performance behaviors. Unfortunately, the micro-architectural details of the processor-interconnect router design are not clearly available. Some high level block diagrams of the processor-interconnect interface are available [6] but these do not necessarily correspond to the micro-architectural implementation. However, based on our evaluation, we can reasonably argue that the processor-interconnect implements

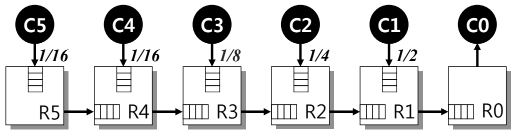


Figure 3. Parking lot problem illustrated on a 1-D mesh network.

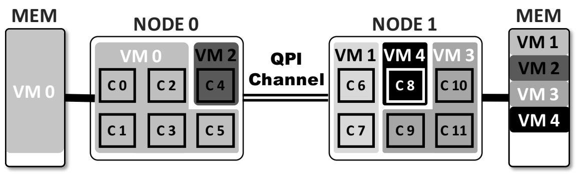


Figure 4. Block diagram of a 2-socket Intel Xeon E5645 system with different vCPU-pCPU bindings.

some form of external concentration and results in global unfairness which we describe in the following section.

2.2 Parking Lot Problem

It is well-known that local fairness does not necessarily guarantee global fairness [8]. A simple example is shown in Figure 3, which is also commonly referred to as the parking lot problem.[2] Assume that all of the terminal nodes want to send traffic to C0. If the arbitration within each router provides local fairness (e.g., round-robin arbitration), the nodes closer to the destination receive a higher fraction of the destination bandwidth and nodes farther away receive exponentially smaller amount of bandwidth – e.g., C1 receives $1/2$ of the bandwidth while C5 only receives $1/16$ of the bandwidth into C0. This problem (and various solutions) has been well studied as local or closer nodes receive higher bandwidth than nodes that are farther away. However, in this work, we show how the opposite can occur when the network has external concentration – i.e., nodes farther away can receive higher bandwidth than local nodes because of the locally fair arbitration and external multiplexer implementation of concentration.

3. Global Unfairness

In this section, we first show a case study of performance unfairness with virtual machines (VMs) – in particular, we show how VMs mapped to remote nodes can achieve *better* performance than local nodes. To better understand the behavior, we use synthetic workloads to show how remote nodes have access to higher bandwidth, compared with local nodes, to local memory because of *external* concentration in the router as the local cores cause a local bottleneck and remote core traffic bypasses the bottleneck to achieve higher performance. In an attempt to isolate the impact of other system components, we provide a sensitivity study to understanding the effect of the processor-interconnect unfairness.

[2] The analogy is that if there is only one exit in the parking lot (e.g., C0) and cars waiting to exit in C1 – C5, the cars closer to the exit will be serviced at a faster rate than the cars that are farther away.

Description	Value
System	Xeon E5645
Codename	Westmere-EP (32nm)
# of Sockets (or nodes)	2
Core Freq.	2.40 GHz
# of Cores	6 per node
L2 Cache	1.5 MB per node (private)
L3 Cache	12 MB per node (shared)
Interconnect	QPI, 5.86 GT/s
# of MC channel	3
Memory B/W	32 GB/s per socket

Table 1. Description of the Intel Xeon E5645 system used in our evaluation.

3.1 Performance Unpredictability with Virtual Machines

Unpredictable performance due to shared resources is a critical concern with virtualization and cloud computing. To demonstrate the performance unpredictability, we evaluated a system with multiple virtual machines (VMs) with a Xen hypervisor [4] on a dual-socket Intel Xeon E5645 server (Table 1). We assumed that 5 different VMs are using this shared system, all with different numbers of CPU requirements, and one particular VM (i.e., VM0) has a larger amount of memory requested than other VMs. The memory binding is determined by the Xen hypervisor at VM creation time while the virtual CPU to physical CPU (vCPU-pCPU) mapping can change as migration occurs depending on the Xen scheduler. In general, the scheduler prefers to have the CPU mapped such that local memory is accessed. In our evaluation, since VM0 has requested a lot of memory, the other VMs memory will likely be mapped to Node 1 and the remaining CPUs will be mapped by the scheduler.

An example of VM mapping is shown in Figure 4 where VM2 is mapped to C4 in Node 0 while it uses memory allocated in Node 1. In this example, we assumed that VM2 and VM4 only consist of 1 vCPU while VM1 has 2 vCPUs and VM3 has 3 vCPUs. During our evaluation, the Xen scheduler could migrate vCPUs across the different pCPUs; thus, VM4 can be mapped to a Node 0 core or one of the vCPUs from VM1 or VM3 can also be mapped to Node 0 as well. In our evaluation, we evaluate the performance of a memory-intensive SPEC benchmark (i.e., `libquantum`) on the different VMs (VM1 to VM4) and assumed that only a guest OS is running on VM0.[3]

The evaluation results are shown in Figure 5 where the normalized execution times are shown for VM1, VM2,

[3] We also evaluated with VM0 executing compute-intensive workloads and similar performance variation was also observed.

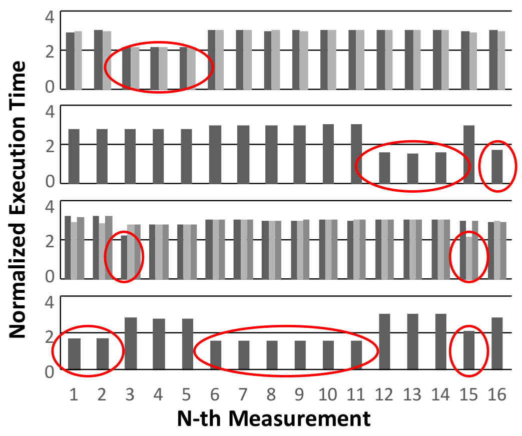


Figure 5. Performance variation for each VM for `libquantum`. The highlighted measurements represent VMs which had its vCPU mapped to a CPU on Node0. The execution time is normalized to `libquantum` that is executing alone on Node1.

VM3, and VM4 for 16 different measurements. For VM1 and VM3 which have multiple vCPUs, each measurement had multiple results for each vCPU, and the various bars in Figure 5 for VM1 and VM3 represent the performance of the different vCPUs. The results show that there can be significant performance variation. For example, for VM2 and VM4, the performance difference can be up to 2×. The measurement data that are highlighted in Figure 5 represents VMs that were predominantly executed on Node0 (or the remote node) and the results clearly show that when the workload is executed on the remote node, the execution time is significantly reduced. For measurements 3, 4, and 5, the performance improvement is distributed across VM1 and VM3 as the migration of the vCPUs of these two VMs between Node1 and Node0 result in this behavior.

3.2 Inverse Global Unfairness

To better understand performance unfairness, we evaluated the impact of remote node memory accesses by executing the same workload on all the cores in Node 0 (or the *local* node) and Core 6 (C6) in Node 1 (or the *remote* node) (Figure 4). The evaluation was done without Xen to avoid any effect from the hypervisor and guest OSs. The NUMA memory mapping was modified such that all cores use Node 0 memory. We compared the performance using both a synthetic workload and `libquantum` used earlier. For the synthetic workload, we use a simple, stream memory access workload (e.g., a[i] = b[j]) and only one element of each cacheline is accessed to create a memory-intensive workload. This access pattern also results in over 99% miss rate for the LLC (last level cache) and minimizes any possible effect from the shared LLC.

The only difference between the workloads running on the local and the remote node is that for the remote node, the

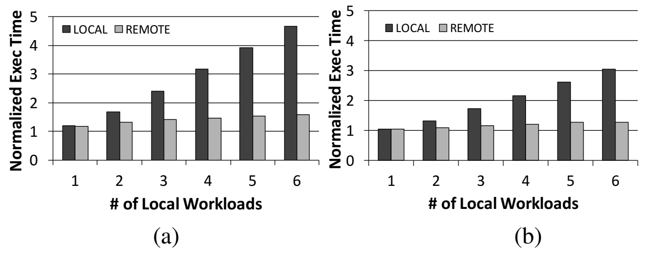


Figure 6. Evaluation comparing remote and local workloads with (a) synthetic and (b) `libquantum`. Results are normalized to remote (local) workloads executing alone.

processor-interconnect (e.g., QPI interconnect) is traversed before the remote memory is accessed, and this adds latency. The results shown in Figure 6 are normalized to the standalone results where the local (and the remote) workloads are executed alone in the system. Although not shown, executing a workload on a remote node results in an approximately 45% increase in runtime for the synthetic workload and a 32% increase for `libquantum`. The idle memory latency for remote memory access is approximately 40% higher than local memory access. The difference in execution time for the remote and local `libquantum` is smaller than synthetic as the remote memory access time can be overlapped with computation.

The results in Figure 6 show the performance of both the remote and the local workload as the number of workloads executed on the local node (e.g., Node 0 in Figure 4) is increased from 1 to 6. With six cores per socket, we evaluated up to 6 workloads such that each workload could be executed exclusively on a single core without sharing core resources. The expectation was that both the local and the remote workloads would show degraded performance as the additional workloads increase contention for shared resources, including the on-chip bandwidth and the memory bandwidth. For both the synthetic and `libquantum` workloads, the local node workload performance continue to degrade as the number of workloads executed on the local node increases – i.e., by 3× for `libquantum` (and by 4.7× for the synthetic workload) when all the local cores are executing the same memory intensive workload. However, the remote node performance is *not* significantly degraded as the number of local workloads increases. When all the local cores are executing the same workload, the synthetic remote workload is degraded by only 58% (compared with 4.7× degradation of the local workload); memory access latency for local nodes increased from the local bottleneck and actually *exceeds* the memory latency of the remote node. For `libquantum`, the degradation for the remote workload is only 28% (compared with 3× for the local workload). As a result, when all the local cores are executing the same memory-intensive workload, the performance of the remote node actually *exceeds* the performance of the local node by 79% for `libquantum` (and by approximately 2× for the synthetic workload). *Sim-*

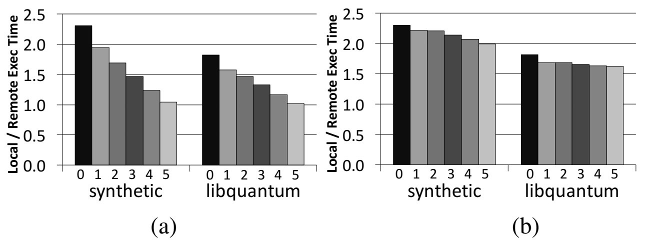


Figure 7. Performance impact of unfairness when additional workloads are added to the remote node with (a) memory intensive workloads and (b) compute intensive workloads.

ilar performance unfairness was observed on other multi-socket systems that we evaluated, including a 4-node Intel Nehalem (6×4 cores), Intel 2-node SandyBridge (8×2 cores), and a 4-node (8-socket) AMD Opteron 6320. As this result is non-intuitive, we provide an analysis of why this occurs in Section 4 with a network-only synthetic workload evaluation.

3.3 Sensitivity Study to Isolate the Source of Unfairness

The results presented in the previous section were measured from a real system but since the details of the internal micro-architectures are not publicly known, it can be difficult to isolate the impact of one component over another component in the system. However, we made various efforts to isolate the impact of the processor-interconnect over other system components. In the synthetic workload, there was over 99% miss rate for the LLC; thus, the impact of the shared LLC is minimal. We also evaluated the unfairness on an AMD Opteron 6282 system that supports LLC cache partitioning to minimize the impact of the shared LLC, and similar trends were observed. Evaluation using uncacheable to reduce the impact of LLC also showed similar unfairness trends. We also disabled the hardware prefetcher and disabled the DVFS to minimize system variability. In addition, we performed a sensitivity evaluation where we added additional workloads on the remote node (Node 1 in Figure 4) which access the memory directly connected to Node 1. The purpose was to show the impact of remote workloads on the performance unfairness described earlier. To minimize the impact of the main memory system, the remote workloads access the memory attached to Node 1; thus, only C6 among the cores in Node 1 accesses the local memory connected to Node 0.

Figure 7 plots the performance (i.e., execution time) ratio of the local and remote workloads, as the number of workloads on the remote node (Node 1) is varied from 0 to 5. The results in Figure 7(a) show memory-intensive workloads on the remote node (i.e., for `libquantum`, we add additional `libquantum` workloads while for synthetic, additional synthetic workloads are added). As more remote workloads are added, the additional local contention within Node 1 degrades the performance of the remote node (C6), and the impact of performance unfairness is reduced. For the synthetic workload, which generates significant amount of memory traffic, the performance ratio approaches approximately 1 – i.e., the remote node and local node workloads have similar performance, and the performance unfairness completely disappears. Similar behavior is also observed for memory-intensive workloads, such as `libquantum` , because the performance unfairness gradually decreases as the number of remote workloads increases.

In Figure 7(b), compute-intensive workloads (`hmmer`) are added to a remote node (Node 1), and we compare the same performance ratio. Since compute-intensive workloads access the memory system less frequently, the performance unfairness is only slightly reduced. Thus, as additional workloads are added to the remote nodes, the performance unfairness decreases and is almost removed if memory-intensive workloads are executed on the remote nodes; however, if compute-intensive workloads are allocated to the remote nodes, performance unfairness still remains because there is limited contention within remote Node 1.

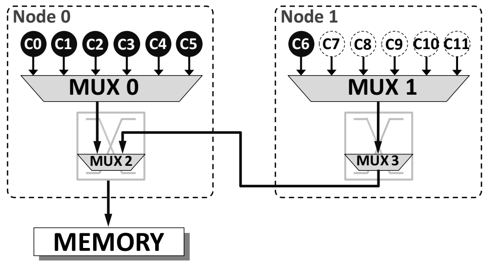
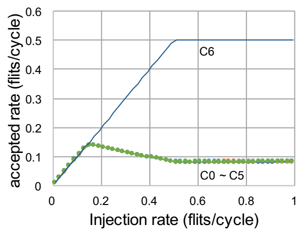


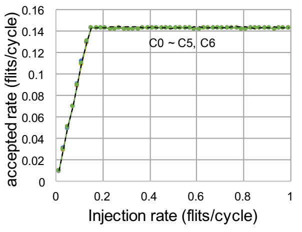
Figure 8. Processor-interconnect model of the 2-socket server system.

4. Analysis of Global Unfairness

To understand the behavior of global unfairness described in the previous section, we model the processor-interconnect using a network-only simulator and evaluate the performance. Figure 8 shows a simplified block diagram of the processor-interconnect for the Intel 2-socket server evaluated in the previous section. Note that this simplified view might not exactly represent the microarchitecture as the details are not publicly available. For example, the external multiplexers might be implemented through a shared queue structure. However, some form of “mux” is required to share the processor-interconnect bandwidth we model, to avoid increasing the port count on the router switch. Since the system has 6 cores, a local 6-to-1 arbitration occurs through the local muxes (MUX0 and MUX1). The output of the mux is connected to the crossbar switch (or the QPI router) within the Northbridge. Within the crossbar switch, another mux is necessary (in this case, represented by MUX2 and MUX3) to access the shared off-chip memory bandwidth. For simplicity, other components, such as last-level caches and memory



(a)



(b)

Figure 9. Synthetic workload evaluation with (a) round-robin and (b) ideal age-based arbitration.

controllers, are not shown in this figure and only the memory connected to Node 0 is shown.

We used a network-only simulator (BookSim 2.0 [14]) to understand the impact of global unfairness in the processor-interconnect. The simulator was modified to model the network configuration shown in Figure 8. We assumed that the input buffers are 16 flit[4] entries deep and evaluated with 3 virtual channels. Since we are focusing on how the bandwidth is being shared by local and remote nodes, we assumed that the latency for each mux (arbitration) is a single cycle and that the wire delay is also a single cycle. In our synthetic evaluation, we assumed that locally-fair round-robin arbitration is used for each mux. Similar to the workload used earlier in Section 3, we modeled a synthetic traffic pattern where cores C0 – C5 and C6 continuously access the memory located in Node 0 (e.g., hotspot traffic to Node-0 memory). We also assumed single-flit packets in our evaluation.

The result of this evaluation is shown in Figure 9(a). With locally fair arbitration, the bandwidth received by Node 0 and 1 are nearly identical as each socket receives approximately 50% of the total bandwidth. However, the bandwidth from the remote node (Node 1) is used only by a single core (C6) while that for the local node (Node 0) is shared by 6 cores (C0 – C5). As a result, the cores on the same local node (C0 – C5) receive only approximately $\frac{(1/2)}{6}$ or 8% of the total bandwidth, whereas the remote node (C6) receives a significantly higher fraction of the bandwidth (or approximately 50%), resulting in approximately 6× difference in bandwidth. Because there is only local fairness, *inverse* global unfairness occurs as the remote node receives higher bandwidth. The remote node goes through 3 stages of arbitration (compared with 2 stages for the local node), but due to the lack of contention for the same destination node, the remote node receives higher bandwidth. Although we assume a simple round-robin (RR) arbitration, the actual system might implement a different arbitration. However, our simplified modeling of the processor-interconnect illustrates the extent to which global unfairness can occur with the remote node obtaining more bandwidth than local nodes with external concentration.

[4] Flit is the unit of flow control in interconnection networks [8].

5. History-Based Arbitration

5.1 Description

In this work, we propose a *history*-based, weighted arbitration that maintains a history of the previous grants from the arbiter. The history of the source node IDs of packets that have received grants is maintained to enable fairness across the different source nodes in the system. As a result, no additional overhead is introduced to the packet since source ID information is commonly found in the packet headers. The history-based approach also enables a distributed global fairness since no explicit communication is needed between sockets, unlike previously proposed NoC QoS mechanisms [12, 19]. We also exploit the fact that processor-interconnects do not have high network diameter (i.e., most multi-socket systems are only one or two hops); thus, the number of nodes in the system is relatively small. As a result, the overhead to maintain the history is relatively small (Section 6).

Within each arbiter, a history-queue is used at each input to keep track of the history of the source ID that previously received arbitration grants. The history queue is essentially a history *window* that provides information on which source nodes have won arbitration and thus, received bandwidth. Based on the history stored in the queue, a weighted arbitration is done where the weight is *inversely proportional to the number of times the same source has been granted in the history queue*. Thus, if the source has been granted more frequently, the weight for that particular source is lower. The weight (w_i) for source i is calculated by counting the number of times the particular source ID exists within the history queue, i.e.,

$$w_i = 1/h_i \tag{1}$$

$$= 1/\sum_{x=0}^{d-1}(hist\_buf[x] == i) \tag{2}$$

where h_i is the count value from the history queue for source i, and d is the history queue depth. Figure 10 is shown for a two-input system with one virtual channel (VC) per port but for a system with multiple virtual channels, a history queue is necessary for each VC. History-arbitration is implemented in a manner similar to a separable allocator [8] with history-arbitration performed among the input VCs and then, performed again among the requests for the same output.

An example of history-based arbitration is shown in Figure 11 for the network configuration shown earlier in Figure 8. In this example, we assume that the history queues are 6 entries deep (i.e., maintain a history of the last 6 packets granted for *each* input). In a steady state, the history for Input 1 is filled with requests from source node $C6$. As a result, the history queue has 6 entries from $C6$ and the weight used in

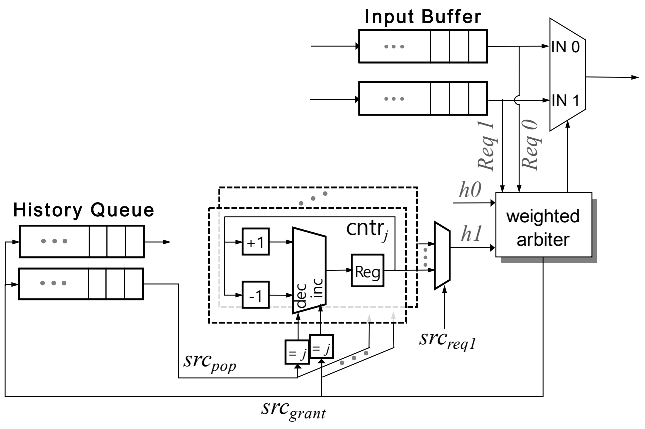


Figure 10. Block diagram of the proposed history-based arbiter with the history queue and the counters. For simplicity, the counter logic for IN0 is not shown.

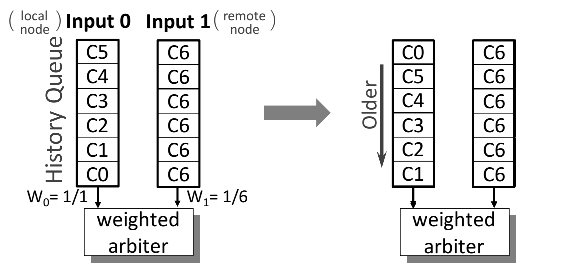


Figure 11. An example of history-based arbitration with 6-entry history queues.

the arbitration is $w_1 = 1/6$. In comparison, the history queue for Input 0 (or local node) is filled with one packet from each of the six different sources C0 – C5 since the first stage was locally-fair, round-robin arbitration. As a result, $w_0 = 1/1$ for the different packets from Input 1 because there is only one entry for each source ID in the history queue. Based on these weights, the local node (Input 0) will be granted 6× more likely than the remote node (Input 1). In the example shown in Figure 11, if we assume Input 0 received the grant, the history queue would change as shown on the right side of Figure 11, with the newly granted source ID (i.e., C0) being added to the history queue and the head of the queue being removed.

5.2 Microarchitecture

A high-level block diagram of the proposed history-based arbitration is shown in Figure 10. Since searching the history queue for various source IDs and counting them can be costly, a separate per-source counter is used to keep track of the history and increment/decrement accordingly based on the arbitration winner (e.g., src_{grant}) and the head of the history queue (e.g., src_{pop}). The history queue[5] is a FIFO queue; if an input receives a grant from arbitration, the source ID information is pushed on to the corresponding history queue, and the oldest entry in the history queue is removed. A high-level description of the history-queue update and the counters is described in Algorithm 1.

Algorithm 1: Updating history queue and counters within the arbiter.

```
foreach history_queue at port i do
    if $grant_i$ then
        if history_queue is full then
            $src_{pop}$ = pop ();
        push $src_{grant}$;
        // update counter
        foreach counter j do
            if ($src_{grant}$ == j) && ($src_{pop}$ != j) then
                cntr$_j$++;
            else if ($src_{grant}$ != j) && ($src_{pop}$ == j) then
                cntr$_j$–;
```

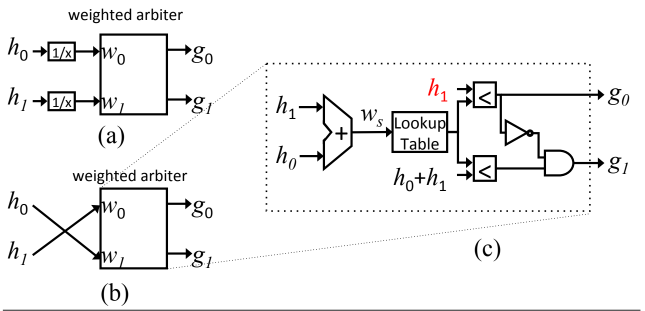


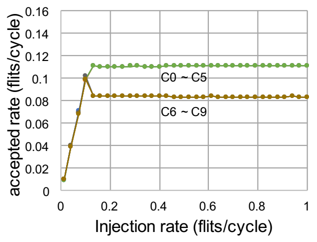
Figure 12. Block diagram of (a) baseline history-based weighted arbiter, (b) simplified weighted arbiter with flipped inputs, and (c) weighted arbiter microarchitecture with a look-up table.

The weighted arbiter can be implemented with a weighted RR arbiter [8] that would require determining a "quota" for each input. Instead, we leverage probabilistic arbitration [20]; a history-based arbiter would further complicate the microarchitecture since the inverse of the counter values needs to be calculated (Eq (1)) before the weighted arbitration (Figure 12(a)). However, since the weights for history-based arbitration are inverse of the counter values, we observe that the *input weights are proportional to the counter values of the other port* and swap the counter values as inputs to the weighted arbiter (Figure 12(b)).[6]

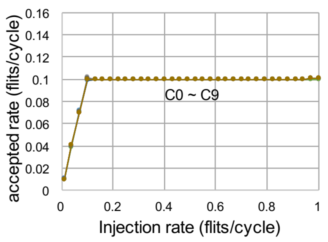
In probabilistic arbitration, a random number is generated between 0 and w_s, or the sum of the weights, and based on the random number value, the appropriate grant is asserted. The random number generation can be on the critical path but for history-based arbitration, the maximum value of w_s is fixed at $2d$, where d is the number of history queue entries. Thus, random numbers between 0 and $2d$ can be pre-

[5] Even though separate counters exist, the history queue is still necessary to keep track of the oldest source ID that was granted and decrement the appropriate counter.

[6] The weighted arbiter complexity would increase if there are more than 2 inputs. However, in a low-diameter processor-interconnect, history-based arbitration is only needed between local and remote nodes; thus, a 2-input weighted arbiter is sufficient for a processor-interconnect.



(a)



(b)

Figure 13. Performance with (a) limited and (b) larger history queue for local:remote = 4:6 traffic ratio.

generated and stored in a look-up table – with each entry i in the table containing a random number between 0 and i. When w_t is calculated, the look-up table is indexed with w_t to determine the random number and perform the weighted arbitration (Figure 12(c)).

5.3 History Queue Size

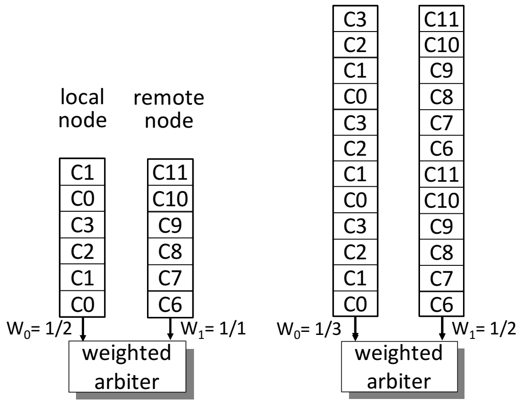
With the history-based arbitration, the global unfairness described in the previous section was removed and all of the nodes received similar bandwidth, nearly identical to the ideal age-based arbitration results shown earlier in Figure 9(b). With a history queue of 6 entries, global fairness was provided across the 7 sources (6 sources from one port and 1 source from another port). However, if the traffic originated from a different number of sources, having only 6 entries does not necessarily provide strict global fairness. For example, assume that 4 local nodes (C0–C3) and 6 remote nodes (C6–C11) are sending traffic to the same local memory. Global fairness is not maintained because of the *bias* created by the limited history window size. The depth of the history queue determines the amount of history that is used in the arbitration, and a history queue with only 6 does not provide an accurate representation of arbitration history.

For this traffic pattern with 4 local nodes and 6 remote nodes accessing the same local memory, the synthetic traffic pattern results are shown in Figure 13(a). With a history queue depth of 6, the remote nodes receive slightly higher bandwidth than the local nodes. Ideally, since the ratio of local to remote nodes requesting bandwidth is 4:6, the weighted arbitration should be based on these weights. However, with a history queue of only 6 entries, the limited history queue creates bias and "appears" as if the local nodes have been serviced more frequently as $w_0 = 1/2$ for C0 and $w_1 = 1$ for C6 (Figure 14(a)), thus, providing higher weight to the remote node.[7] In comparison, if the history queue is doubled to 12 entries (Figure 14(b)), the bias is removed with a larger history window and global fairness across both input is provided as shown in Figure 13(b). In general, to provide fairness between remote and local bandwidth across a range of possible numbers of cores, the history queue depth

[7] The w_0 calculation assumes the request in the arbiter is from $C0$. If the request was from $C2$ or $C3$, the weight would be $w_0 = 1/1$.

Table 2. Simulator configuration used in the evaluation.

	Parameter	Configuration
CPU	Core	Six Out-of-Order cores @ 2GHz Issue width: 4, ROB size: 128
	L1 I/D cache	32KB, 4-way, inclusive
	L2 cache	Private 1MB, 8-way, inclusive
	Cache line size	64B
MEM	scheduler	FR-FCFS, PAR-BS [23]
	DRAM timings	tCK = 1ns
		tRCD = tRP = 14
		tCL = 12
		tRAS = 34



(a) (b)

Figure 14. (a) History-based arbitration with a bias and (b) doubling the history queue to provide fairness.

needs to be the least-common multiple (LCM) of the various possible numbers of sources that can contend. In the example from Figure 14 with 4 local and 6 remote sources, $LCM(4, 6) = 12$ was sufficient.

5.4 Real Workload Evaluation

In addition to a network-only synthetic workload, we evaluate the impact of history-based arbitration using different SPEC workloads using the McSimA+ [2] simulator. The network within McSimA+ was replaced with a cycle-accurate BookSim 2.0 [14] simulator that was used for the network-only evaluation earlier. The configuration for our processor is summarized in Table 2. The network was modeled to mimic the processor-interconnect and we assumed 4-cycle router delay with an extra cycle delay for the external concentration and a single-cycle wire delay. Each VC in the network was assumed to be 16-entries deep, and credit-based flow control was used between the routers.

Figure 15 compares the performance of round-robin arbitration and our proposed history-based arbitration using our simulator. For the different SPEC workloads, we simulated 7 different copies of them with one of them running remotely and 6 copies running on 6 local cores, similar to

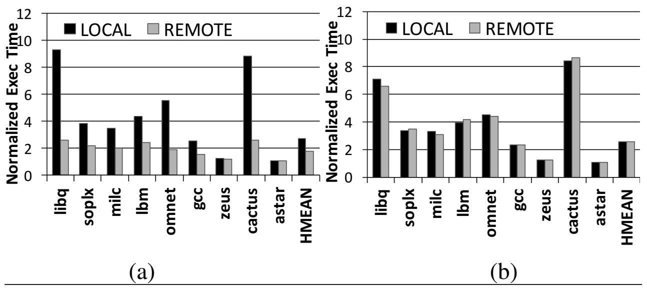


Figure 15. Simulator-based evaluations with (a) round-robin arbitration and (b) history-based arbitration.

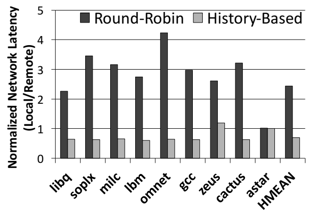


Figure 16. Network latency comparison.

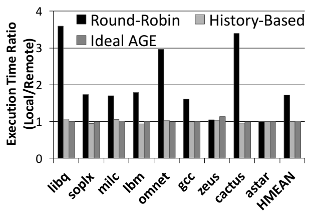


Figure 17. Unfairness comparison.

the setup described earlier in Figure 8. The local and remote results are normalized to the workload running alone on local and remote. With baseline, round-robin arbitration (Figure 15(a)), there is significant performance degradation across most of the workloads with the local experiencing significant slowdown. Some workloads result in over 3× difference between local and remote. For two of the less memory-intensive workloads (`astar, zeusmp`), there is a minimal difference in performance. With history-based arbitration (Figure 15(b)), the difference between the remote and local is significantly reduced as the performance of local and remote are within 2% of each other on average. For memory non-intensive workloads (`astar, zeusmp`), history-based arbitration has minimal impact.

The difference in the processor-interconnect latency is shown in Figure 16 where we plot the ratio of the processor-interconnect latency between local and remote node requests. With round-robin arbitration, the local node experiences significantly higher latency (up to 4.2× higher) compared with the remote nodes because of the additional contention in the local node. However, with history-based arbitration, the difference between local and remote is significantly reduced. For most of the workloads, the ratio plotted has a value under 1 – i.e., the local nodes have *lower* latency than remote nodes. However, the plot only includes the processor-interconnect latency and does not include the additional queuing latency at the interface (core processor-interconnect interface, processor-interconnect memory interface). To further understand performance unfairness, we plot the ratio between local and remote slowdown in Figure 17. Compared to an ideal age-based arbitration, our history-based arbitration is able to achieve fairness that is within 2%. Because of space constraints, the results for synthetic workload are not shown but they follow similar trend as the memory-intensive SPEC workloads.

5.5 Cost & Scalability

The main source of cost are the queue structures needed. The width of the history-queues is $log_2(N)$ bits, where N is the total number of cores. For example, in the Intel Xeon E5645 system with 12 cores (6 cores per socket), 4 bits are sufficient, and $d = LCM(1, 2, 3, 4, 5, 6) = 60$ entries are needed; thus, 60×4 = 240 bits of storage. Considering that the flit size of a QPI interconnect is 80 bits [13], the overhead of the history queue only corresponds to 3 flit entries of the input buffers. Separate counters are maintained for each source, and the widths of the counters are $log_2(history\_depth)$, with N counters for each history queue; thus, approximately 72 bits of storage are required for the counters. The look-up table needs to have $2d$ entries; thus, 128×7 or approximately 900 bits of storage are required, so the cost overhead of the data structure is relatively small. A detailed cost comparison with state-of-the-art NoC QoS is presented later in Section 6.

As the number of cores on a single socket increases, the scalability of the history queue structure impacts cost since the history queue depth needed is $LCM(1, .., n)$, where n is the number of cores per socket. With 10 cores, approximately 1.5kB of storage is needed but with 12 cores, this increases to 16kB and adds significant overhead. However, buffer storage can be significantly reduced at the cost of additional control logic. As described earlier in Section 5.3, $LCM(1, .., n)$ is necessary to avoid any bias in the history window to support fairness across various combinations of remote and local cores. However, for any given combination of remote and local cores, the maximum number of history buffer entries that are actually needed is only $LCM(n, n-1)$. Thus, instead of increasing the buffer entries to support various combination, the queues entries can be effectively *decreased* by only using $LCM(x, y)$ buffer entries where $x, y \leq n$ and x, y are the numbers of cores from remote and local nodes contending for the memory bandwidth. For example, with 12 cores per socket, if the number of remote cores is 11 and all 12 local cores are being used, the entire history queue of 132 entries would be used. However, if the number of cores contending for the memory bandwidth is smaller, (i.e., 5 local and 6 remote) the actual history queue can be reduced to LCM (5,6) = 30 entries. Thus, even with 12 cores, the amount of storage for the history queue is only $LCM(11, 12)$ entries, which requires less than 60 bytes of storage.

The history-based arbitration also enables flexibility in the arbitration. For example, the system can be configured such that local nodes receive more bandwidth than remote nodes by adjusting the weights properly. The results in Figure 18 are shown where the local nodes are provided twice

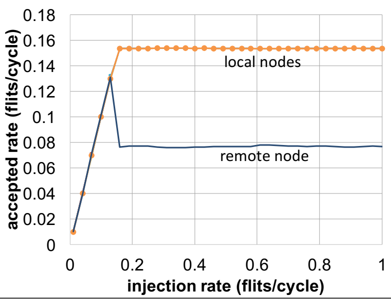


Figure 18. Impact of using different weight between local and remote node arbitration.

the bandwidth compared to the remote nodes, for a similar traffic pattern used earlier in Figure 9. In addition, depending on when entries are pushed into the history queue, the arbitration weight can be modified. For example, one flexibility can be "should global fairness be provided based on packets or simply bandwidth used by the various sources?" Some packets can consist of a large number of flits while other packets consist of short, single-flit packets. Depending on when the history information is queued, the weight adjusted to a different type of packets can be modified in a flexible manner. The history queue can also be pushed with "empty" elements to pop the oldest element of the queue and adjust the weight as desired. Our evaluation of the history-based arbitration did not include clearing or resetting the history; however the history can also be cleared as necessary. For example, if a particular source has not used the network bandwidth for a long period of time, keeping an old history can provide instantaneous unfairness to that particular source.

5.6 Software Solutions and Scheduling

In a NUMA system, while local memory is obviously preferred, accessing remote memory can commonly occur for various reasons, including the result of thread/page migration or shared data structures. For example, a first-touch allocation policy is commonly used in NUMA systems; thus, shared data structures (e.g., hash tables and dictionaries) might be allocated to a single memory node while the workers (or threads) are distributed across different sockets. As a result, the problem that we demonstrate in this work is not necessarily the result of a naive or simple NUMA scheduler.

However, it is possible that a better scheduler or a software solution can minimize the impact of unfairness. For example, alternative Xen schedulers (e.g., [27]) can minimize performance degradation and unfairness. In our example earlier in Figure 4, the scheduler would identify the local node is experiencing high penalty and switch it with the remote vCPU that has lower penalty. If all of the local vCPUs are switched with the remote node equally, all of the vCPUs can achieve fairness but can result in additional overhead, which include additional context switches and other indirect overheads including cold cache misses caused by migrating the vCPUs to a different node. In addition, other approaches such as MemGuard [35] can be extended to NUMA systems to reduce the interconnect unfairness that we identified. Similar to other software-based scheduling, the biggest benefit of MemGuard is that no hardware changes are needed, but it also introduces some limitations, including coarse-grain bandwidth reservations as it is done by the OS.

Prior work [10] proposed an alternative NUMA scheduling that incorporates congestion at the memory controller and the interconnect in the scheduling decision. While we have demonstrated the unfairness in the processor-interconnect, the unfairness can possibly be exploited in NUMA scheduling as overall system performance can actually increase in some configurations by placing workloads remotely. For example, assume 6 memory-intensive workloads need to execute on Node 0 of Figure 4 and access its local memory. Instead of scheduling all of the workloads on Node 0, distributing one workload on the remote node (Node 1) and the remaining 5 workloads on Node 0 resulted in approximately 10% improvement in overall system performance, compared with placing all 6 workloads on Node 0 for the `libquantum` workload. However, further attempting to load-balance by placing more workloads remotely did not improve performance. The overall performance also depends on what other workloads are scheduled on the remote nodes, as shown earlier in Section 3.3. Thus, it remains to be seen how this unfairness and the way the processor-interconnect bandwidth is shared can be exploited in NUMA scheduling.

6. Discussion

In this work, we evaluated the impact of the shared processor-interconnect resource but there are other shared resources in a multi-socket server, including the cache and the memory controller. In this section, we discuss the impact of such shared resources as well as the cost of the proposed scheme.

Shared Cache: Tang et al. [33] evaluated the impact of multi-socket NUMA servers, and in particular, they evaluated the impact of a shared LLC. They showed how reducing cache contention can have a significant impact on NUMA performance. In this work, we tried to minimize the impact of the shared LLC by running independent workloads, and there was minimal amount of sharing between the workloads. However, it remains to be seen how the cache and the interconnect resource in a NUMA server can be properly shared to maximize overall performance. Similar to avoiding contention in the cache [33], minimizing contention in the processor-interconnect can also have a significant impact on overall performance.

Memory Controller: The memory controller (and the memory bandwidth) is another critical shared resource in multi-socket systems. There has been a significant amount of work done on providing fairness in the memory controller (Section 7). However, the memory controller is limited to the requests that it receives – if the requests arriving at the memory controller are significantly unfair because of congestion in the interconnect, the memory controller and its schedul-

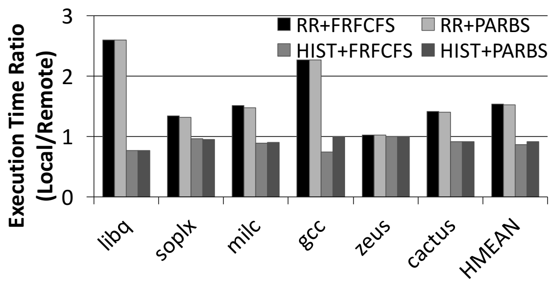


Figure 19. Impact of memory scheduler and processor-interconnect arbitration.

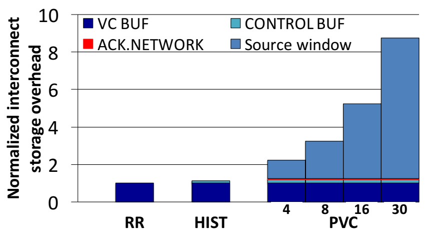


Figure 20. Normalized interconnect storage overhead. The history-queues and the PVC per-flow state are shown as control buffers.

ing can not necessarily provide global fairness. In Figure 19, we evaluate an alternative memory scheduler (PAR-BS [23]) and compare it with both round-robin (RR) arbitration and history-based (HIST) arbitration. For this particular workload, we run a mix of SPEC workloads (sphinx3, leslie3d, GemsFDTD, dealII, bzip2) across five local cores. Another SPEC workload is simulated on the other local core, and the same workload is also simulated on the remote core. The performance ratio between the same workload on local and remote cores is shown in Figure 19. As shown in the results, simply changing the memory scheduling to a fair memory scheduler (RR+PAR-BS) has very minimal impact on the overall unfairness. However, using a fair interconnect arbitration (HIST) with a greedy memory scheduler (FR-FCFS) helps to remove the global unfairness in the system. For most of the workloads, the combination of a fair interconnect arbitration (HIST) and a fair memory scheduler has minimal impact. However, it is interesting to note that for one particular workload (gcc), this combination further reduces the unfairness in the system. Our proposed history-based arbitration is not application-aware ([9]), and it remains to be seen if the processor-interconnect can incorporate such application characteristics. In addition, it remains to be seen if the fairness provided in the interconnect and the memory scheduler can be optimized jointly to improve overall fairness and/or simplify the scheduling (arbitration) within the interconnect and at the memory controller [34].

Cost Comparison: While an *ideal* age-based arbitration provided fairness, implementing age-based arbitration incurs significant overhead to handle aging, arbitration, and counter rollover [1]. In addition, changes are also required not only for the routers but also for the core interfaces as age needs to be maintained from when the core injects the requests. Limited hop count in a processor-interconnect can also result in “false” aging – for example, higher zero-load latency from a remote node (with the additional delay through the QPI interconnect and interface) can unfairly bias remote node traffic at low load compared with a local node.

In terms of performance, previously proposed QoS schemes for NoC, including PVC [12] and GSF [19], provide similar performance as the proposed history-based arbitration. However, the cost of these NoCs is not necessarily appropriate for processor-interconnects. We compare the cost of our proposed history-based arbiter with a baseline router using round-robin arbitration and PVC [12] in Figure 20 in terms of interconnect storage. The history-based arbiter only introduces 12% additional storage, mostly from the history queue and the counters. As for PVC, the additional storage required in the ACK/NACK network and the per-flow state within each router is relatively small, 5% and 19% respectively; however, the additional storage for the source window can be significant. Although the network size is small, with 6-way concentration, a source window is necessary for each core or source. In our simulation environment, a source window of 4 entries was sufficient, but it resulted in over 2× interconnect storage overhead. However, the inter-router latency will likely be much higher in production processor-interconnect, with the additional latency through the I/O components, backplane, etc. Since the source window is proportional to the round-trip latency in the network, the cost of PVC will increase further. In addition, another significant component of cost in PVC is the additional bandwidth required for the ACK/NACK network. If we assume a single-flit ACK/NACK network [12], there is no performance impact, but an almost 2× increase in the processor-interconnect bandwidth cost is incurred because the ACK/NACK network would need to be 16 bits wide (compared to 20 bits for QPI channels). However, if the width of the ACK/NACK is reduced, the overall performance is also degraded. For example, reducing the ACK/-NACK network width by a factor of 2 still results in 40% overhead, while 7% performance degradation was observed for the 2-node processor-interconnect system evaluated in this work, and up to 18% degradation for larger networks.

7. Related Work

There are well-known arbitration algorithms that provide strong fairness, or more generally *quality-of-service* (QoS), many of which have been introduced in the context of IP and multi-chip multiprocessor networks. Age-based arbitration [1, 8], fair queueing [11], virtual clock [36] and fairness clock [36] are the best known fairness algorithms among those. While proven in the field, they require per-flow

	On-chip Networks	**Off-chip Networks**	**Processor Interconnect**
Buffers	$$$	$	$$
(entries)	<10	>100	10-20
Bandwidth	$	$$$	$$
(bits/lane)	∼128-512	∼3	∼20

Table 3. Qualitative comparison of the different type of interconnection networks ($: cheap, $$$: expensive).

queues and states, which make them not as viable in a buffer-constrained on-chip environment. Consequently, many QoS proposals for NoCs are based on existing scheduling algorithms for off-chip environments, including [5, 12, 19–21]. However, most of these proposals require the user to calculate fair share for a given traffic pattern and to set the QoS parameters accordingly. In addition, most of these approaches take advantage of opportunities that an on-chip environment offers, which are not available for a processor-interconnect – e.g., a low-latency on-chip barrier network [19] or a separate ACK network [12]. While the cost of these features is low for an on-chip network, these approaches are much less practical for a multi-socket system. In addition, many on-chip networks assume a synchronous clock domain, which is generally not the case for a multi-socket system. Prior work [30] introduced similar global unfairness caused by locally-fair arbitration in a multi-socket system, but they did not propose any arbitration mechanism to enable global fairness

A high-level qualitative comparison of the different types of interconnection network is shown in Table 3. For on-chip networks [12, 19, 20, 26], buffers are the critical resource, while for off-chip networks [8, 16, 36], bandwidth is the critical resource. For the processor-interconnect, given the packaging constraints, both the buffers and the bandwidth need to be carefully allocated. For example, in the Intel QPI [24], the width of a channel is 20 lanes and provides more bandwidth than off-chip networks but less than on-chip networks. As a result, the bandwidth usage needs to be carefully considered as well as the cost and complexity. In this work, we propose a low-cost, history-based arbitration scheme that balances both bandwidth overhead and complexity to provide global fairness in processor-interconnects. Prior work [31, 32] have investigated processor-interconnect in terms of security vulnerability but do not discuss performance evaluation of processor-interconnect.

There has been a significant amount of work done on providing fairness in memory scheduling [17, 22, 23] and they provide different trade-offs on performance and fairness. To provide fairness in main memory systems, the last access time from individual cores [25], the amount of slowdown of each core due to contention [22], and the number of requests per core in a short period of time (around a μs) [23] have been proposed to determine priority between competing requests. Even if some of the proposals collect short-term memory or history of accesses, similar to this work, all of these studies have considered only the requests that have arrived at MCs and are not able to deal with the global unfairness caused by processor-interconnects.

8. Summary

In this work, we demonstrated on real multi-socket servers how global unfairness in processor-interconnects can result in performance unfairness. In particular, the external concentration implementation of processor-interconnect routers can result in remote nodes obtaining more bandwidth than local nodes, resulting in higher performance for remote nodes. As a case study, we also demonstrated how this unfairness can impact performance variation for VMs. To overcome this unfairness, we proposed a hardware-based solution through a novel history-based arbitration to provide an efficient, globally-fair arbitration in the processor-interconnect.

While our approach provides fairness, we do not argue that fairness is always necessary in a system. However, we demonstrate an intriguing counter intuitive behavior in multi-socket systems that is caused, to the best of our knowledge from our analysis, by the processor-interconnect. While different software solutions can be used to provide fairness, they result in higher overhead from frequent rescheduling or only provide coarse-grain fairness to reduce the overhead from software solutions. This work is one of the first work to identify how the processor-interconnect architecture can negatively impact the behavior of an overall NUMA-system. It remains to be seen how the interaction between NUMA scheduling and the interconnect bandwidth can be jointly optimized to achieve better performance.

Acknowledgments

We thank the anonymous reviewers for their feedback and Changhyun Kim for this contributions. This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2015M3C4A7065647).

References

[1] D. Abts and D. Weisser. Age-Based Packet Arbitration in Large-Radix k-ary n-cubes. In *ICS*, 2007.

[2] J. Ahn, S. Li, O. Seongil, and N. P. Jouppi. McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling. In *ISPASS*, 2013.

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

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP*, 2003.

[5] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS Architecture and Design Process for Network on Chip. *Journal of Systems Architecture*, 2004.

[6] P. Conway and B. Hughes. The AMD Opteron Northbridge Architecture. *IEEE Micro*, 2007.

[7] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Iinterconnection Networks. In *DAC*, 2001.

[8] W. J. Dally and B. P. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[9] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Application-Aware Prioritization Mechanisms for On-Chip Networks. In *MICRO*, 2009.

[10] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. In *ASPLOS*, 2013.

[11] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *SIGCOMM*, 1989.

[12] B. Grot, S. W. Keckler, and O. Mutlu. Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip. In *MICRO*, 2009.

[13] Intel. An Introduction to the Intel QuickPath Interconnect, 2009. URL `http://www.intel.com/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf`.

[14] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim. A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator. In *ISPASS*, 2013.

[15] R. E. Kessler and J. L. Schwarzmeier. CRAY T3D: A New Dimension for Cray Research. In *COMPCON*, 1993.

[16] J. H. Kim and A. A. Chien. Rotating Combined Queueing (RCQ): Bandwidth and Latency Guarantees in Low-Cost, High-Performance Networks. In *ISCA*, 1996.

[17] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *MICRO*, 2010.

[18] P. Kumar, Y. Pan, J. Kim, G. Memik, and A. Choudhary. Exploring concentration and channel slicing in on-chip network router. In *NOCS*, 2009.

[19] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks. In *ISCA*, 2008.

[20] M. M. Lee, J. Kim, D. Abts, M. Marty, and J. W. Lee. Probabilistic Distance-based Arbitration: Providing Equality of Service for Many-core CMPs. In *MICRO*, 2010.

[21] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip. In *DATE*, 2004.

[22] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*, 2007.

[23] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems. In *ISCA*, 2008.

[24] B. Mutnury, F. Paglia, J. Mobley, G. K. Singh, and R. Bellomio. QuickPath Interconnect (QPI) Design and Aanalysis in High Speed Servers. In *EPEPS*, 2010.

[25] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing Memory Systems. In *MICRO*, 2006.

[26] J. Ouyang and Y. Xie. LOFT: A High Performance Network-on-Chip Providing Quality-of-Service Support. In *MICRO*, 2010.

[27] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu. Optimizing Virtual Machine Scheduling in NUMA Multicore Systems. In *HPCA*, 2013.

[28] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar. A 2 Tb/s 6 x 4 Mesh Network for a Single-Chip Cloud Computer with DVFS in 45 nm CMOS. *IEEE Journal of Solid-State Circuits*, 2011.

[29] G. Sartori. Hypertransport Technology. In *Platform Conference*, 2001.

[30] W. Song, H. J. Jung, J. Ahn, J. Lee, and J. Kim. Evaluation of performance unfairness in numa system architecture. *IEEE Computer Architecture Letters*, 2016.

[31] W. Song, J. Kim. D. Abts, and J. Lee. Security Vulnerability in Processor-Interconnect Router Design. In *CCS*, 2014.

[32] W. Song, H. Choi, J. Kim, E. Kim, Y. Kim, and J. Kim. PIkit: A New Kernel-Independent Processor-Interconnect Rootkit. In *USENIX Security*, 2016.

[33] L. Tang, J. Mars, X. Zhang, R. Hagmann, R. Hundt, and E. Tune. Optimizing Google's Warehouse Scale Computers: The NUMA Experience. In *HPCA*, 2013.

[34] G. L. Yuan, A. Bakhoda, and T. M. Aamodt. Complexity effective memory access scheduling for many-core accelerator architectures. In *MICRO*, 2009.

[35] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms. In *RTAS*, 2013.

[36] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. In *SIGCOMM*, 1990.