2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems

NoC Scheduling for Improved Application-Aware and Memory-Aware Transfers in Multi-Core Systems

Tejasi Pimpalkhute, Sudeep Pasricha
Department of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, U.S.A.
tejasi@rams.colostate.edu, sudeep@colostate.edu

Abstract– *In a multi-core environment with several applications executing in parallel, system performance is significantly impacted by network and memory performance. The manner in which network packets and off-chip memory bound packets are handled determines end-to-end latencies across the network and memory. Several techniques have been proposed in the past that schedule packets in an application-aware manner or memory requests in a DRAM row/bank locality aware manner. In this paper, we propose a holistic framework that integrates novel scheduling techniques for both network and memory accesses and operates cohesively in an application-aware and memory-aware manner to optimize overall system performance. Experimental results indicate that our technique performs well for systems with high speed memories, improving system throughput by up to 16.7%, memory latency by up to 11%, and energy consumption by up to 10% compared to prior work on NoC packet scheduling.*

1. Introduction

Network-on-chip (NoC) architectures are becoming the backbone of modern multi-core systems due to their superior scalability and adaptability compared to buses [1]-[2]. Typically, in today's multi-core systems there are several applications co-running at any given time and sharing critical resources such as the NoC fabric, caches, memory controller, and main memory. In such a shared environment, the application packets are constantly in competition to win the race to reach their destination. Consequently, they are always interfering with other application packets resulting in slowdown of applications in the shared environment as opposed to when they are running alone. Determining which packets get precedence over the others has a significant impact on overall system performance. An inefficient NoC packet scheduling strategy can starve critical applications and reduce system throughput. Thus, it is important that requests are scheduled in an application-specific manner that is globally optimal, so as to minimize network latency and benefit overall system performance.

Application data and instruction requests may encounter misses in the on-chip cache hierarchy, requiring request packets to traverse the NoC to reach a memory controller and subsequently access off-chip main memory. For such requests, in addition to network latency, a notable main memory latency is added that increases the overall end-to-end packet latency. The memory latency can be elevated if packets experience stalls in the main memory due to bank conflicts and data contention. It is the responsibility of the memory controller to re-order requests to the main memory so that memory stall cycles are minimized and memory utilization is maximized. The memory controller accomplishes this by keeping track of which bank is being utilized currently and preparing against bank conflicts by scheduling packets to maximize bank level parallelism (BLP). Intelligent techniques for memory scheduling can improve memory latency and end-to-end latency, thereby improving overall system throughput [31]. In this paper, we will refer to the on-chip cache requests as network packets and off-chip memory requests as memory packets.

There are thus two critical stages where packet scheduling is important: in the NoC (specifically on-chip routers) and at the memory controller. Previous work in this area has proposed application-aware techniques e.g., [3]-[8] and memory-aware techniques e.g., [9]-[12] to impart intelligence in the shared resources (NoC routers and memory controller) to improve overall system performance. These application-aware and memory-aware techniques leverage certain characteristics of packets (e.g. throughput intensiveness) or memory (bank level parallelism) to lower latencies and increase throughput. However, the interdependence of application requests across the NoC and to memory causes these techniques to behave sub-optimally.

In this paper, we propose a novel framework to schedule packets in multi-core systems that are multi-programmed (i.e., running multiple applications), such that network latency is minimized and stalls in high speed memories are reduced. The proposed framework improves upon prior work that has failed to achieve a comprehensive solution for optimizing end-to-end packet latency because it addresses the challenges of network scheduling and memory scheduling separately. Our work makes the following key contributions:

- We discuss fairness issues in multi-programmed NoC-based systems and propose a novel anti-starvation algorithm to prevent slowdown of applications and boost overall performance.
- We evaluate the constraints on BLP in modern high speed memories and propose an improved memory scheduling algorithm.
- We propose to employ our memory scheduling algorithm in NoC routers in the vicinity of the memory controller (instead of modifying the controller) for better scalability and improved performance.
- We adapt a multi-level scheduling framework for network and memory packets and demonstrate its benefits on overall system performance, compared to some of the best prior works in this area.

Our experimental evaluations indicate that the proposed scheduling framework not only outperforms the traditional scheduling technique used in many multi-core systems, but also improves upon some of the best performing recent works in the area of packet scheduling.

2. Related Work

The potential of efficient packet scheduling in influencing overall performance for multi-core systems is well recognized. Several recent efforts have proposed architectural and algorithmic strategies for either NoC packet scheduling or memory-bound packet scheduling. We summarize a representative subset of these techniques below.

NoC architectural techniques: Several efforts propose to architect NoCs to cope with diverse application communication demands. In [14], Grot et al. propose KiloNoC, a topology-aware QoS-enabled NoC architecture. In [5], Mishra et al. propose sub-networks for separating compute-intensive and memory-intensive applications in NoC based systems. For the sub-networks proposed in [5], additional hardware circuitry redirects packets injected into the network into a sub-network based on the application type. In [13], the authors propose using heterogeneous routers that can more effectively match application packet QoS requirements while saving power. In [15], Phadke et al. propose a heterogeneous main memory architecture for optimizing latency, bandwidth and power requirements. All of these prior works demand extensive customization of hardware blocks which may be costly or even impractical for commodity processing cores and memory.

Memory-bound packet scheduling techniques: Main memory requests often face delays due to stalls at the head of the issue queue owing to bank conflicts or data contention. Hence, it is important to re-order these requests, as has been explored in some previous efforts on memory scheduling. For example, in [9], SDRAM-aware NoC routers are proposed that are programmed with the main memory timing parameters, allowing for request re-ordering in a memory-aware manner to improve performance. Other techniques also consider thread criticality and fairness while scheduling requests in the memory controller [6][7][16][17]. However, such scheduling algorithms have a large communication overhead due to limited visibility of the network.

NoC packet scheduling techniques: Das et al. propose NoC scheduling using the ranking metrics stall time criticality [3] and packet slack [4] for co-running applications. Stall time criticality is calculated based on the number of L1 cache misses per instruction (L1MPKI) and packet slack is a measure of delay tolerance of a particular packet in the NoC. However, L1MPKI cannot be used as a fine grain classifier of applications (as discussed in Section 3) and packet slack does not

1063-9667/14 $31.00 © 2014 IEEE
DOI 10.1109/VLSID.2014.47

consider network conditions while determining slack. So, these metrics are not very accurate for ranking application criticality. Recently proposed network scheduling schemes in [18] aim to bring uniformity in network latencies and main memory bank utilization by applying separate rules to request and response packets. However, the techniques are application-oblivious and can lead to increased delays in the system while trying to equalize packet latencies.

Fair scheduling techniques: Das et al. [3][4] use time-based batching and packet-based batching techniques to ensure anti-starvation in a strict priority enforced environment. In [19], the authors propose a source-based throttling mechanism to ensure fairness in shared CMPs. However, none of these techniques differentiate between on-chip cache requests and main memory requests for their starvation criteria.

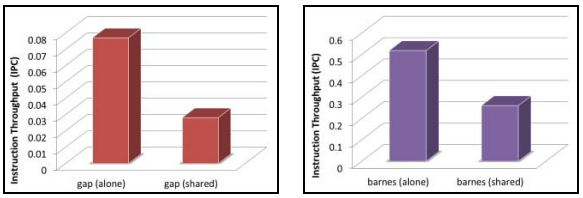
In summary, existing memory scheduling techniques are typically network-unaware, whereas existing network scheduling techniques are agnostic to main memory performance, often being analyzed using simple memory models with fixed latencies. However, we believe that as overall system performance is a function of both network latency and memory latency, focusing on one of these aspects alone to improve performance is sub-optimal. Hence, we propose a novel holistic solution for intelligently scheduling network packets (on-chip cache requests) and memory packets (off-chip requests) to overcome the key drawbacks of existing techniques and improve overall performance.

3. Background and Motivation

In this section, we discuss the challenges of co-running applications in shared chip multiprocessors (CMPs) and the need for devising efficient fairness, packet classification, and scheduling strategies.

3.1 Fairness issues in packet scheduling

Modern multi-core systems typically execute multiple applications in parallel. Such applications inject packets into the NoC, where routers handle the journey of these packets to their destination. Routers frequently must cope with contention while granting the output channel to one of the contending packets. In such scenarios of inter-application interference, it is important to distinguish between packets and make an appropriate choice in prioritizing one packet over others. During such packet scheduling that is usually based on a certain criteria (e.g. stall time criticality [3], packet slack [4]), there is a possibility that certain packets get de-prioritized all the time and suffer from starvation. As a result, some application packets get significantly slowed down. Figure 1 shows the slowdown experienced by the applications *barnes* and *gap* when they are run together in a multi-programmed environment employing an arbitration strategy as in [3] versus when they are run in a standalone manner with no interference from other applications. *Barnes* is a compute-intensive application while *gap* is memory-intensive, thus *gap* injects more packets into the NoC. When these applications are run together, both applications experience some slowdown. As *gap* is memory-intensive and is frequently de-prioritized by the strategy in [3] over the *barnes* compute-intensive application, its application throughput (instructions per cycle) takes a more significant hit (Figure 1). To prevent this slowdown and ensure fairness in the system, anti-starvation strategies are often employed in addition to scheduling algorithms. Some of the standard and previously proposed anti-starvation strategies are discussed below.



(a) (b)

Figure 1: Slowdown for (a) memory intensive workload (gap) (b) compute-intensive workload (barnes) while running standalone vs together.

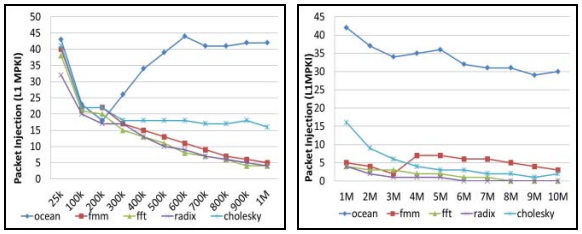
In [3][4][18], *time-based (TB) batching* is employed to enable fairness among application packets. In time-based batching, a packet is tagged with a *batch id* that is a function of the time when it is injected into the NoC. Requests originating at the same time or within the same batching interval form a batch. At the interval of every T cycles, this batch id is incremented cyclically. With the view of enforcing fairness, when two packets belonging to different batches compete with each other, the packet with lower batch id is given priority over the other packet. The underlying assumption is that the older packet has been starved for a long time. However, batching is not necessarily an effective way of coping with starvation. For example, with a typically used batching interval of 16,000 cycles, a packet injected at cycle 0 has batch id 0 and a packet injected at cycle 15,998 also has batch id 0. In the following interval, the batch id is incremented and now the packet originating at cycle 16,002 has batch id 1. If the packet originating at cycle 15,998 competes with one originating at cycle 16,002, the former packet will win out even though it may not be starved, rendering the TB batching technique ineffective in such scenarios.

In another similar technique called *packet-based (PB) batching* [3], instead of a time-based threshold, the threshold is based on number of packets injected. Here the batch id is incremented when N packets have been injected into the NoC. This technique has the same disadvantages as those discussed for TB batching. In addition, this technique requires co-ordination among all the nodes to enable a global batching across the system, resulting in a prohibitive communication overhead.

The two batching techniques discussed above are widely used in NoC packet scheduling to enforce fairness but face inherent drawbacks as discussed above. Moreover, both strategies do not distinguish between on-chip cache requests and off-chip memory requests. Thus, higher ranked off-chip memory packets can defeat lower ranked on-chip cache request packets. However, de-prioritizing lower latency on-chip cache requests can hurt overall performance. Thus, more balanced anti-starvation scheduling mechanisms are essential to aggressively improve performance. Section 4 presents one such mechanism.

3.2 Challenges in packet classification for applications

In a shared CMP with multiple co-running applications, at any given time, two or more packets can be contending for the same resource. Intelligent techniques are required to efficiently allocate resources based on the needs of different applications and their respective network and memory packets. Applications rarely demonstrate invariant characteristics and usually fluctuate between memory-intensive and compute-intensive phases over their lifetime. Figure 2 shows packet injection trends in terms of L1MPKI (L1 misses per 1000 instructions) for applications from the SPLASH-2 benchmark suite. The benchmarks show varying degree of memory-intensiveness at different times and hence the throughput demands of the applications change as well.



(a) (b)

Figure 2: Packet injection trend for benchmarks from SPLASH-2 suite over a window of (a) 1M instructions and (b) 10M instructions.

We can observe from figure 2 that some applications (e.g., *fft* and *radix*) have similar packet injection trend, so using L1MPKI alone to distinguish between packets of different applications during scheduling as done in some prior works (e.g., [3]) is not sufficient. We need a more sophisticated metric to perform finer grain classification of applications. We propose one such metric in Section 4.

3.3 Issues in memory scheduling for high-speed memories

Most modern DDRs and caches are multi-bank architectures that can operate in parallel. A memory request address is composed of its bank address, row address, and column address. Contemporary SDRAMs operate on three main commands: precharge (PRE), activate (ACT) and Read/Write (R/W) [11]. The command generator of the main memory issues the ACT command to the banks if the bank being requested by the memory packet is not already active. A row buffer is used to hold the desired row until the R/W operation is completed. For an open-page policy, this row is held in the row buffer as long as the subsequent requests are *row hits*. A PRE command is required if another request comes to the same bank but desires a different row. The PRE command

restores the previously copied row from the row buffer. It also makes the bank idle if there are no further requests to it.

Generally, multiple banks can be activated in parallel to increase memory throughput. Exploiting bank-level parallelism together with row locality can enable huge gains in performance of memory accesses [20]. However, there is a limit to how many banks of the same rank can be activated in succession in today's high speed memories, to prevent undesirable spikes in the current and power consumption of DRAM. This is represented by the parameter tFAW (Four Activate Window) in DRAM datasheets [21]. In DDR3, for instance, up to 4 banks can be activated without any constraints. If a fifth bank of the same rank needs to be accessed, it has to wait until either one of the already activated banks are pre-charged or for tFAW cycles. Thus, even if we try to maximize BLP by exploiting spatial locality of packets, packets can still experience head of queue latency at the memory if they encounter the above scenario.

Previously proposed memory scheduling techniques (e.g., [5], [9]) give priority to memory packets going to different banks over other packets to maximize BLP. However, in high speed memories such as DDR3, we have a tFAW constraint on maximizing BLP degree. Thus, in such scenarios, existing techniques such as [5] and [9] that attempt to accelerate memory packets going to different banks can end up reducing the overall system performance by prioritizing memory packets (that encounter blocking) over other network packets. Therefore it is crucial that we consider these intricacies of contemporary DRAM in order to perform efficient memory-aware scheduling to improve overall system performance and memory latency.

4. Proposed Dual-Scheduled Packet Framework

To address the challenges discussed in the previous section, we propose a novel Dual Scheduled Packet (DSPK) framework. DSPK is composed of two main components: *(i)* a packet classification technique that is application-aware at the first level and memory-aware at the second level; and *(ii)* an anti-starvation technique employed system-wide to prevent unfairness in the system due to strict priority-based scheduling. Details of our framework are presented next.

4.1 Anti-starvation technique

On-chip L2 cache request packets have a lower overall latency than off-chip memory packets. If the on-chip cache request packets encounter a hit, they can contribute to overall performance (system throughput) to a much greater extent than packets going off-chip. However, when one application is ranked higher than the other, there are instances when L2 packets of lower ranked applications consistently lose out to off-chip memory requests of higher ranked applications. Such L2 packets will get a "starved" status only after a significant amount of time has elapsed (corresponding to the batch interval size) if the batching strategies described in Section 3 are employed. We conjecture that there should be different starvation criteria for L2 requests and off-chip requests. We therefore propose an ***anti-starvation technique*** that decides the "starved" status of a packet depending on not only how long it has been delayed but also depending on its destination. The network interface tags a packet with *initial request time* as soon as it is injected in the NoC. In the switch allocation stage at each NoC router, *current delay* faced by the packet is calculated by subtracting its *initial request time* from current time. We set separate thresholds for L2 and off-chip requests in each router that are calculated based on the running average of *current delay* values for the L2 and off-chip destined packets seen by a router in the last N cycles. If the *current delay* of a network or memory packet exceeds the relevant threshold at a router, then the packet is considered *starved* and immediately granted the output channel over other higher ranked packets. The next subsection discusses how we classify and rank packets.

4.2 Application- and memory-aware packet classification

Applications are frequently classified based on their memory access behavior, as *memory-intensive* if they spend most of their execution time communicating with memory, or as *compute-intensive* if they spend most of their time performing computations and not accessing memory as often. Intuitively, in scenarios with multiple co-running applications, the *compute-intensive* applications contribute to a greater extent towards system throughput (i.e., total instructions executed per cycle across all cores in the system). Therefore, packets from these applications should be given higher priority during scheduling. But it is also important to consider the varying degrees of memory-intensiveness and memory level parallelism (MLP) exhibited by different applications over their lifetime. To capture these unique application-specific characteristics while assigning ranking (priority) weights to packets from applications, we propose using a two-stage approach.

In the *first stage*, we dynamically measure the memory-intensiveness of an application at runtime using the metric average L1 misses per Kilo Instructions (L1MPKI). If the average L1MPKI over a time window is less than a threshold T_{S1}, it can be categorized as a compute-intensive application whose packets deserve higher priority (weightage). On the other hand, if average L1MPKI is less than the threshold, the application is labeled as memory-intensive and its packets assigned lower weights. In the *second stage* of classification, we consider the memory level parallelism (MLP) capabilities of an application to assign its final weights. We measure the MLP of an application using the outstanding count in the request queues of L1 MSHRs (Miss Status Handling Registers) [24]. The length of the MSHR queue has been shown to be directly proportional to MLP exhibited by the application [25]. If this length is less than a certain threshold T_{S2}, the application has lower MLP and is more critical. Thus, applications with low L1MPKI and low MLP are assigned the highest weight (3 on a scale of 0-3). This is followed by applications with low L1MPKI and higher MLP being assigned a weight of 2. Memory-intensive applications with low MLP are assigned a weight of 1 and the applications in the remaining categories are assigned the lowest weight of 0.

We implement this two-stage weighing based classification at the NoC routers two or more hops away from the memory controller. We refer to the technique at this level as ***DSPK-I***. Counters at each nodes network interface keep track of average L1MPKI of the application running at that node and the length of the MSHR queue over a time interval. The counters are reset and weights are assigned at the start of every new interval (every 100K cycles). Header flits of packets injected into the NoC have a 2-bit field that contains the assigned weight. As these flits traverse the NoC, the routers schedule the flits with higher weights over the flits with a lower weight. In the case of tie, priority is given to the packet travelling a shorter distance e.g. L2 cache requests over off-chip memory requests for faster turnaround time.

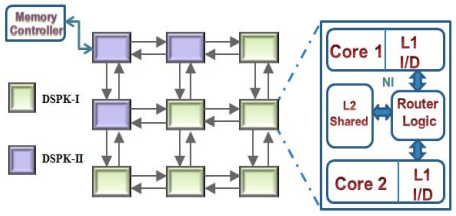


Figure 3: Proposed framework applied to a NoC-based CMP

Most of the network packets are handled by DSPK-I. However, the request packets that miss on-chip caches and must traverse the NoC again to go to off-chip memory must be treated differently, to prepare them against bank conflicts and row misses. Hence, we apply a second-level scheduling algorithm on these memory packets. This algorithm is employed at NoC routers that are a single hop away from the memory controller. We refer to the technique at this level as ***DSPK-II***. The primary focus of here is to schedule memory packets to exploit bank level parallelism and maximize row hits while taking into account the constraints in modern high speed memories.

DSPK-II NoC routers maintain a Used Bank Row (UBR) table for each rank. This table maintains the bank and row history for a maximum of N previous requests for each rank. When memory requests are contending at a DSPK-II router for an output channel towards the memory controller, entries in the UBR table are checked. If the bank and row of the packet destination matches an existing bank and row entry, the packet is given the channel immediately and the request is declared a *row hit*. Otherwise, the bank is declared as *busy* and the packet is held at the router granting the output channel, with preference given to other network packets. This prevents head of the queue stalls at the memory controller due to bank busy requests.

In addition, we also need to consider that no more than four banks can be activated in a rank in a window of T cycles for modern DDR3 memories. To satisfy this constraint, we keep n-bit counters for each rank at DSPK-II routers to count up to T cycles. The value of T is set to tFAW (a maximum 5-bit value specified in DDR3 datasheets) when the number of UBR table entries for any rank becomes equal to 4. If a memory packet going to a rank whose UBR table has 4 or more entries,

even if the entry is not present in the table, it is held back and packets of another rank satisfying the criteria and/or other network packets are scheduled over this packet. Note that the entry from the UBR table is deleted once the bank is free. The pseudo-code below summarizes this scheduling algorithm.

DSPK-II Memory Scheduling Algorithm

Rule I: Applicable only to off-chip memory packets at each cycle:
Decrement each tFAW counter
Check the destination bank and row for packet in UBR
If ((entry absent) && size of (UBR) <= 4))
grant output channel
else if ((entry absent) && tFAW_counter == 0)
grant output channel
else if ((entry present) && Row Hit)
grant output channel
else apply Rule II

Rule II: Applicable only to network request packets:
Schedule L2 request packets of higher-weighted applications first until one of the Rule I criteria is satisfied.

Figure 3 summarizes how our framework is applied to a NoC-based CMP. The figure shows a 3×3 mesh NoC with a concentration degree of 2 (i.e., 2 cores/router), and how our proposed DSPK-I and DSPK-II scheduling techniques are applied to various NoC routers. In addition, the proposed anti-starvation technique is applied to all NoC routers. Together, these techniques constitute a holistic framework for addressing the scheduling challenges facing modern NoC-based multi-core systems.

5. Experiments

5.1 Experimental Setup

We use the cycle-accurate event-driven GEM5 simulator [22] for validating our proposed packet scheduling framework GEM5 is a full system simulator providing support for state-of-the-art out-of-order cores as well as models for a detailed NoC architecture, cache hierarchy, and main memory subsystem [23]. We use the directory-based MESI coherence protocol as our default coherence protocol for the on-chip cache hierarchy. Table 1 shows the configuration of our baseline CMP consisting of a 3×3 concentrated mesh NoC with a concentration factor of 3 (i.e., 3 cores connected to each router) for a total of 27 cores on the die. We assume one-to-one application to core mapping with requests from a core allowed to access L2 cache banks at a remote core. Each NoC router has a state-of-the-art 5-stage pipelined implementation. For our baseline CMP, we use one memory controller connected to node 0 and assume a NUMA configuration to support scalability. The default memory controller services requests in a first come first served manner.

Table 1: Baseline CMP configuration

CPU	1 GHz; out of order;128 instruction queue
L1 Cache	I/D-cache 16 KB, 2-way, 2-cycle latency, cacheline 128B, 16 MSHRs
L2 Cache	Unified, 128kB bank shared, 4-way set associative, cacheline 128B
Main Memory	2GB, DDR3-1333, 2 Ranks/DIMM, 8 Banks/Rank, detailed memory model using open-row policy and row-interleaving
NoC Router	5-stage Virtual Channel Router; Credit-based flow control, 4 VCs per port, 4 buffers/data virtual channel, 1 buffer/ctrl virtual channel
NoC Topology	3×3 2D Concentrated Mesh, Concentration factor 3
Routing scheme	Deterministic X-Y

For our proposed framework, we empirically set the value of threshold T_{S1} as 14 and T_{S2} as 3 in DSPK-I routers. For DSPK-II routers, we set a UBR table size of 8. For the anti-starvation technique, we keep a track of packets arriving at each router over a window of 4096 cycles. We modeled our proposed approach as well as the best known prior works on NoC and memory scheduling for comparison. The prior works and configurations we compare against are: 1) a baseline technique (Round-Robin) that uses a fair round-robin scheduling algorithm in all NoC routers and is widely used in CMPs today; 2) a memory-aware technique [9] that uses SDRAM-specific timing parameters to determine delay and priority of an off-chip memory request at specific routers (MAT); and 3) an application-aware technique (AAT) [3] that employs application-aware scheduling based on batching and application ranking (using L1MPKI values) at all the NoC routers. The parameters used for AAT are: batching interval=16000 cycles; ranking interval=350K cycles; batching as well as ranking levels=8. For MAT, we use DDR3 timing parameters from the Micron datasheet [21] and replace three conventional NoC routers in the vicinity of the memory controller with memory-aware routers. These prior techniques have used a trace-based simulator for their evaluation purposes while we use an event-driven simulator with detailed micro-architecture models. All the simulations in our studies were run for at least 150M instructions. For workloads, we considered 17 diverse applications from the SPLASH-2 [26] and SPEC2K [27] benchmark suites. We first profiled all of the applications and separated them into two categories: compute-intensive and memory intensive. Benchmarks with average L1MPKI values less than *equake* are classified as compute-intensive while the remaining benchmarks are classified as memory-intensive. Figure 4 shows the L1MPKI and memory level parallelism (MLP) exhibited by the benchmarks. We then used different combinations of these benchmarks to create several multi-programmed workloads. Table 2 shows the benchmark combinations we created for our experimental studies. Workloads 1 and 4 are formed by mixing compute-intensive and memory-intensive benchmarks; whereas workloads 2, 3 and 5 represent homogeneous workloads with either all-compute-intensive or all-memory-intensive benchmarks co-running on the system. The number next to a benchmark refers to the number of cores it runs on.

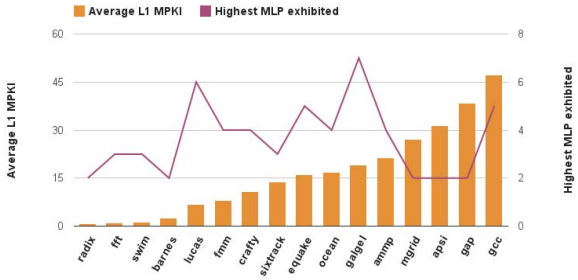


Figure 4: L1MPKI and MLP results of application profiling for benchmarks from the SPLASH-2 and SPEC2K suites

Table 2: Workloads for execution on baseline CMP

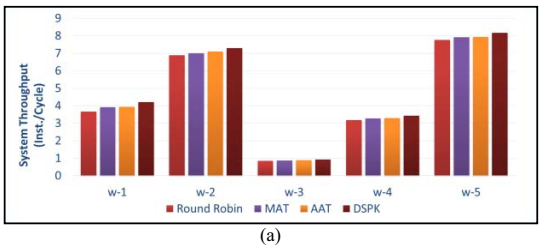
Workload 1 (w-1)	ocean(4), gcc(4), apsi(6), swim(8), applu(5)
Workload 2 (w-2)	lucas(9),barnes(9), radix(9)
Workload 3 (w-3)	gcc(5), ammp(9), galgel(7), gap(6)
Workload 4 (w-4)	crafty(5), gap(7), fft(6), gcc(5), barnes(4)
Workload 5 (w-5)	equake(9), crafty(9), applu(9)

5.2 Experimental Results

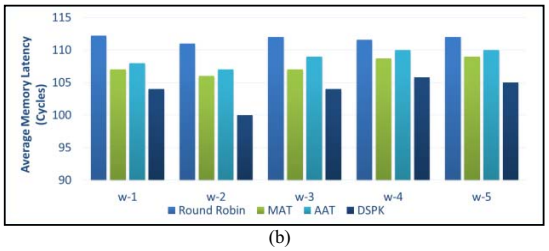
In this section, we provide results for our experimental evaluations for the baseline Round Robin, Application-Aware technique (AAT) [3] and Memory-Aware technique (MAT) [9] compared to our proposed technique (DSPK). In section 5.2.1 we present comparison results on our baseline CMP. In section 5.2.2 we evaluate the impact of memory speed scaling on the effectiveness of scheduling techniques. In section 5.2.3 we explore the scalability of our technique for different network sizes. In section 5.2.4, we compare the energy consumption for the different scheduling techniques. Finally, in section 5.2.5, we evaluate the area overhead of the NoC routers of the techniques considered.

5.2.1 Multi-programmed workload evaluation on baseline CMP

With the growing heterogeneity of applications co-running on modern multi-core systems, it is important that the packet scheduling techniques perform well for various workload combinations. Figure 5 shows the performance comparison and presents results for system throughput and average memory latency for the various scheduling techniques. It can be seen that our proposed DSPK framework outperforms the previously proposed scheduling techniques from [3] and [9], and the baseline round robin scheme for all workload combinations. This improvement is due to the comprehensive nature of DSPK in scheduling both network packets and memory packets efficiently by considering application-specific and memory-specific characteristics of the system, unlike any of the other techniques. DSPK improves system throughput on average by 14.4%/7.4%/6.8% and average memory latency by 11%//6%/7% over the baseline Round Robin, MAT, and AAT techniques, respectively.



(a)

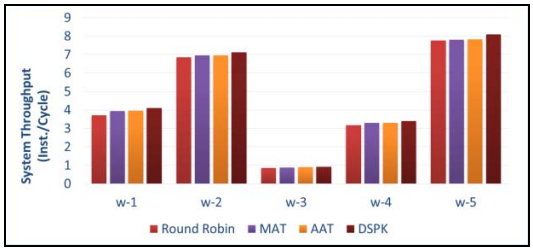


(b)

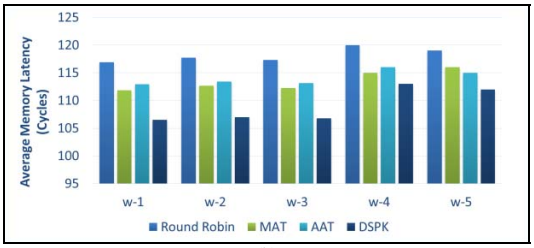
Figure 5: Performance evaluation (a) system throughput (b) average memory latency with DDR3-1333 for workloads from Table 2.

5.2.2 Memory speed scaling evaluation

Today's multi-core systems employ high speed memories to help overcome the "memory wall". However, as memory speed (clock frequency) increases, the number of cycles taken for any memory operation also rises. Any memory-aware scheduling technique should be in compliance with the characteristics and constraints of the memory architecture being utilized in the system. To check for applicability and suitability of our scheduling technique for higher speed memories, we compared its performance to that of other scheduling techniques for the baseline CMP system running with a faster memory model DDR3-1600 [21], instead of the baseline DDR3-1333 model.



(a)



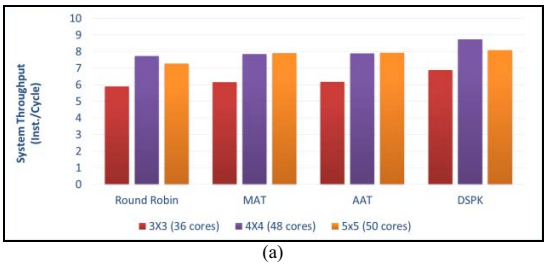
(b)

Figure 6: Memory speed scaling evaluation (a) system throughput (b) average memory latency, for DDR3-1600 with baseline CMP system

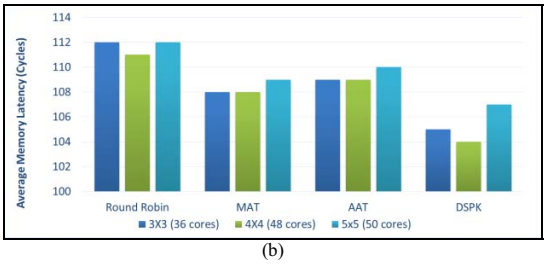
Figure 6 shows the results of this comparison study. We observed that the performance of MAT takes a dip with the higher speed memory, because its memory-aware scheduling algorithm does not consider the four activate window constraint (discussed in Section 3.3) while aiming to maximize bank level parallelism. Packets using the MAT technique face frequent head of the queue stalls at the memory controller thereby increasing its memory latency and lowering its system throughput. AAT is agnostic to memory, and hence it achieves the some performance improvement on the basis of application-aware scheduling. However, DSPK outperforms all the techniques as it is prepared for these issues and handles memory packets and network packets very efficiently ensuring fairness in the system. DSPK achieves an average improvement of 10.3%/4.6%/3.6% for overall system throughput and 10%/5.3%/6% for average memory latency over the baseline Round Robin, MAT, and AAT techniques, thereby proving the superior memory speed scalability of our technique.

5.2.3 Network size scaling evaluation

Next, we were interested in observing how our technique scales with increasing system complexity and for larger network sizes. Therefore we studied three different CMP platform complexities with varying network sizes: a 3×3 concentrated NoC with concentration degree of 4 (36 cores), a 4×4 concentrated NoC with concentration degree of 3 (48 cores), and a 5×5 concentrated NoC with concentration degree of 2 (50 cores). Figure 7 shows the results for system throughput and memory latency averaged over all workloads from Table 2 adapted for the three different platforms considered. We observed that our proposed DSPK framework works even better when congestion in the network is higher, as it creates many more opportunities to classify packets and schedule them efficiently. We also noted that AAT ranks applications according to their relative L1MPKI requiring co-ordination among all the nodes in the system to enable a global batching; hence for workloads running on larger network sizes, the technique has a higher overhead making it less effective. MAT focuses only on memory packets with the help of its three memory-aware routers while ignoring network packets, which results in lower performance for larger network sizes with more network packet dominated communication flows. DSPK has an edge over all of these techniques owing to its holistic nature, providing an average improvement of 16.7%/11.8%/11.4% for overall system throughput and 6.7%/3.8%/4.8% for average memory latency over the baseline Round Robin, MAT and AAT techniques for the three CMP platforms with DDR3-1333 memory.



(a)



(b)

Figure 7: Network scaling evaluation (a) system throughput (c) average memory latency for 3×3 (36 core), 4×4 (48 core) and 5×5 (50 core) CMPs

5.2.4 Energy consumption evaluation

We were also interested in evaluating the energy consumption of DSPK and comparing it with the other previously proposed scheduling techniques. We calculated the power for out-of-order cores, on-chip network, and memory (caches and DDR3 DRAM) using McPAT [28], Orion 2.0 [29], and CACTI 4.0 [30]. We carried out our experiments on the baseline 3×3 CMP platform with 27 cores, for 3 different workload combinations consisting of compute-intensive workloads (w-2), memory-intensive workloads (w-3), and a mix of these two types of workloads (w-1). Figure 8 presents the energy consumption results from this evaluation study. We observed that the baseline round robin scheduling based configuration consumes the highest energy due to its application and memory-oblivious nature, which increases execution time considerably (even though its power overhead is lower than the other techniques due to the simplicity of its implementation). MAT improves upon the baseline round robin technique as it schedules

packets in a memory-operation friendly manner, thereby lowering memory latency and saving memory energy. AAT is unaware of the memory but does application-aware scheduling efficiently for w-1 and w-2 workloads where network packets are in the majority, while for w-3 its energy consumption increases as the memory packets in the network increase and AAT is unable to handle them efficiently. DSPK on the other hand has lower energy consumption than all of these techniques as it is able to handle all kinds of workloads efficiently. We observe an average improvement of 10%/5%/4.7% for overall energy consumption with DSPK over the baseline round robin, MAT and AAT techniques.

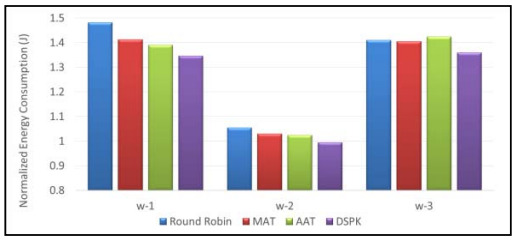
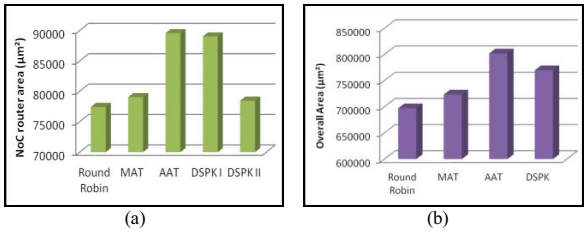


Figure 8: Normalized energy consumption across mixed (w-1), compute-intensive (w-2) and memory-intensive (w-3) workloads.

5.2.5 Area overhead estimation

Lastly, we evaluated the area overhead for our technique in comparison with others techniques. Figure 9(a) shows the area of the intelligent routers in DSPK and other scheduling techniques, for the 32nm CMOS technology node. The round robin scheme is the simplest and therefore not surprisingly has the lowest area overhead. MAT keeps memory-specific information at each router and stores the complete addresses of winners from previous arbitrations, which increases its overhead. AAT stores the L1 MPKI of each core and has the circuitry to compute ranks of all the cores, which significantly increases its area overhead. DSPK uses small counters locally, and also operates on a smaller header flit compared to AAT (as it does not carry a larger batch id and only needs 2-bit rank data). Therefore, its overhead is lower than the AAT technique. We also calculated the total area of the NoC fabric with each of the scheduling techniques, and found that DSPK adds only 9.4% to the overall NoC area over the baseline round robin scheme and 6% more area over the MAT scheme. When compared to the AAT scheme, DSPK has 4.4% lower area overhead. Thus our DSPK framework improves system throughput and reduces memory latency and energy consumption at the cost of a slight area increase.



(a) (b)

Figure 9: Area estimation for scheduling techniques: (a) NoC router area and (b) overall NoC fabric area.

6. Conclusion

In this paper, we addressed the packet scheduling challenge faced by state-of the-art NoC-based CMPs. We discussed the shortcomings with some of the best performing recently proposed works on packet scheduling. To overcome these shortcomings, we proposed a holistic framework to optimize the end-to-end latency of packets in CMPs with multiple co-running applications. Due to the multi-level nature of our technique, it can intelligently identify the specific network or memory characteristics of a packet and give it appropriate priority in arbitration decisions. We also proposed a novel anti-starvation mechanism for establishing fairness in multi-programmed workload based systems. Our experiments performed using a full-system cycle-accurate event-driven simulator validated our motivation and intuition. Our proposed scheduling framework (DSPK) was shown to achieve up to 16.7%, 11%, and 10% improvements for system throughput, average memory latency, and energy consumption, respectively compared to other previously proposed scheduling techniques. Given its scalability and superior performance across various workload types, we believe that our approach is an attractive option for future multi-core NoC-based systems executing multiple and diverse applications.

Acknowledgement

This research is sponsored in part by grants from NSF (CCF-1252500), SRC, and AFOSR (FA9550-13-1-0110).

References

[1] S. Pasricha and N. Dutt, "On-Chip Communication Architectures", Morgan Kauffman, Apr 2008.
[2] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm", Computer, vol. 35, no. 1, Jan. 2002, pp. 70-78.
[3] R. Das et al., "Application-Aware Prioritization Mechanisms for On-Chip Networks", Proc. MICRO-42, 2009, pp. 280-291.
[4] R. Das et al., "Aérgia: Exploiting Packet Latency Slack in On-Chip Networks", Proc. ISCA 2010.
[5] A. K. Mishra, O. Mutlu and C. Das, "A Heterogeneous Multiple Network-On-Chip Design: An Application-Aware Approach", Proc. DAC 2013.
[6] Y. Kim et al., "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior.", Proc. MICRO-43, 2010.
[7] Y. Kim et al., "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers", Proc. HPCA-16, 2010.
[8] Z. Fang et al., "Core-Aware Memory Access Scheduling Schemes", Proc. IEEE IPDPS, 2009, pp. 1-12.
[9] W. Jang and D. Pan, "An SDRAM-Aware Router for Networks-on-Chip", Proc. DAC 2009, pp. 800-805.
[10] R. Ausavarungnirun et. al., "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems", Proc. ISCA 2012.
[11] S. Rixner et al., "Memory Access Scheduling," Proc. ISCA 2000.
[12] N. Dutt, "Memory-aware NoC Exploration and Design", Proc. DATE 2008, pp. 1128-1129.
[13] A.Mishra, N. Vijaykrishnan and C. Das, "A Case for Heterogeneous On-Chip Interconnects for CMPs", Proc. ISCA 2011, pp. 389-400.
[14] B.Grot et al., "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees" Proc. ISCA 2011.
[15] S. Phadke and S. Narayanasamy, "MLP-Aware Heterogeneous Memory System", Proc. DATE 2011, pp. 1-6.
[16] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems", Proc. ISCA-35, 2008.
[17] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors", Proc. MICRO-40, 2007.
[18] A. Sharifi et al., "Addressing End-to-End Memory Access Latency in NoC-Based Multicores", Proc. MICRO 2012, pp. 294-304.
[19] E. Ebrahimi et al., "Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-core Memory Systems", Proc. ASPLOS 2010.
[20] A. Glew, "MLP Yes! ILP No! Memory Level Parallelism, or, Why I No Longer Worry About IPC", Proc. ASPLOS WACI Session, 1998.
[21] Micron DDR3 Datasheet, http:// download.micron.com/pdf/ datasheets/dram/ddr3/2Gb_DDR3_SDRAM.pdf.
[22] N. Binkert et al., "The Gem5 Simulator", ACM SIGARCH Computer Architecture News, vol. 39, Issue 2, May 2011, pp. 1-7.
[23] N. Agarwal et al., "GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator", ISPASS 2009.
[24] D. Kroft, "Lock-up Free Instruction Fetch/pre-fetch Cache Organization", Proc. ISCA 1981, pp. 81-87.
[25] N. Jerger and L. Peh, "On-Chip Networks", Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers 2009.
[26] S. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations",Proc. ISCA 1995, pp. 24-36.
[27] "SPEC2K Benchmark Suite", http://www.spec.org/cpu2000/
[28] S. Li et al., "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", Proc. MICRO-42 2009.
[29] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A Power-Area Simulator for Interconnection Networks", Trans. Very Large Scale Integration (VLSI) Systems, 20(1), Jan. 2012.
[30] "CACTI 4.0", http://www.hpl.hp.com/research/cacti/.
[31] S. Pasricha, and N. Dutt, "A Framework for Cosynthesis of Memory and Communication Architectures for MPSoC," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(3), pp. 408-420, Mar. 2007.