Balancing DRAM Locality and Parallelism in Shared Memory CMP Systems

Min Kyu Jeong*, Doe Hyun Yoon†, Dam Sunwoo‡, Mike Sullivan*, Ikhwan Lee*, and Mattan Erez*

* *Dept. of Electrical and Computer Engineering, The University of Texas at Austin*
† *Intelligent Infrastructure Lab, Hewlett-Packard Labs*
‡ *ARM Inc.*

{mkjeong, mbsullivan, ikhwan, mattan.erez}@mail.utexas.edu
doe-hyun.yoon@hp.com dam.sunwoo@arm.com

Abstract

Modern memory systems rely on spatial locality to provide high bandwidth while minimizing memory device power and cost. The trend of increasing the number of cores that share memory, however, decreases apparent spatial locality because access streams from independent threads are interleaved. Memory access scheduling recovers only a fraction of the original locality because of buffering limits. We investigate new techniques to reduce inter-thread access interference. We propose to partition the internal memory banks between cores to isolate their access streams and eliminate locality interference. We implement this by extending the physical frame allocation algorithm of the OS such that physical frames mapped to the same DRAM bank can be exclusively allocated to a single thread. We compensate for the reduced bank-level parallelism of each thread by employing memory sub-ranking to effectively increase the number of independent banks. This combined approach, unlike memory bank partitioning or sub-ranking alone, simultaneously increases overall performance and significantly reduces memory power consumption.

1 Introduction

Modern main memory systems exploit spatial locality to provide high bandwidth while minimizing memory device power and cost. Exploiting spatial locality improves bandwidth and efficiency in four major ways: (1) DRAM addresses can be split into row and column addresses which are sent to the memory devices separately to save address pins; (2) access granularity can be large to amortize control and redundancy overhead; (3) *page mode* is enabled, in which an entire memory row is saved in the row-buffer so that subsequent requests with the same row address need only send column addresses; and (4) such *row-buffer hit* requests consume significantly less energy and have lower latency because the main data array is not accessed.

Many applications present memory access patterns with high spatial locality and benefit from the efficiency and performance enabled by page mode accesses. However, with the increasing number of cores on a chip, memory access streams have lower spatial locality because access streams of independent threads may be interleaved at the memory controller [2, 23]. Figure 1 shows, for example, the impact of access interleaving on the spatial locality of the SPEC CPU2006 benchmark `lbm`. The DRAM row-buffer hit rate of one `lbm` instance is shown for three different workload mixes. When `lbm` runs alone, 98% of the accesses hit in the row-buffer. When 4 instances of `lbm` run together, however, the hit rate drops to 50%. The spatial locality of an interleaved stream can drop even further depending on the application mix, because some applications are more intrusive than others. The SPEC CPU2006 benchmark `mcf`, for example, is memory intensive but has very poor spatial locality, and thus severely interferes with the memory accesses of other applications. In a multi-programmed workload containing `mcf`, the row-buffer hit rate of `lbm` drops further to 35%. Loss of spatial locality due to inter-thread interference increases energy consumption because of additional row activations, increases the average access latency, and typically lowers the data throughput of the DRAM system.

Out-of-order memory schedulers reorder memory operations to improve performance and can potentially recover some of the lost spatial locality. However, reclaiming lost locality is not the primary design consideration for these schedulers. Instead, they are designed either to simply maximize the memory throughput for a given interleaved stream (e.g., FR-FCFS [20]), or to maximize some notion of fairness and application throughput (e.g., ATLAS [9] and TCM [10]). In other words, improved scheduling can somewhat decrease the detrimental impact of interleaving but does not address the issue at its root. Furthermore, the effectiveness of memory access scheduling in recovering locality is constrained by the limited scheduling buffer size and the (often large) arrival interval of requests from a single

This work is supported, in part, by the following organizations: The National Science Foundation under Grant #0954107, Intel Labs Academic Research Office for the Memory Hierarchy Innovations program, and The Texas Advanced Computing Center.

978-1-4673-0826-7/12/$26.00 ©2011 IEEE

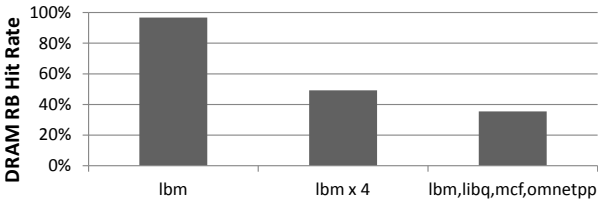


Figure 1: DRAM row-buffer hit rate of `lbm` when run alone, with 4 instances, and when run with other applications. FR-FCFS scheduling is used for higher hit rate. See Section 4 for details of the simulation setup.

stream. As the number of cores per chip grows, reordering will become less effective because the buffer size does not scale and contention for the shared on-chip network can increase the request arrival interval of each thread. With an increased arrival interval, subsequent spatially adjacent requests have a higher chance of missing the scheduling window, and the row is precharged prematurely.

We investigate a fundamentally different approach that addresses locality interference at its root cause. Rather than recovering limited locality with scheduling heuristics, we propose to reduce (and possibly eliminate) row-buffer interference by restricting interfering applications to non-overlapping sets of memory banks. In some cases, however, restricting the number of banks available to an application can significantly degrade its performance as fewer banks are available to support concurrent overlapping memory requests. This is of particular concern because the total number of banks available to a processor is growing more slowly than the total number of threads it can concurrently execute. Therefore, we combine *bank partitioning* with memory sub-ranking [24, 27, 3, 2, 4], which effectively increases the number of independent banks. Together, bank partitioning and memory sub-ranking offer two separate tuning knobs that can be used to balance the conflicting demands for row-buffer locality and bank parallelism. We show that this powerful combination, unlike memory sub-ranking or bank partitioning alone, is able to simultaneously increase overall performance and significantly reduce memory power consumption.

The rest of this paper is organized as follows: Section 2 discusses how the DRAM address mapping affects locality interference and describes our bank partitioning mechanism. Section 3 describes how bank partitioning reduces the available DRAM bank-level parallelism, and investigates how sub-ranking can recover the lost parallelism. Section 4 describes our evaluation methodology, and Section 5 discusses results. Section 6 reviews related work, and Section 7 concludes.

2 Bank Partitioned Address Mapping

Row-buffer locality interference in multicore processors results from the mechanisms used to map the addresses of threads onto physical memory components. Current mapping methods evolved from uniprocessors, which do not finely interleave independent address streams. These methods are sub-optimal for chip multi-processors where memory requests of applications can interfere with one another. In this section, we first describe the mechanism commonly in use today and then discuss our proposed mapping mechanism that simultaneously improves system throughput and energy efficiency.

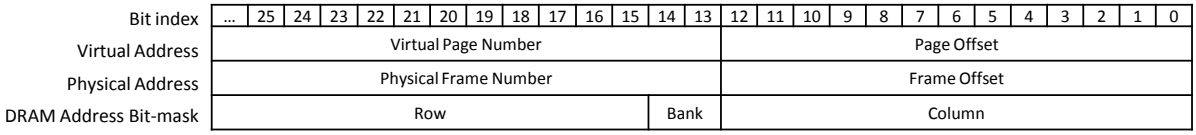
2.1 Current Shared-Bank Address Mapping

In traditional single-threaded uniprocessor systems with no rank-to-rank switching penalty, memory system performance increases monotonically with the number of available memory banks. This performance increase is due to the fact that bank-level parallelism can be used to pipeline memory requests and hide the latency of accessing the inherently slow memory arrays. To maximize such overlap, current memory architectures interleave addresses among banks and ranks at fine granularity in an attempt to distribute the accesses of each thread across banks as evenly as possible. However, because row-buffer locality can be exploited to improve efficiency and performance, it is common to map adjacent physical addresses to the same DRAM row so that spatially close future accesses hit in the row-buffer. Therefore, the granularity of bank interleaving in this case is equal to the DRAM row size.

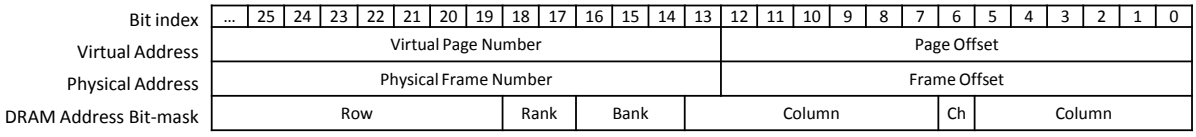
Figure 2 gives an example of the typical row-interleaved mapping from a virtual address, through a physical address, to a main memory cell. The mapping given in Figure 2(a) is for a simple memory system (1 channel, 1 rank, and 4 banks) and is chosen to simplify the following discussion. The mapping in Figure 2(b) is given for a more realistic DDR3 memory system with 2 channels, 4 ranks, and 8 banks. When two threads concurrently access memory, both threads may access the same memory bank because the address space of each thread maps evenly across all banks. Figure 2(c) depicts the full address translation process and shows how two threads with the simple mapping (Figure 2(a)) can map to the same bank simultaneously. When thread P0 and P1 accesses their virtual pages 2 and 3, they conflict in banks 2 and 3. Bank conflicts, such as this one, thrash the row-buffers and can significantly reduce the row-buffer hit rate. The likelihood of this type of interference grows with the number of cores as more access streams are interleaved.

2.2 Bank Partitioned Address Mapping

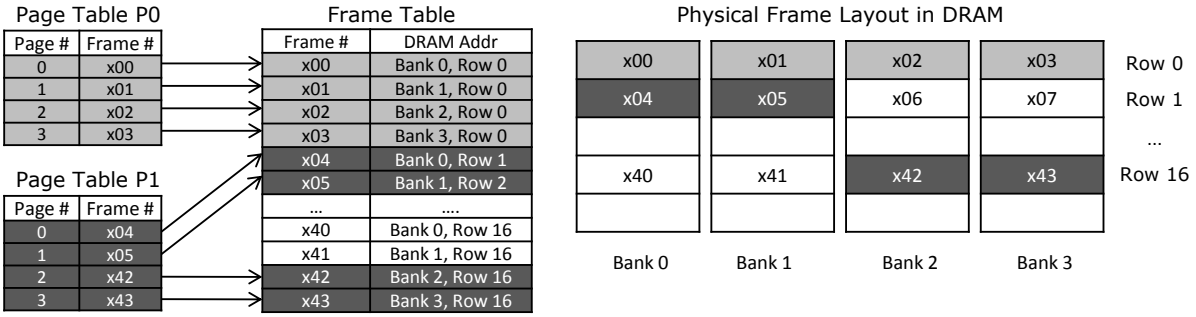
If independent threads are not physically mapped to the same bank, inter-thread access interference cannot occur. We call this interference avoidance mechanism *bank partitioning* as proposed by Mi et al. [13]. Figure 3(a) gives an example address mapping scheme for a simple system which utilizes bank partitioning. In this example, bit 14 of the physical address denotes the core ID. Core 0 is allocated physical frames $\{0, 1, 4, 5, 8, 9, ...\}$, and core 1 is allocated frames $\{2, 3, 6, 7, 10, 11, ...\}$. This mapping partitions the banks into two sets such that processes running on different cores are constrained to disjoint banks, thus avoiding row-buffer interference. Figure 3(c) shows the corresponding change in virtual page layout in DRAM and illustrates



(a) Virtual to physical to DRAM address mapping scheme for a simple memory system.



(b) Generalized Virtual to physical to DRAM address mapping scheme.



(c) The layout of virtual pages allocated to each process in the DRAM banks according to the map (a).

Figure 2: Conventional address mapping scheme

how bank interference is eliminated. Virtual pages in P0 and P1 are mapped to only banks $\{0, 1\}$ and $\{2, 3\}$, respectively. Therefore, they do not share row-buffers and cannot interfere with one another. Figure 4 gives an example of memory behavior with and without bank partitioning. P0 and P1 concurrently access their virtual page #2 in an interleaved manner; bank partitioning eliminates the precharge commands and the second activate command for P0. This improves both throughput and power efficiency relative to the memory behavior using conventional frame allocation.

Figure 3(b) shows a generalized bank partitioning address mapping scheme for a typical DDR3 memory system with 2 channels, 4 ranks, and 8 banks. Different *colors* represent independent groups of banks. Given a number of colors, there is a wide decision space of allocating colors to processes. We can imagine an algorithm that allocates colors depending on the varying applications' need for bank-level parallelism. However, this study uses a static partition which assigns an equal number of colors to each core. Such static partitioning does not require profiling of workloads and is robust to dynamic workload changes. We find that there is a diminishing return in increasing the number of banks that a modern out-of-order core can exploit from more bank-level parallelism. For realistic modern memory system configurations which have internal banks and consist of multiple channels and ranks, sub-ranking can compensate for reduced per-thread bank count. More details on bank-level parallelism will be discussed in Section 3.

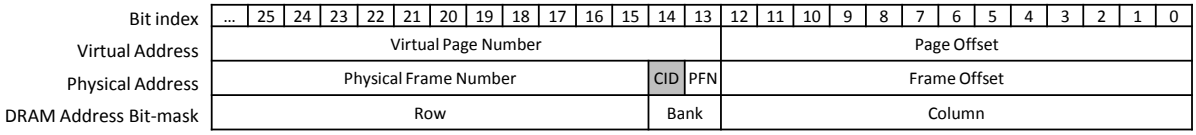
Note that the mapping between physical and DRAM addresses stays the same, allowing the address decoding logic in the memory controller to remain unchanged. Bank isolation is entirely provided by the virtual to physical address mapping routine of the operating system. Therefore, when a core runs out of physical frames in colors assigned to it, the OS can allocate different colored frames at the cost of reduced isolation. The decision of whose color the frames are spilled to can be further explored, but is beyond the scope of this paper.

XOR-permuted bank indexing is a common memory optimization which reduces bank conflicts for cache-induced access patterns [26]. Without any permutation, the set index of a physical address contains the bank index sent to DRAM. This maps all cache lines in a set to the same memory bank, and guarantees that conflict-miss induced traffic and write-backs will result in bank conflicts. XOR-based permutation takes a subset of the cache tag and hashes it with the original bank index such that lines in the same cache set are mapped across all banks.

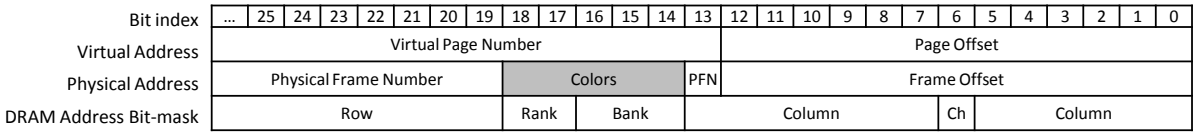
XOR-permuted bank indexing nullifies color-based bank partitioning and re-scatters the frames of a thread across all banks. We use a technique proposed by Mi et al.[13] to maintain bank partitioning. Instead of permuting the memory bank index, we permute the cache set index. In this way, a single color still maps to the desired set of banks. At the same time, cache lines within a set are mapped across multiple banks and caches sets are not partitioned.

3 Bank-Level Parallelism

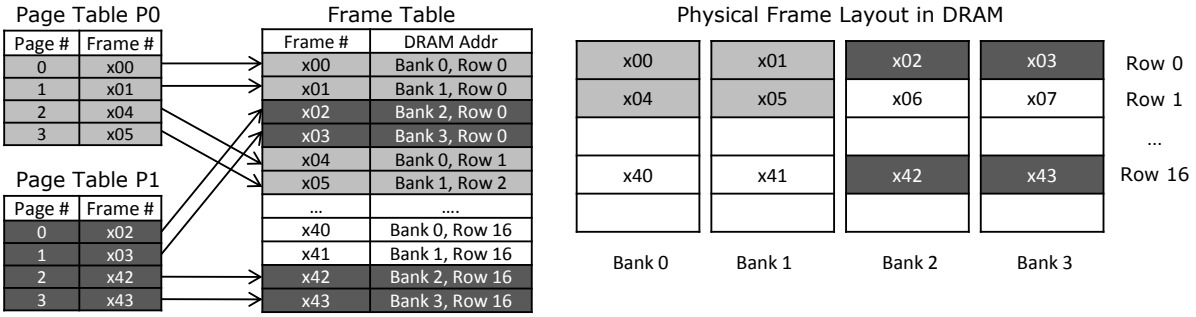
Bank partitioning reduces the number of banks available to each thread. This section examines the impact that re-



(a) Virtual to physical to DRAM address mapping scheme for a simple memory system.



(b) Generalized Virtual to physical to DRAM address mapping scheme.



(c) The layout of virtual pages allocated to each process in the DRAM banks according to the map (a).

Figure 3: Address mapping scheme that partitions banks between processes.

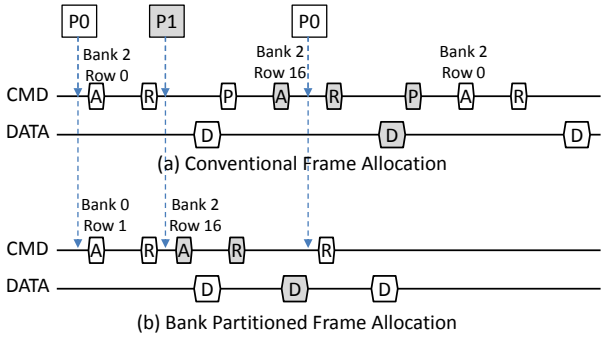


Figure 4: Benefits of bank partitioning. The upper diagram corresponds to Figure 2(c), and the lower diagram corresponds to Figure 3(c). A, R, P, and D stand for activate, read, precharge, and data, respectively. The row-buffer is initially empty.

duced bank-level parallelism has on thread performance, and investigates the ability of memory sub-ranking to add more banks to the system.

3.1 Bank-level Parallelism

The exploitation of bank-level parallelism is essential to modern DRAM system performance. The memory system can hide the long latencies of row-buffer misses by overlapping accesses to many memory banks, ranks, and channels.

In modern out-of-order processor systems, a main memory access which reaches the head of the reorder buffer (ROB) will cause a structural stall. Given the disparity of off-chip memory speeds, most main memory accesses reach the head of the ROB, negatively impacting performance [8]. Furthermore, memory requests that miss in the open row-buffer require additional precharge and activate latencies which further stall the waiting processor. If a thread with low spatial locality generates multiple memory requests to different banks, they can be accessed in parallel by the memory system. This hides the latency of all but the first memory request, and can dramatically increase performance.

A thread, especially one with low spatial locality, can benefit from many banks up to the point where its latency is completely hidden. With DDR3-1600, a row-buffer miss takes about 33 DRAM clock cycles before it returns data (in the following 4 DRAM clock cycles). Therefore, to completely hide the latency of subsequent requests, a thread needs to have 8 banks working in parallel [1] [2]. In general, as more requests are overlapped, the CPU will stall for less time and performance will increase.

While bank-level parallelism undoubtedly improves performance, its benefits have a limit. After the available par-

[1] tFAW limits the number of activate commands to 6 per row-buffer miss latency, due to power limitations. Also, tRRD has increased so that activates should be separated by more than 4-cycle data burst, causing idle cycles on the data bus in the case that all requests miss in the row-buffer.

[2] Since a row-buffer miss takes a constant latency, more bank-level parallelism is needed with faster interfaces such as DDR3-1866 and 2133.

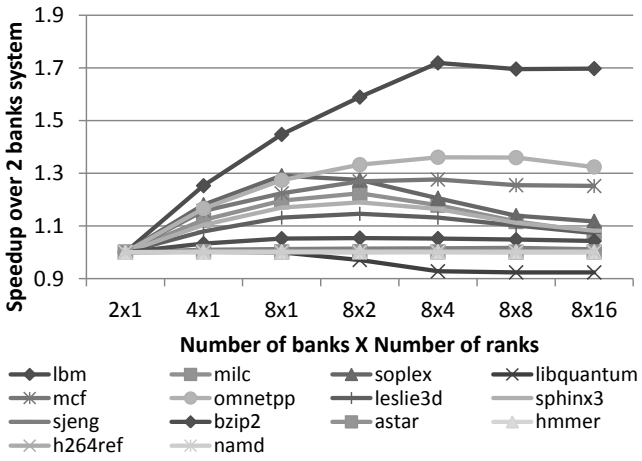


Figure 5: Normalized application execution time with varying number of banks.

allelism can hide memory latencies, there is no further benefit to increasing the number of banks. Figure 5 shows applications' sensitivity to the number of banks in the memory system. Most applications give peak performance at 8 or 16 banks. After this point, applications show no further improvement or even perform worse, due to the 2 cycle rank-to-rank switching delay. `lbm` is an outlier that benefits beyond 16 banks. `lbm` shows a higher row-buffer hit-rate as the number of banks increases– it accesses multiple address streams concurrently, and additional banks keep more row-buffers open and can fully exploit each stream's spatial locality.

3.2 Sub-ranking

The most straightforward way to retain sufficient bank-level parallelism with bank partitioning is to increase the total number of banks in the memory system. The total number of banks in the system is equal to the product of the number of channels, the number of ranks per channel, and the number of internal banks per chip. Unfortunately, increasing the number of available banks is more expensive than simply adding extra DIMMs.

There are a number of factors which complicate the addition of more memory banks to a system. First, and most fundamentally, paying for additional DRAM chips and power just to increase the number of banks is not a cost-efficient solution for system designers. Also, the number of channels is limited by the available CPU package pins. Furthermore, when fine-grained channel interleaving is used to maximize data link bandwidth, the contribution of the channel count to the effective number of banks is limited. In addition, the number of ranks per channel is constrained by signal integrity limitations. Point-to-point topologies as used in FB-DIMMs [12] can overcome this problem, but come at a cost of additional latency and power consumption. Finally, adding internal banks to each DRAM chip requires expensive circuitry changes. Considering the extreme competition and low margin of the DRAM market, more internal banks would be difficult to achieve in a cost effective manner. For these reasons, with continued device scaling and the recent trend of increasing core count, future systems are expected to have higher core to bank ratio than current systems.

We propose an alternative approach to increase the bank-level parallelism without adding expensive resources. We employ DRAM sub-ranking [2, 27, 24, 25] to effectively increase the number of banks available to each thread. DRAM sub-ranking breaks a wide (64-bit) rank into narrower (8-bit, 16-bit, or 32-bit) sub-ranks, allowing individual control of each sub-rank. In the conventional memory system, all banks within a rank work in unison and hold the same row. In the 32-bit sub-ranked system, two groups of four banks are independent and can hold different rows. By controlling the sub-ranks independently, multiple long-latency misses can overlap, effectively increasing the available bank-level parallelism. However, since fewer devices are involved in each request, sub-ranked DRAM takes more reads (and thus a longer latency) to complete the same size of request. Although narrow 16-bit and 8-bit sub-ranks can provide abundant bank-level parallelism, additional access latencies due to sub-ranking become prohibitive. Given a 4GHz processor, 16-bit and 8-bit sub-ranks take 40 and 80 additional CPU cycles, respectively, to access main memory. In addition, narrow sub-ranks put greater pressure on the address bus, which could impact performance or necessitate expensive changes to DRAM. Accordingly, we use 32-bit sub-ranks to balance bank-level parallelism and access latency, and avoid the need for extra address bus bandwidth.

4 Evaluation Methodology

We evaluate our proposed bank partitioning and sub-ranking scheme using Zesto, a cycle-level x86 out-of-order processor simulator [11], and a detailed in-house DRAM simulator [7]. Zesto runs user space applications bare-metal and emulates system calls. We extend the physical frame allocation logic of Zesto to implement bank partitioning among cores as presented in Section 2.2. Our DRAM simulator supports sub-ranked memory systems. It models memory controllers and DRAM modules faithfully, simulating the buffering of requests, scheduling of DRAM commands, contention on shared resources (such as address/command and data buses), and all latency and timing constraints of DDR3 DRAM. It also supports XOR interleaving of bank and sub-rank index to minimize conflicts [26], which is used for the baseline shared-bank configuration and replaced with cache-set index permutation described in 2.2 for bank-partitioned configurations.

System Configuration

Table 1 summarizes the system parameters of our simulated systems. We simulate the following 4 configurations to evaluate our proposed mechanism.

Table 1: Simulated system parameters

Processor	8-core, 4GHz x86 out-of-order, 4-wide
L1 I-caches	32KiB private, 4-way, 64B line size, 2-cycle
L1 D-caches	32KiB private, 8-way, 64B line size, 2-cycle
L2 caches	256KiB private for instruction and data, 8-way 64B line size, 6-cycle
L3 caches	8MiB shared, 16-way, 64B line size, 20-cycle
Memory controller	FR-FCFS scheduling, open row policy 64 entries read queue, 64 entries write queue
Main memory	2 channels, 2 ranks / channel, 8 banks / rank 8 x8 DDR3-1600 chips / rank All parameters from the Micron datasheet [14]

1. `shared`: Baseline shared-bank system.
2. `bpart`: Banks partitioned among cores. Each core receives consecutive banks from one rank.
3. `sr`: Ranks split into two independent sub-ranks.
4. `bpart+sr`: Bank partitioning with sub-ranking used together.

Power models

We estimate DRAM power consumption using a power model developed by the Micron Corporation [1]. For processor power, we use the IPC-based estimation presented in [2]: the maximum TDP of a 3.9GHz 8-core AMD bulldozer processors is reported as 125W; half of the maximum power is assumed to be static (including leakage) and the other half is dynamic power that is proportional to IPC. Our study focuses on main memory and, as such, our mechanisms have a minimal impact on the core power.

Workloads

We use multi-programmed workloads consisting of benchmarks from the SPEC CPU2006 suite [22] for evaluation. Our evaluation is limited to multi-programmed workloads due to the limitation of our infrastructure. The principle of placing memory regions that interfere in access scheduling in different banks to better exploit spatial locality, however, applies to both multi-programming and multi-threading. Identifying conflicting regions can be more challenging with multi-threading, yet doable (e.g., parallel domain decomposition can indicate bank partitions; and thread-private data is often significant).

To reduce simulation time, we use SimPoint [5] and determine a representative 200 million instruction region from each application. Memory statistics from identified region are used to guide the mix of applications in our multi-programmed workloads. Table 2 summarizes the characteristics of each application. Some benchmarks are not included in our evaluation due to limitations of Zesto.

Two key application characteristics which are pertinent to our evaluation are the memory access intensity and the row-buffer spatial locality. The memory access intensity, represented by last-level cache misses-per-kilo-instructions(LLC MPKI), is an indicator of how much an application is affected by the memory system performance. We focus most our evaluation on workloads that include memory intensive applications.

Table 2: Benchmarks statistics when running on the baseline. IPC: Instructions per cycle, LLC MPKI: Last level-cache misses per 1000 instructions, RB Hit Rate: Row-buffer hit rate, and Mem BW: Memory bandwidth used in MiB/S.

Benchmark	IPC	LLC MPKI	RB Hit Rate	Mem BW
lbm	0.71	12.80	97%	3394.56
milc	0.69	16.15	82%	3576.32
soplex	0.53	20.88	90%	3246.08
libquantum	0.53	15.64	98%	2967.04
mcf	0.54	16.15	10%	2258.94
omnetpp	0.69	9.51	49%	2136.83
leslie3d	0.73	8.04	89%	1942.53
sphinx3	1.03	0.66	77%	171.72
sjeng	0.81	0.68	17%	141.49
bzip2	0.94	0.49	82%	114.48
gromacs	1.35	0.54	96%	88.76
astar	0.91	0.17	62%	36.07
hmmer	1.00	0.13	97%	34.20
h264ref	0.71	0.16	81%	28.03
namd	1.09	0.09	91%	23.25

The spatial locality of application, represented by the row-buffer hit rate, is another key characteristic. Applications with high row-buffer hit rates suffer the most from row-buffer locality interference; conversely, they stand to benefit the most from bank partitioning. On the other hand, applications with low row-buffer hit rates rely on bank-level parallelism, and may be adversely affected by any reduction in the number of available banks.

Among the memory-intensive benchmarks, `mcf` and `omnetpp` have low spatial locality. `libquantum`, `lbm`, `milc`, `soplex` and `leslie3d` all demonstrate high spatial locality in the absence of interference.

Table 3 shows the mix of benchmarks chosen for our multi-programmed workloads. Unless otherwise noted, all benchmark mixes contain programs with high memory intensity. Group `HIGH` consists of workloads with high spatial locality. Group `MIX` is a mixed workload of programs with both high and low spatial locality. Group `LOW` consists of workloads with low spatial locality. Finally, group `LOW_BW` is composed of workloads that contain non-memory-intensive benchmarks to see the effect of our mechanisms on compute-bound workloads. Each mix in the table contains one, two, or four benchmarks. These benchmark mixes are replicated to have eight benchmarks to run on each of the eight cores in our simulated system.

Metrics

To measure system throughput, we use Weighted Speedup (WS) [21], as defined by Equation 1. IPC_{alone} and IPC_{shared} are the IPC of an application when it is run alone and in a mix, respectively. The number of applications running on the system is given by N.

Table 3: Multi-programmed workloads composition. Workloads are then replicated to have 8 programs per mix. RBH : Row-buffer hit rate.

	Memory Intensive		Memory non-intensive	
Workload	High RBH	Low RBH	High RBH	Low RBH
HIGH				
H1	lbm,milc,soplex,libq			
H2	lbm,milc,libq,leslie3d			
H3	lbm,milc,soplex,leslie3d			
H4	lbm,soplex,libq,leslie3d			
H5	milc,soplex,libq,leslie3d			
MIX				
M1	milc,libq,leslie3d	mcf		
M2	lbm,milc,libq	omnetpp		
M3	soplex,leslie3d	mcf,omnetpp		
M4	lbm,libq	mcf,omnetpp		
M5	milc,leslie3d	mcf,omnetpp		
M6	soplex,milc	mcf,omnetpp		
M7	libq,soplex	mcf,omnetpp		
LOW				
L1		mcf		
L2		mcf,omnetpp		
L3		omnetpp		
LOW_BW				
LB1	leslie3d,libq,soplex		bzip2	
LB2	milc,leslie3d	omnetpp	sphinx3	
LB3	milc,soplex		sphinx3	sjeng
LB4	lbm	mcf	bzip2	sjeng
LB5	libq	omnetpp	bzip2	sjeng

$$WS = \sum_{i=0}^{N-1} \frac{IPC_i^{shared}}{IPC_i^{alone}} \quad (1)$$

System power efficiency, expressed in terms of throughput (WS) per unit power, is also reported. System power is the combined power consumption of cores, caches, and DRAM. Since we adopt a multi-programmed simulation environment, we report power efficiency, rather than energy efficiency. Each application starts executing at its predetermined execution point. We simulate the mix until the slowest application in the mix executes the desired number of instructions. Statistics per application are gathered only until each core (application) reaches the fixed number of instructions. However, we keep executing the faster applications to correctly simulate the contention for shared resources. When IPCs are compared, the IPCs for the same number of instructions across different configurations are used. Consequently, the same application in a different configuration can complete a different amount of work. For this reason, it is difficult to make fair energy comparisons across configurations; we compare power efficiency instead.

We also use minimum speedup (i.e. maximum slowdown) to determine the fairness of the system [9]. The minimum speedup of a workload is defined by Equation 2

$$MinimumSpeedup = \min_{i=0}^{N-1} \frac{IPC_i^{shared}}{IPC_i^{alone}} \quad (2)$$

The minimum speedup of a workload helps to identify whether a technique improves overall system throughput by improving the performance of some applications significantly while degrading the rest. The harmonic mean of speedups is also widely used as a system fairness metric; we favor the minimum speedup, however, to highlight the disadvantage that a program can suffer due to each approach.

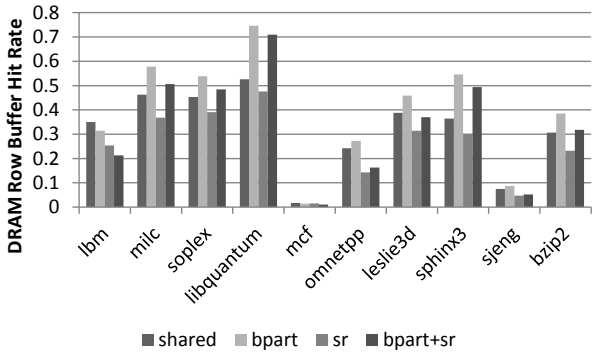


Figure 6: Average DRAM row-buffer hit rate of each benchmark.

5 Results

5.1 Row-buffer Hit Rate

We first present the effect of bank partitioning on row-buffer hit rate. Figure 6 shows the row-buffer hit rate of each benchmark with varying configurations. We gather the per-core (and thus per-benchmark) row-buffer hit rate and average the row-buffer hit rate for the same benchmark across different workloads. For most benchmarks, the row-buffer hit rate improves with bank partitioning. The most improved benchmark is `libquantum`, which recovers half of the locality lost due to inter-thread interference. On the other hand, the row-buffer locality of `lbm` is degraded. This may be surprising considering that `lbm` has a very high row-buffer hit rate (98%) when run alone. As seen in Figure 5, the spatial locality of `lbm` is sensitive to the number of banks. With a reduced number of banks due to bank partitioning, `lbm` cannot keep enough concurrent access streams open and loses row-buffer locality.

With sub-ranking, the row-buffer hit rate drops since the granularity of sub-rank interleaving is 64B and the number of independent, half-sized banks that a program accesses increases. For example, two consecutive 64B accesses would normally result in 50% hit rate, but result in a 0% hit rate once sub-ranking is introduced. Note that this is a start-up cost and is amortized over the subsequent accesses if the application has sufficient spatial locality. However, when sub-ranking is applied on top of bank partitioning, the row-buffer hit rate drops slightly due to this effect.

5.2 System Throughput

Figure 7 shows the system throughput for each memory configuration and workload. As expected, the workload group with many high spatial locality benchmarks (`HIGH`) benefits most from bank partitioning. These benchmarks recover the locality lost due to interference and their increased row-buffer hit rate results in improved throughput. Since many of the benchmarks in group `HIGH` have high

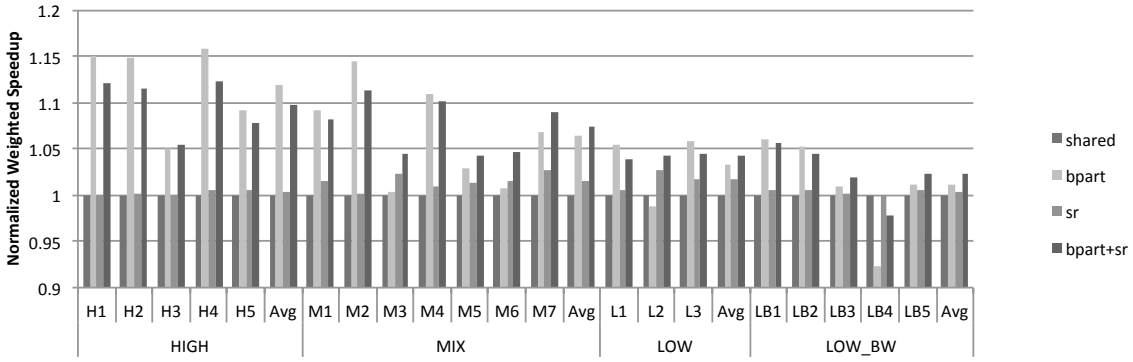


Figure 7: Normalized throughput of workloads.

spatial locality, they do not gain much from the additional banks that sub-ranking provides. Sub-ranking alone does not affect (or slightly improves) performance for the HIGH benchmarks, but it degrades the improvements of bank partitioning when both are applied. With shared banks, inter-thread interference results in many long-latency row-buffer misses. Additional latencies due to sub-ranking are insignificant compared to row-buffer miss latencies (4 cycles vs. 37 cycles). With bank partitioning, however, many requests are now row-buffer hits and take little time (11 cycles). Therefore, the extra latency from sub-ranking becomes relatively more costly and affects performance.

Workload group MIX enjoys less benefit from bank partitioning since it includes benchmarks with low spatial locality. MIX is also moderately improved by sub-ranking. When both techniques are applied together, however, most MIX workloads show synergistic improvement. Exceptions to this observation include workloads M1, which has only one application with low spatial locality to benefit from sub-ranking, and M4, whose performance with sub-ranking is slightly degraded due to the presence of lbm (for reasons described above). M2 suffers from both afflictions, and its performance relative to bank partitioning degrades the most from the addition of sub-ranking, though it is still within 3% of the maximum throughput.

Workload group LOW shows interesting results. Intuitively, workloads composed of only low spatial locality benchmarks would suffer performance degradation with bank partitioning due to reduced bank-level parallelism. However, benchmarks L1 and L3 show a healthy 5% throughput improvement from bank partitioning. This is because bank partitioning load-balances among banks. As most of the requests are row-buffer misses occupying a bank for a long time waiting for additional precharge and activate commands, the banks are highly utilized. Therefore, all the bank-level parallelism the DRAM system can offer is already in use and bank load-balancing becomes critical for the DRAM system throughput. Figure 8 shows the average queueing delay of requests at the memory controller scheduling buffer. When banks are partitioned among low spatial locality threads, the requests are evenly distributed among banks, lowering the average queueing delay.

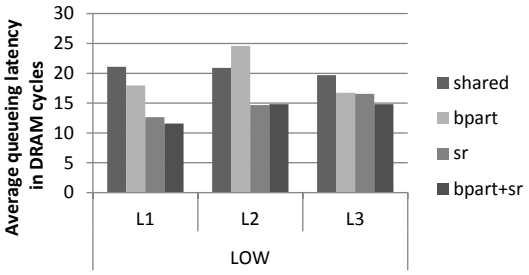


Figure 8: Average queueing delay of requests for workload group LOW

Workloads in group LOW_BW contain non-memory-intensive benchmarks, thus the collective impact on throughput is smaller but has a similar trend. Workload LB4 is an outlier that shows a 7.6% throughput drop from bank partitioning. It consists of lbm, mcf, bzip2, and sjeng, where lbm and mcf are ill-affected by bank partitioning as shown in Figure 6; bzip2 and sjeng are non-memory-intensive and, thus, remain unaffected. Altogether, no improvement results from bank partitioning for this workload. However, memory sub-ranking recovers some of the lost bank-level parallelism and performance. Bank partitioning combined with sub-ranking gives average throughput improvements of 9.8%, 7.4%, 4.2%, and 2.4% for HIGH, MIX, LOW, and LOW_BW, respectively. The maximum throughput improvement for each group is 12.2%, 11.4%, 4.5%, and 5%.

Workload throughput alone only shows half of the story. Figure 9 shows the minimum speedup of workloads. Although bank partitioning alone could provide almost all of the system throughput improvements, it does so at the expense of benchmarks with low spatial locality. Workloads in group MIX and LB4 and LB5 in group LOW_BW show a significant reduction in minimum speedup when bank partitioning is employed. In all such cases, the benchmark which suffers the lowest speedup is either mcf or omnetpp; both benchmarks have low spatial locality. By partitioning banks, these programs suffer from reduced bank-level parallelism. When sub-ranking is employed along with bank

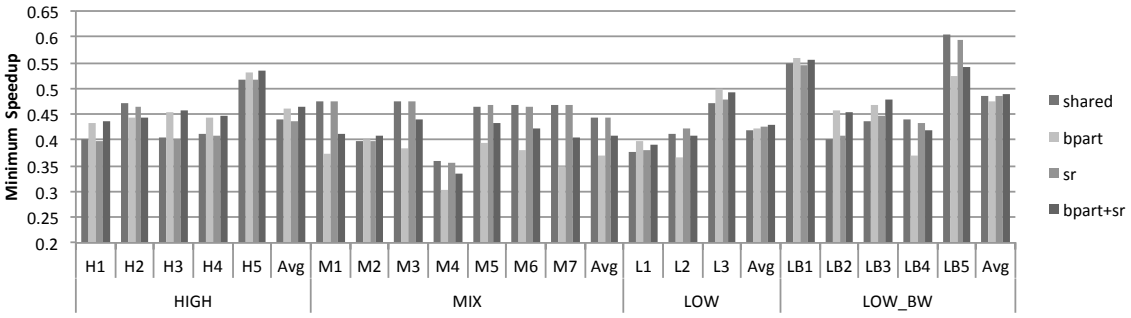


Figure 9: Minimum speedup of workloads.

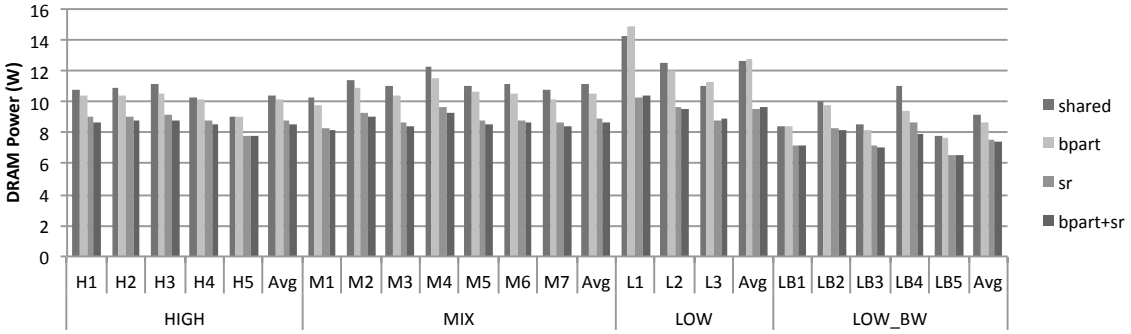


Figure 10: DRAM power consumption.

partitioning, the minimum speedup recovers from this drop in addition to providing system throughput improvements. Bank partitioning and memory sub-ranking optimize conflicting requirements, locality and parallelism, and employ them together results in a more robust and fairer solution.

5.3 Power and System Efficiency

As described in Section 4, we estimate the system power by modeling the DRAM power and the power consumed by the processor. Figure 10 shows the DRAM power component. Although the DRAM power reduction due to bank partitioning only appears to be modest, the actual energy reduction is significantly greater since bank partitioning greatly reduces activations by improving the row-buffer hit rates, and thus performance, as shown in Figures 6 and 7. Sub-ranking reduces DRAM power consumption by only activating a fraction of the rows. Naturally, bank partitioning and sub-ranking, combined, brings additive benefits. With both schemes together, the DRAM power consumption is reduced dramatically, averaging at a 21.4% reduction across all benchmark mixes.

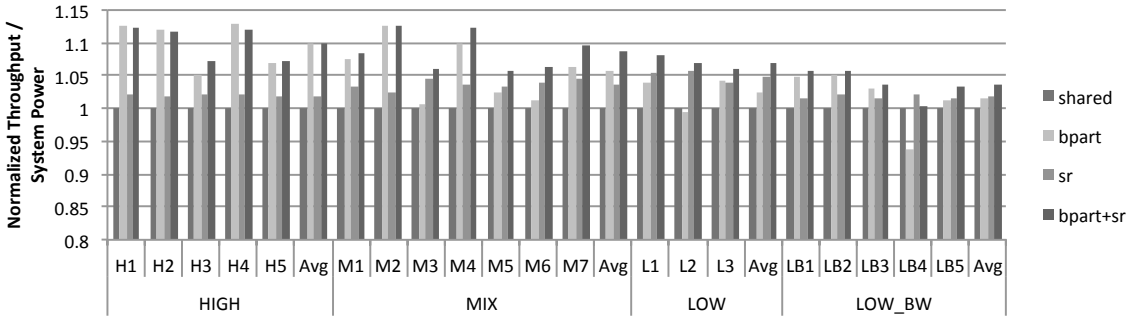
System efficiency is measured by throughput (WS) per system power (Watts), where system power includes DRAM and CPU power. Since the dynamic power consumption of the CPU is modeled as a function of IPC (Section 4), the CPU power slightly increases with the improved IPC. However, the combined system power (DRAM+CPU) remains fairly constant across benchmark mixes.

Figure 11(a) shows the normalized throughput per system power. In group `HIGH`, most of the benefit comes from bank partitioning alone. As mentioned earlier, this group has mixes of benchmarks that all have high row-buffer hit rates and does not benefit much by sub-ranking. On average, the improvement in system efficiency is 10%. In group `MIX`, the overall system efficiency is best enhanced by the synergy of bank partitioning and sub-ranking. This is the result of the throughput improvement from bank partitioning combined with the dramatic power savings from sub-ranking. Efficiency improves by 9% on average, whereas bank partitioning and sub-ranking alone improve efficiency only 6% and 4%, respectively. In group `LOW_BW`, although the improvement is relatively lower, using bank partitioning and sub-ranking together still brings the best benefit. Workload LB4 again shows degraded power efficiency with bank partitioning because of its impact on throughput for this workload.

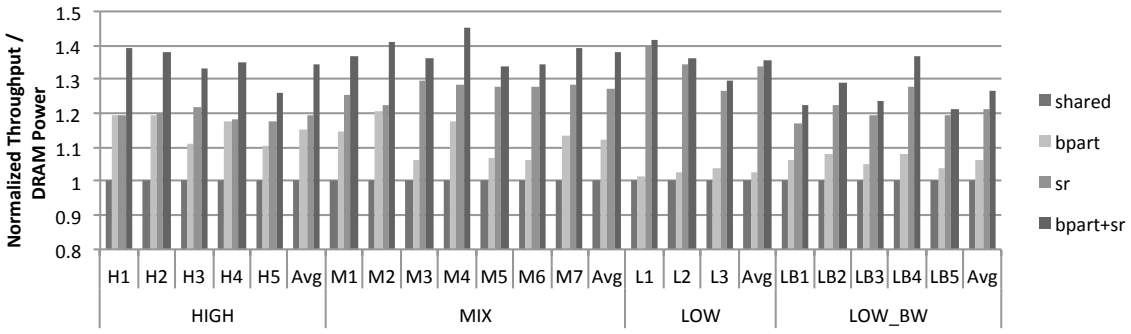
Current processor and memory scaling trends indicate that DRAM power is a significant factor in system power, especially in servers with huge DRAM capacities [6]. Such systems will see greater improvement in system efficiency with combined bank partitioning and sub-ranking. Figure 11(b) verifies this trend with the normalized throughput per DRAM power. Only taking DRAM power into account, bank partitioning with sub-ranking shows improvements of up to 45% over the baseline.

5.4 Bank limited systems

Processor and memory scaling trends also indicate that CMP core count increases faster than available memory banks in the system, especially for embedded or many-core



(a) Normalized throughput per system power.



(b) Normalized throughput per DRAM power.

Figure 11: System efficiency of two-rank systems.

systems. To approximate a system with limited memory banks per core, we conduct experiments on a system with a single rank per channel instead of the two ranks previously shown. Figure 12(a) illustrates the system efficiency for this bank-limited machine. The results show that the bank-limited system enjoys greater benefit from bank partitioning with sub-ranking. The average improvements in system efficiency on the bank-limited system are 18%, 19%, 25%, and 7% for groups `HIGH`, `MIX`, `LOW` and `LOW_BW`, respectively. Figure 12(b) shows the system efficiency per DRAM power for a single rank system; combining bank partitioning and sub-ranking again provides significant improvements.

6 Related Work

6.1 DRAM bank partitioning

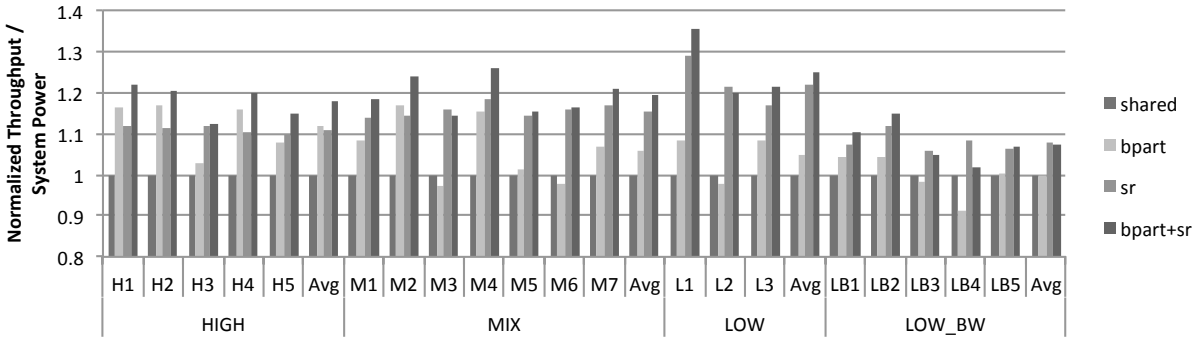
DRAM bank partitioning is first proposed by Mi et al. [13], who use page coloring and XOR cache mapping to reduce inter-thread interference in chip multiprocessors. They develop a cost model and search the entire color assignment space for each workload off-line. While this approach can approximate a near-optimal color assignment for a specific memory system and workload mix, the cost of this search is prohibitive for dynamically changing workload mixes. Also, Mi et al. do not consider the impact of reduced bank-level parallelism; rather, they assume a very large number of banks, exceeding the practical limit of modern DRAM systems. The number of banks can be increased by adding more ranks to a system, but rank-to-rank switching penalties can limit performance as we show in Figure 5. We propose a sub-ranking-based alternative which is both cost-effective as well as power efficient, and provide in-depth analysis of the interaction between locality and parallelism.

Another existing approach by Muralidhara et al. partitions memory channels instead of banks [15]. Their partitioning method is based on the runtime profiling of application memory access patterns combined with a memory scheduling policy to improve system throughput. However, they do not account for or analyze the impact of reduced bank-level parallelism. Also, their method interferes with fine-grained DRAM channel interleaving and limits the peak memory bandwidth of individual applications. To the best of our knowledge, our work is the first to preserve both spatial locality and bank-level parallelism.

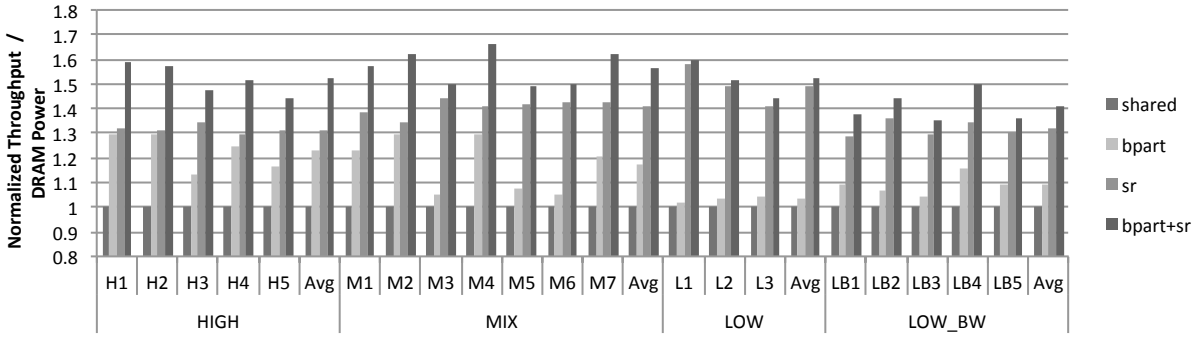
6.2 Memory scheduling policies

A number of memory scheduling policies have recently been proposed that aim at improving fairness and system throughput in shared-memory CMP systems.

Computer-network-based fair queuing algorithms using virtual start and finish times are proposed in [18] and [19], respectively, to provide QoS to each thread. STFM [16] makes scheduling decisions based on the stall time of each thread to achieve fairness. A parallelism-aware batch scheduling method is introduced in [17] to achieve a balance between fairness and throughput. ATLAS [9] prioritizes threads that attained the least service from memory controllers to maximize system throughput at the cost of fairness. TCM [10] divides threads into two separate



(a) Normalized throughput per system power



(b) Normalized throughput per DRAM Power

Figure 12: System efficiency of one-rank systems.

clusters and employs a different memory request scheduling policies to each cluster. The authors claim that both throughput and fairness are enhanced with the proposed scheme.

These previous works address the same problem of inter-thread interference in shared memory, but focus on improving the scheduling of shared resources to threads considering their memory access behavior and system fairness. Our approach is orthogonal to the existing scheduling policy work, as it focuses on improving efficiency by preserving row-buffer locality, and on balancing parallelism with access latency. Bank partitioning can benefit from better scheduling, and better scheduling policies can benefit from improved row-buffer spatial locality.

6.3 Sub-ranking

Recently, several schemes to reduce the active power of DRAM have been proposed. Sub-ranking is a technique that breaks the conventional wide (64-bit) rank into narrower (8-bit, 16-bit or 32-bit) sub-ranks. As each access involves fewer DRAM chips, it reduces the power consumption significantly.

Ware and Hampel [24] propose the Threaded Memory Module, in which the memory controller itself controls individual sub-ranks with chip-enable signals. They analytically present the potential performance improvement due to the effective increase in the number of memory banks. Zheng et al. [27] propose Mini-Rank memory. They keep the memory controller module interface the same, and access sub-ranks using a module-side controller. They focused on savings in activate power from sub-ranking. Ahn et al. [3, 2] propose MC-DIMM. Their work is similar to the Threaded Memory Module, except demultiplexing and the repetitive issue of sub-rank commands are handled on the module side. Also, Ahn et al. empirically evaluate MC-DIMM on multi-threaded applications, with full-system power evaluation. Their target is a heavily threaded system which can naturally tolerate the additional latency from narrow sub-ranks. Unlike the previous works, we focus on the additional bank-level parallelism that sub-ranking can provide in the context of latency sensitive, general purpose out-of-order processor systems.

7 Conclusion

In this paper, we present a mechanism that can fundamentally solve the problem of interleaved memory access streams at its root. Our technique balances the conflicting needs of locality and parallelism. We rely on bank partitioning using OS-controlled coloring to assign a set of independent banks to each potentially conflicting thread. To compensate for the reduced bank parallelism available to each thread we use DRAM sub-ranking to effectively increase the number of banks in the system without increasing its cost. This combined bank partitioning and sub-ranking technique enables us to significantly boost performance and efficiency simultaneously, while maintaining fairness. Our evaluation demonstrates the potential of this technique for

improving locality, which leads to performance and efficiency gains. Moreover, we show that our technique is even more important when considering system configurations that inherently have a small number of banks per thread. We expect future systems to have such bank-constrained configurations because of the relatively high cost of increasing memory banks compared to the cost of increasing the number of concurrent threads in the CPU.

Bank partitioning works best when all applications in a multi-programmed workload have high spatial locality. In such mixes, partitioning eliminates interference and improves performance and power efficiency by $10 - 15\%$. When applications with low spatial locality, which require a larger number of banks to achieve best performance, are added to the application mix then combining partitioning with sub-ranking yields superior results. The combination yields execution throughput improvements (weighted speedup) of 7% and 5% on average for mixed workloads and workloads consisting of only low spatial locality applications respectively. Our combined technique even slightly improves the performance and efficiency of applications that rarely access memory. The importance of combining partitioning with sub-ranking is more apparent when fairness and power efficiency are considered. The minimum speedup (a fairness metric) improves by 15% with the combined technique. In addition, our experiments show that system power efficiency improves slightly more than the measured improvement in system throughput.

The previously given power efficiency improvements are shown for systems that are dominated by CPU power, in which DRAM power is only 20% or less. When looking just at DRAM power, power efficiency improvements are $20-45\%$ for the combined technique as opposed to roughly $10-30\%$ when the techniques are applied separately. For a bank-constrained configuration, improvements are roughly 50% better, relative to the whole-system measurements.

References

[1] Calculating memory system power for DDR3. Technical Report TN-41-01, Micron Technology, 2007.

[2] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Future scaling of processor-memory interfaces. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC)*, Nov. 2009.

[3] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi. Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *IEEE Computer Architecture Letters*, 8(1):5–8, Jan. - Jun. 2009.

[4] T. M. Brewer. Instruction set innovations for the Convey HC-1 computer. *IEEE Micro*, 30(2):70–79, 2010.

[5] G. Hamerly, E. Perelman, J. Lau, and B. Calder. SimPoint 3.0: Faster and more flexible program analysis. In *Proc. the Workshop on Modeling, Benchmarking and Simulation (MoBS)*, Jun. 2005.

[6] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[7] M. K. Jeong, D. H. Yoon, and M. Erez. DrSim: A platform for flexible DRAM system research. `http://lph.ece.utexas.edu/public/DrSim`.

[8] T. Karkhanis and J. E. Smith. A day in the life of a data cache miss. In *Workshop on Memory Performance Issues*, 2002.

[9] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *Proc. the 16th Int'l Symp. High-Performance Computer Architecture (HPCA)*, Jan 2010.

[10] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *Proc. the 43rd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2010.

[11] G. H. Loh, S. Subramaniam, and Y. Xie. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *Proc. the Int'l Symp. Performance Analysis of Software and Systems (ISPASS)*, Apr. 2009.

[12] M. McTague and H. David. Fully buffered DIMM (FB-DIMM) design considerations. Intel Developer Forum (IDF), Feb. 2004.

[13] W. Mi, X. Feng, J. Xue, and Y. Jia. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors. In *Proc. the 2010 IFIP Int'l Conf. Network and Parallel Computing (NPC)*, Sep. 2010.

[14] Micron Corp. *Micron 2 Gb ×4, ×8, ×16, DDR3 SDRAM: MT41J512M4, MT41J256M4, and MT41J128M16*, 2011.

[15] S. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *Proc. the 44th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2011.

[16] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Proc. the 40th IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2007.

[17] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *Proc. the 35h Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[18] K. Nesbit, N. Aggarwal, J. Laudon, and J. Smith. Fair queuing memory systems. In *Proc. the 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2006.

[19] N. Rafique, W. Lim, and M. Thottethodi. Effective management of DRAM bandwidth in multicore processors. In *Proc. the 16th Int'l Conf. on Parallel Architecture and Compilation Techniques (PAC)*, Sep. 2007.

[20] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens. Memory access scheduling. In *Proc. the 27th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2000.

[21] A. Snavely and D. M. Tullsen. Symbiotic job scheduling for a simultaneous multithreaded processor. In *Proc. the 9th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Nov. 2000.

[22] Standard Performance Evaluation Corporation. SPEC CPU 2006. `http://www.spec.org/cpu2006/`, 2006.

[23] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM design and organization for energy-constrained multi-cores. In *Proc. the 37th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2010.

[24] F. A. Ware and C. Hampel. Improving power and data efficiency with threaded memory modules. In *Proc. Int'l Conf. Computer Design (ICCD)*, Oct. 2006.

[25] D. H. Yoon, M. K. Jeong, and M. Erez. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *Proc. the 38th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2011.

[26] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proc. the 33rd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2000.

[27] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *Proc. the 41st IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Nov. 2008.