Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era

Dimitris Kaseridis
Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
kaseridis@mail.utexas.edu

Jeffrey Stuecheli
IBM Corp. & Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
jeffas@us.ibm.com

Lizy Kurian John
Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
ljohn@ece.utexas.edu

ABSTRACT

Contemporary DRAM systems have maintained impressive scaling by managing a careful balance between performance, power, and storage density. In achieving these goals, a significant sacrifice has been made in DRAM's operational complexity. To realize good performance, systems must properly manage the significant number of structural and timing restrictions of the DRAM devices. DRAM's use is further complicated in many-core systems where the memory interface is shared among multiple cores/threads competing for memory bandwidth.

The use of the "Page-mode" feature of DRAM devices can mitigate many DRAM constraints. Current open-page policies attempt to garner the highest level of page hits. In an effort to achieve this, such greedy schemes map sequential address sequences to a single DRAM resource. This non-uniform resource usage pattern introduces high levels of conflict when multiple workloads in a many-core system map to the same set of resources.

In this paper we present a scheme that provides a careful balance between the benefits (increased performance and decreased power), and the detractors (unfairness) of page-mode accesses. In our *Minimalist* approach, we target "just enough" page-mode accesses to garner page-mode benefits, avoiding system unfairness. We use a fair memory hashing scheme to control the maximum number of page mode hits, and direct the memory scheduler with processor-generated prefetch meta-data.

Categories and Subject Descriptors

B.3.1 [**Memory Structures**]: Semiconductor Memories—*Dynamic memory (DRAM)*; B.3.2 [**Memory Structures**]: Design styles—*cache memories, Primary Memories, Shared Memory, Interleaved Memories*

MICRO 44, December 3-7, 2011, Porto Alegre, Brazil

General Terms

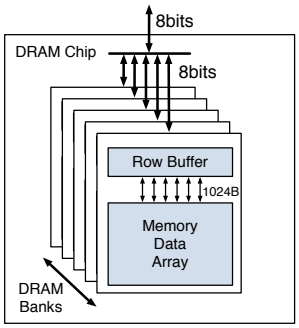
Design, Performance

1. INTRODUCTION

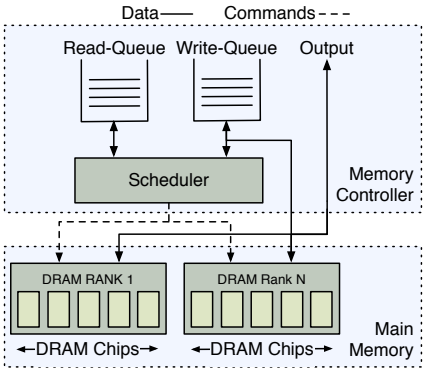
Since its invention, the DRAM memory subsystem has proven to be one of the most important system components. The requirement of a properly designed memory subsystem is further amplified in the case of chip-multiprocessors where memory is shared among multiple, concurrently executing threads. As the memory interface is shared among a growing number of cores, providing both sustained system throughput and thread execution fairness is equally critical. To do so, the memory controller must be able to make an intelligent selection of requests to execute at any given point. This selection must carefully balance thread execution speed and overall throughput, functioning well across a broad range of memory utilization levels.

To provide an efficient balance between memory density, request latency, and energy consumption, DRAM designers have adopted a complex architecture that imposes a number of structural and timing limitations. One of the most important components of this system is the *Row Buffer* (shown in Figure 1(a)). The row buffer serves two primary purposes. It acts as an interface between the narrow external pins and the much wider internal buffer structure width. Additionally, the row buffer captures the full data width of the destructive DRAM read access, such that it can be restored at the completion of the operation. The row buffer can service multiple data transfers from much wider DRAM cell access. These row buffer or "page-mode" accesses can effectively amortize the high cost of the DRAM cell reads across multiple data transfers, improving system performance and reducing DRAM power consumption.

Current implementations and proposals tend to be grouped into two classes with respect to row buffer usage. Leaving a row buffer open after every access (*Open-page* policy) enables more efficient access to the open row, at the expense of increased access delay to other rows in the same DRAM array. *Open-page* policies attempt to gather multiple requests into each row buffer access by speculatively delaying the precharge operation in an effort to execute additional row reads. This enables latency, scheduling, and power improvements possible with page-mode accesses. However, as these policies are applied to the numerous request streams of a many-core system, priority is given to accesses to already opened pages, introducing memory request priority inversion



(a) DRAM Bank structure



(b) Memory controller and organization

Figure 1: Memory Organization

and potential thread fairness/starvation problems [16, 18, 6, 7]. *Closed-page* policies avoid the complexities of row buffer management by issuing a single access for each row activation. This class of policies provide a consistent fair latency at the expense of potential page-mode gains.

In this paper, we propose a page-mode policy with a combination of open/closed properties based on two important observations. First, page-mode gains, such as power and scheduling conflict reduction, can be realized with a relatively small number of page accesses for each activation. Based on this insight, we adapt the DRAM address mapping scheme to target this small number of hits, which enables more uniform bank utilization and prevents thread starvation in cases of conflict. Secondly, we show page-mode hits exploit spatial reference locality, of which the majority can be captured in modern prefetch engines. Therefore, the prefetch engine can be used to explicitly direct page-mode operations in the memory scheduler. With this, the system can essentially guarantee that the target page hit rate will be met, irrespective of conflicts between competing execution threads. This insight motivates a less aggressive open-page policy that can effectively eliminate the negative aspects of page-mode, such as starvation and row buffer conflicts, while maintaining most of its gains. We refer to our policy as "Minimalist", in that we seek to achieve only the necessary number of page-mode hits, and nothing more. Overall in this work we make the following contributions:

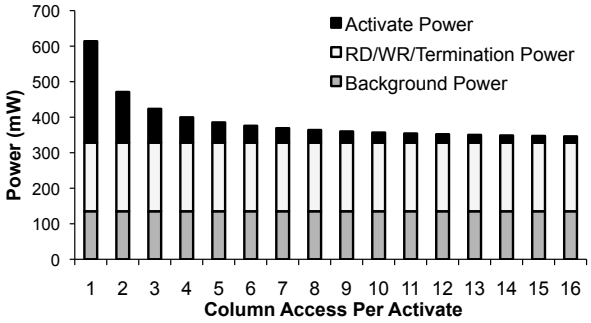
1. Recognize page-mode access gains can be realized with only a small number of accesses per activation (four with DDR3 memory).
2. Propose a fair DRAM address mapping scheme that prevents row-buffer locality starvation and increases bank level parallelism.
3. Identify that most of the memory operations with "page-mode" opportunities are the results of memory accesses generated through prefetch operations. Based on this insight, we direct the memory controller page-mode control with prefetch meta-data to enforce a specific number of page-mode hits.
4. Propose an intuitive criticality-based memory request priority scheme where demand read and prefetch operations are prioritized based on the latency sensitivity of each operation.

2. BACKGROUND

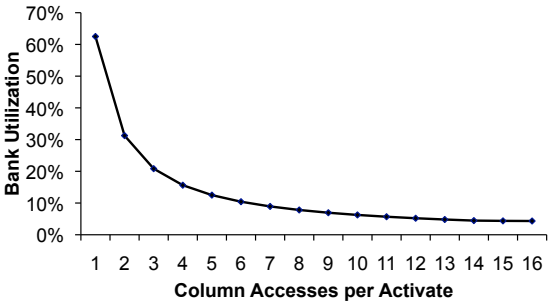
A typical memory controller organization is shown in Figure 1(b). DRAM chips are optimized for cost, meaning that technology, cell, array, and periphery decisions are made giving higher priority to bit-density. This results in devices and circuits which are slower than standard logic, and chips that are more sensitive to noise and voltage drops. A complex set of timing constraints has been developed to mitigate each of these factors for standardized DRAMs, such as outlined in the JEDEC DDR3 standard [4]. These timing constraints result in "dead times" before and after each random access; the processor memory controller's job is to reduce these performance-limiting gaps through exploitation of parallelism.

While DRAM devices output only 16-64 bits per request (depending on the DRAM type and burst settings), internally, the devices operate on much larger, typically 1KB pages (also referred to as *rows*). As shown in Figure 1(a), each DRAM array access causes all 1KB of a page to be read into an internal array called *Row Buffer*, followed by a "*column*" access to the requested sub-block of data. Since the read latency and power overhead of the DRAM cell array access have already been paid, accessing multiple columns of that page decreases both the latency and power of subsequent accesses. These successive accesses are said to be performed in *page-mode* and the memory requests that are serviced by an already opened page loaded in the row buffer are characterized as *page hits*.

Two of the most important DRAM timing parameters that introduce a number of important timing complexities in DRAM's operation are the tRC and tRP parameters. tRC represents the minimum delay between back-to-back activations of two different rows within the same bank (50ns in recent DDRx DRAM specifications [4]). As a result, when a memory controller activates a page, it needs to wait for at least tRC time before activating a new page in the same DRAM bank. When multiple threads access different rows in the same bank, tRC delay can potentially introduce a significant latency overhead. tRP delay is known as the precharge delay, and represent the delay between issuing a precharge command and actually activating the new page on the row buffer. In an open-page policy, the memory controller keeps pages open in anticipation of additional page hits. In such case, when there is a page conflict, tRP is the penalty paid to "close" the current page before a new one is opened.



(a) DRAM power for each 2GBit DDR3 1333 Mhz chip at 40% read, 20% write utilization [14]



(b) Bank utilization for a 2-Rank 1333 Mhz system at 60% data bus utilization

Figure 2: Improvements for increased access per activation.

3. MOTIVATION

Due to the reductions in both latency and energy consumption possible with page-mode, techniques to aggressively target page-mode operations are often used. There are downsides however, which must be addressed. In this section we explore the nature of page-mode benefits, highlighting the diminishing returns as the number of page-mode accesses for each row buffer activation increases.

1. **Latency Effects:** A read access to an idle DDRx DRAM device has a latency of approximately 25ns. An access to an already opened page reduces this latency in half to 12.5ns. Conversely, the accesses to different rows in the DRAM bank can result in increased latency. Overall, increases in latency are caused by two mechanisms in the DRAM. First, if a *row* is left open in an effort to service page hits, to service a request to another page incurs a delay of 12.5ns to close the current page followed by the latency to open and access the new page. Secondly, the aforementioned tRC delay has remained approximately 50ns across the most recent DDRx DRAM devices. In a system that attempts to exploit page-mode accesses, the overall effect on loaded memory latency and program execution speed due to the combination of these properties can significantly increase the observable latency.
2. **Power Reduction:** Page mode accesses reduce DRAM power consumption by amortizing the activation power associated with reading the DRAM cell data and storing them into the row buffer. Figure 2(a) shows the DRAM power consumption of a 2GBit DDR3 1333MHz DRAM as the number of row accesses increases. This follows the power corollary of Amdahl's law, where page-mode only reduces the page activation power component. DRAM power quickly becomes dominated by the data transfer and background (not proportional to bandwidth) power components.
3. **Bank Utilization:** The utilization of the DRAM banks can be a critical parameter in achieving high scheduling efficiency. If bank utilization is high, the probability that a new request will conflict with a busy bank is greater. As the time to activate and precharge the array overshadows data bus transfer time, having available banks is often more critical than having available data bus slots. Increasing the data transferred with each DRAM activate, through page-mode, amortizes the expensive DRAM bank access, reducing utilization. Figure 2(b) shows the bank utilization of a DDR3 1333 MHz system, with two sets of devices (*ranks*) sharing a data bus at 60% bus utilization. A closed-page policy, with one access per activate would produce an unreasonably high bank utilization of 62%. However, the utilization drops off quickly as the accesses per activate increases. For example four accesses per activate reduces the bank utilization to 16%, greatly reducing the probability that a new request will be delayed behind a busy bank.
4. **Other DRAM Complexities:** Beyond the first order effects described above, more subtle DRAM timing rules can have significant effects of DRAM utilization, especially as the data transfer clock rates increase in every DRAM generation. Many DRAM parameters do not scale with frequency due to either constant circuit delays and/or available device power. One example is the tFAW parameter. tFAW specifies the maximum number of activations in a rolling time window in order to limit peak instantaneous current delivery to the device. Specifically for the case of 1333MHz DDR3 1KB page size devices, the tFAW parameter specifies a maximum of four activations every 30ns. A transfer of a 64 byte cache block requires 3ns, thus for a single transfer per activation tFAW limits peak utilization to 80% ($6ns * 4/30ns$). However, with

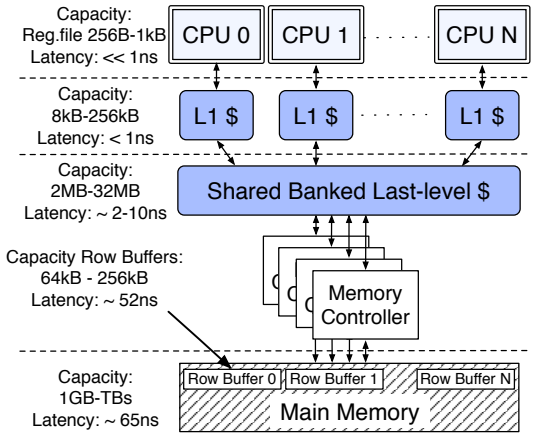


Figure 3: Capacities and latencies of memory

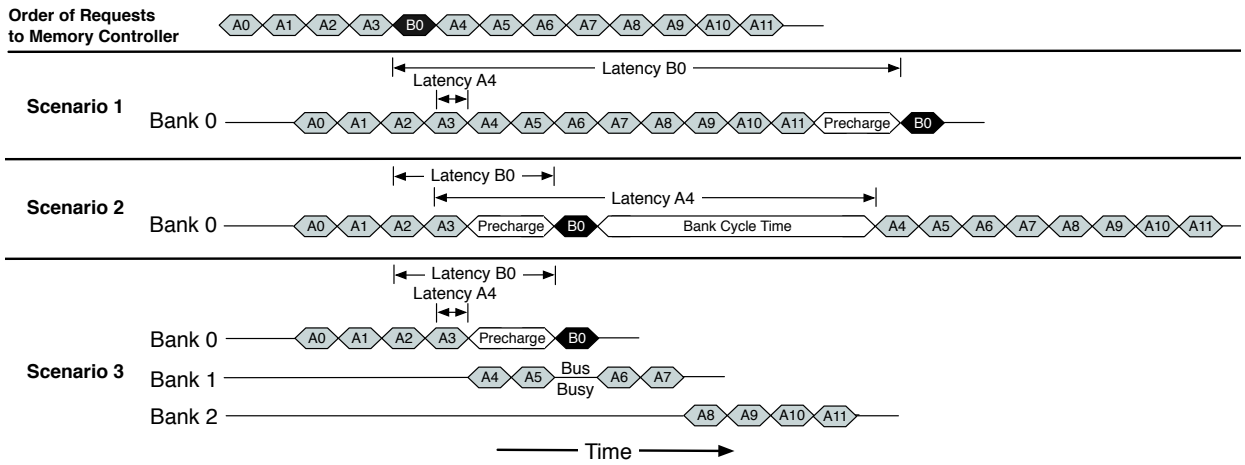


Figure 4: Row buffer policy examples. Scenario 1: System gives higher priority to workload A. Scenario 2: System gives higher priority to workload B, interrupting workload A. Scenario 3: Minimalist approach, sequential accesses are executed as 4 blocks per row buffer followed by switching to next memory bank.

only two accesses per activation, tFAW has no effect ($12ns * 4/30ns > 1$). The same trend is observed across several other DRAM parameters, where a single access per activation results in efficiency degradation, while a small number of accesses alleviates the restriction.

In summary, based on the above trends, we found that a relatively small number of accesses to a page to be very effective in taking advantage of DRAM page-mode for both scheduling and power efficiency. For example, at four row accesses per activation, power and bank utilization are 80% of their ideal values. Latency effects are more complex, as scheduling policies to increase page hits introduce bank precharge penalties (through speculatively delaying closing a bank), making raw latency reductions difficult to achieve. These effects are described in the following sections.

3.1 Row Buffer Locality in Modern Processors

In this section we describe our observations regarding page-mode accesses as seen in current workstation/server class designs. Contemporary CMP processor designs have evolved to impressive systems on a chip. Many high performance processors (eight in current leading edge designs) are backed by large last-level caches containing up to 32 MB of capacity [5]. A typical memory hierarchy that includes the DRAM row buffer is shown in Figure 3. As a large last-level cache filters out requests to the memory, row buffers inherently exploit only spatial locality. Applications' temporal locality results in hits to the much larger last-level cache.

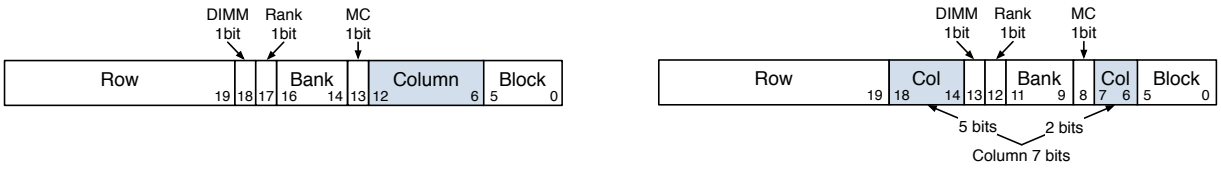
Access patterns with high levels of spatial locality, which miss in the large last level cache, are often very predictable. In general, speculative execution and prefetch algorithms can be exploited to generate memory requests with spatial locality in dense access sequences. Consequently, the latency benefit of page-mode is diminished.

3.2 Bank and Row Buffer Locality Interplay With Address Mapping

The mapping of the real memory address into the DRAM device address (row, column, bank) has a very significant contribution into memory system behavior. Mapping the spatial locality of request streams to memory resources is the dominant concern. Commonly used open-page address mapping schemes map all DRAM column address bits to the low order real address directory above the cache offset [19, 11, 18, 6]. This maps each memory page to a sequential region of real memory. With this approach, very common linear access sequences reference all data in the row buffer, minimizing DRAM activates.

As identified by Moscibroda *et al.* [16], this hashing can produce interference between the applications sharing the same DRAM devices, resulting in significant performance loss. The primary problem identified in that work is due to the FR-FCFS [19] policy where page hits have a higher priority than requests with lower page affinity. Beyond fairness, schemes that map long sequential address sequences to the same row, suffer from low bank-level parallelism (BLP). If many workloads with low BLP share a memory controller, it becomes inherently more difficult to interleave the requests, as requests from two workloads mapping to the same DRAM bank will either produce a large number of bank conflicts, or one of them has to stall, waiting for all of the other workload's request to complete, significantly increasing its access latency.

Figure 4 illustrates an example where workload A generates a long sequential access sequence, while workload B issues a single operation mapping to the same DRAM. With a standard open-page policy mapping, both requests map to the same DRAM bank. With this mapping, there are two scheduling options, shown in Scenarios 1 and 2. The system can give workload A higher priority until all page hits are completed, significantly increasing the latency of the workload B request (Scenario 1, Figure 4). Conversely, workload A can be interrupted, resulting in very inefficient activate to activate commands conflict for request $A4$ (Scenario 2, Figure 4), mainly due to the time to load the new page in the row buffer and the unavoidable tRC timing requirement between back-to-back activations of a page in a bank. Neither of these solutions optimize fairness and throughput. In our proposal we adapt the memory hash to convert workloads with high row buffer locality (RBL), into



(a) Typical Mapping

(b) Proposed Mapping

Figure 5: System Address Mappings to DRAM Address - Example system in figure has 2 memory controllers (MC), 2 Ranks per DIMM, 2 DIMMs per Channel, 8 Banks per Rank and 64B Cache-lines.

workloads with high bank-level parallelism. This is shown in Scenario 3 of Figure 4, where sequential memory accesses are executed as reading four cache blocks from each row buffer, followed by switching to the next memory bank. With this mapping, operation B can be serviced without degrading the traffic to workload A.

4. MINIMALIST OPEN-PAGE MODE

We base the *Minimalist Open-page* scheme on the observation that most of the page-mode gains can be realized with a relatively small number of page accesses for each page activation. In addition, address hashing schemes that map sequential regions of memory to a single DRAM page result in poor performance due to high latency conflict cases. The Minimalist policy defines a target number of page hits that enables a careful balance between the benefits (increased performance and decreased power), and the detractors (resource conflicts and starvation) of page-mode accesses. With this scheme several system improvements are accomplished. First, through the address mapping scheme, row buffer starvation is avoided. This alleviates the memory request priority scheme requirements compared to prior approaches that must address row-buffer starvation. In our proposal, the scheduler can focus its priority policy on memory request criticality, which is important in achieving high system throughput and fairness.

Through our analysis we found that most of the memory operations with "page-mode" opportunities are the results of memory accesses generated through prefetch operations. Therefore, we use processor's prefetch engine to provide request meta-data information which directs the scheme's page-mode accesses and request priority scheme. In effect, this enables the prefetch generated page-mode access to be done reliably, with back to back scheduling on the memory bus.

In the remaining of this section we describe the Minimalist policy in detail. First, we describe the address mapping scheme that enables bank-level parallelism with the necessary amount of row-buffer locality. This is followed by the prefetch hardware engine, as this component provides prefetch request priorities and prefetch-directed page-mode operation. Finally, we describe the scheduling scheme for assigning priorities and issuing the memory request to the main memory.

4.1 DRAM Address Mapping Scheme

The differences between a typical mapping and the one we use in our proposal are summarized in Figure 5. The basic difference is that the *Row Column* access bits that are used to select the row buffer columns are split in two places. The first 2 LSB bits (Least Significant Bits) are located right after the *Block* bits to allow the sequential access of up to 4 consequent cache lines in the same page. The rest of the MSB (Most Significant Bits) column bits (five bits in our case if we assume that 128 overall cache lines are stored in every row buffer) are located just before the *Row* bits. Not shown in the figure for clarity, higher order address bits are XOR-ed with the bank bits shown in the figure to produce the actual bank selection bits. This reduces row buffer conflicts as described by Zhang *et al.* [21]. The above combination of bit selection allows workloads, especially streaming, to distribute their accesses to multiple DRAM banks; improving bank-level parallelism and avoiding over-utilization of a small number of banks that leads to thread starvation and priority inversion in multi-core environments.

4.2 Data Prefetch Engine

To harvest the predictable page-mode opportunities, we need to utilize an accurate prefetch engine. The engine targets spatial locality prediction and is able to predict repeatable address strides. To do so, each core includes a hardware prefetcher that is able to detect simple access streams of stride 1, as described by Lee *et al.* [9]. To keep bandwidth overhead low and throttle prefetch aggressiveness, our prefetcher uses a prefetch depth distance predictor to decide how far from the currently accessed memory address each access stream should be prefetched. To avoid prematurely fetched data, the prefetcher "ramps up" gradually to the full depth only when there is confidence that an access stream is a useful, long access stream. To dynamically decide on the maximum depth of each access stream we utilize a structure based on the "Adaptive Stream Detection" (ASD) prefetcher from Hur *et al.* [2]. More specifically, we used the "Stream Length Histograms" (SHL) from ASD to decide on the depth of each access stream. Finally, to avoid polluting the caches with unused prefetches, all the prefetched requests are stored in the LRU position of our last-level cache until used by executing instructions.

4.2.1 Multi-line Prefetch Requests

Although the presented prefetcher makes decisions on the granularity of a single cache-line, we utilize multi-line prefetch operations in our scheme. A multi-line prefetch operation consists of a single request sent to the memory controller, to indicate a specific sequence of cache-lines to be read from a memory page. This policy was initially introduced in the IBM POWER6 design [9]. Multi-line operations reduce the command bandwidth and queue resource usage. Specifically for our scheme, multi-line operations can consolidate the accesses to a DRAM in a controlled

burst. That enables the issue of a single request to the memory controller queue; processing of multi-line requests that are directed to the same DRAM page together as a single request in the queue; and, in the end, close the page after completing all of the prefetches. Consequently, multi-line requests: a) improve bandwidth use, b) simplify our priority scheme by executing back-to-back page-related requests, and c) improve our controller efficiency for closing the DRAM pages.

4.3 Memory Request Queue Scheduling Scheme

Previous priority-based open-page scheduling proposals either exhibit unfairness [19], or use request priority as a fairness enforcement mechanism [18, 6, 7]. For example, the ATLAS scheme [6] assigns the same priority to all of the requests of a thread in the memory queue based on attained service, assuming all requests from a thread are equally important. We found that in out-of-order execution, the importance of each request can vary both between and within applications. This range from requests that are critical for the performance (*e.g.* demand-misses) to requests that can tolerate more latency, such as misses in applications exhibiting high levels of Memory-level Parallelism (MLP) and prefetch requests. As we solve fairness (row buffer starvation) through the address mapping scheme, priority is directed with each memory request's instantaneous priority, based on both the current MLP and metrics available within the prefetch engine.

4.3.1 DRAM Memory Requests Priority Calculation

In our memory scheduling priority scheme we assign a different priority to every memory request based on its criticality to performance. We separate the requests in two categories: a) *Read requests* (normal), and b) *Prefetches*. Based on each request's category and its criticality to performance, the memory controller assigns to them an initial priority. The priorities assigned for every case are shown in Table 1. To improve fairness and avoid starvations, we implemented a *time-based dynamic priority scheme*. The scheme assigns an initial priority to every request and as a request remains in the queue waiting to be serviced, its priority is gradually increased. We use a priority value between 0 and 7 for every request. At a time interval of 100ns, each request's priority is increased by one. If the priority of a prefetch request is increased more then the maximum priority for prefetches (priority of 4 in our case), the prefetch request is ignored and removed from the memory controller queue. Intuitively, if a prefetch request remains for a significant amount of time in the queue, it is most likely not a useful prefetch because a demand read request for the same cache line will soon follow and there is not enough time to service the prefetch. A static version of this scheme is also used in Lee *et al.* [10].

Read requests are assigned higher priority than prefetches since the latency of demand misses is highly correlated to performance. We used the Memory-level Parallelism (MLP) information of the core that issued each request to identify criticality. The MLP information is directly collected from each core's *Miss Status Holding Registers* (MSHR) [8]. The MSHR tracks of all the outstanding L1 misses being serviced by the lower levels of memory hierarchy (L2 and main memory). As a result, the number of entries in each core's MSHR which indicate an L2 miss represents the current MLP of the application. Low MLP means that there is a small number of L2 outstanding misses and therefore each one is very important for the execution progress of the application. Any delay on serving these misses results in significant performance degradation. As MLP increases, there are more outstanding misses available but, on the average case, most of them do not block the progress of speculative execution on an out-of-order core, making their latency less important for performance. Our algorithm statically assigns priorities by classifying the possible levels of MLP in three categories, as shown in the left part of Table 1.

Table 1: Memory read requests priority assignment

Normal Requests		Prefetch Requests	
MLP level	**Priority (3bits)**	**Distance from Head**	**Priority (3bits)**
MLP < 2 (Low MLP)	7	$0 \leqslant$ Distance < 4	4
$2 \leqslant$ MLP < 4 (Medium MLP)	6	$4 \leqslant$ Distance < 8	3
MLP $\geqslant 4$ (High MLP)	5	$8 \leqslant$ Distance < 12	2
		$12 \leqslant$ Distance < 16	1
		Distance $\geqslant 16$	0

Prefetch requests are assigned a priority level lower than normal requests using prefetch meta-data information sent by our prefetch engine. Their priority is based on the distance in cache blocks from the actual consuming instructions to the prefetch request. Requests with a small distance have higher priority, since they are more likely to be used in the near future. As the distance from the head of the stream increases, the prefetch's latency is less critical for performance. The priorities based on this distance are shown in the right part of Table 1. Finally, if a normal read request arrives in the memory queue and there is already a prefetch request in the queue waiting to be serviced, our priority scheme upgrades the prefetch request to a normal read request, assigning to it a priority based of the application's MLP information, as for the case of normal read requests.

4.3.2 DRAM Page Closure (Precharge) Policy

In general, our Minimalist policy does not speculatively leave DRAM pages open. If a multi-line prefetch request is being processed, the page is closed with an auto-precharge sent along with the read command (In DDRx the auto-precharge bit indicates to the DRAM to close the page after the data are accessed [3]). This saves the command bandwidth of an explicit precharge command. For read and single line prefetch operations, the page is left open based on the following principle: the *tRC* DRAM parameter specifies the minimum time between activations to a DRAM bank. The *tRC* is relativity long (50ns) compared to the precharge delay of 12.5ns. Therefore, closing a bank after a single access does not allow a reactivation of a new page on the bank until the *tRC* delay expires. For this reason, we speculatively leave pages open for the *tRC* window, as this provides for a "free" open page interval.

Table 2: Full-system detailed simulation parameters

System Configuration	8 cores CMP, 2 Memory Controllers			
Core Characteristics	**Clock Frequency**	**Pipeline**	**Branch Predictor**	**L1 Data & Inst. Cache**
	4 GHz	30 stages / 4-wide fetch / decode 128 Reorder Buffer Entries 64 Scheduler Entries	Direct YAGS / indirect 256 entries	64 KB, 2-way associative, 3 cycles access time, 64B blocks
H/W Prefetcher	H/W stride n with dynamic depth, 32 streams / core (see Section 4.2 for details)			
Memory Subsystem	**L2 Cache**	**Outstanding Requests**	**Best-case Idle DRAM Memory Latency**	**Memory Bandwidth**
	16 MB, 8-ways associative, 12 cycles bank access, 64B blocks	16 Requests per Core	65ns	21.333 GB/s
	Memory Controller Organization	**Controller Resources**	**DRAM Type**	
	2 Ranks per Controller 8 x 4Gbit DRAM chips per Rank	32 Read Queue & 32 Write Queue Entries	DDR3 1333MHz 8-8-8	

Priority Rules 1 Minimalist Open-page: Request Scheduling Rules in Memory Controller Queue

1. **Higher Priority Request First:** Requests with higher priority are issue first in our per-request scheme
2. **Ready-Requests First:** Requests that belong to the same multiline-prefetch request that is currently being serviced in a open bank are prioritize over other requests that are not "ready" for scheduling yet.
3. **First-Come First-Served:** Older requests issued first.

4.3.3 Overall Memory Requests Scheduling Scheme

The rules in Priority Rules 1 summarize the *per-request* scheduling prioritization scheme that is used in the *Minimalist Open-page* scheme. The same set of rules are used by all of the memory controllers in the system. As explained in Section 3.1, the Minimalist address mapping scheme guarantees memory resource fairness while preventing starvation and priority inversion. Therefore, there is no need for any communication/coordination among the multiple controllers.

Our scheduling scheme is based on assigning priorities to each requests individually based on their criticality to performance. Our first, most important rule, is to schedule requests with the highest priority first. Our second rule, namely "Ready-Requests First", guarantees that between requests with the same priority, requests that are mapped to the same temporally opened page are scheduled first. To clarify, if the controller is servicing the multiple transfers from a multi-line prefetch request, it can be interrupted by a higher priority request (assuming the needed bank is beyond *tRC*). This guarantees that requests that are very critical for performance can be serviced with the smallest latency, enabling our controller to work well in a wide range of memory bus utilization levels.

4.3.4 Handling Write Operations

The dynamic priority scheme only applies to read requests as they directly limit the completion of instructions. While the completion of write requests does not directly affect an application's execution progress, the memory bandwidth consumed by memory writes and their interference with read requests' latency are still important components for performance. To alleviate the pressure of write requests, we follow an approach similar to the *Virtual Write Queue* (VWQ) proposal [20] in the handling of write operations inside the memory controller. The scheme enables read requests to be handled separate from write requests avoiding interactions when possible, while causing minimal intrusion when the VWQ queues become full.

Table 3: DRAM simulation parameters

Name	Description	Value
CL	Column Address Strobe (CAS) Delay	8 mem. cycles
tRCD	Row Address to Column Address Delay	12ns
tRP	Row Precharge time	12ns
tRAS	Row active time	36ns (min.)
tRC	row cycle time	48ns
tRFC	Refresh row cycle time	300ns
tFAW	Four page activates time window	30ns
tREFI	Refresh Interval	7.8μs
tRTP	Read to Precharge delay	7.5ns
tWTR	Write to Read delay	7.5ns
Rank to Rank Trans.	Delay to transition the bus from one Rank to the next	5ns
BL	Burst Length	8 data beats
tRRD	Back-to-back row activations to any bank	7.5ns

5. EVALUATION

To evaluate our scheme, we simulated an 8 core CMP system using the Simics functional model [13] extended with the GEMS toolset [15]. We used an aggressive out-of-

Table 4: Randomly selected 8-core workload sets from SPEC cpu2006 suite.

Exp. #	Workload Sets (Core-0 → Core-7)	% of peak DRAM bandwidth
1	gcc, lbm, lbm, milc, namd, namd, soplex, tonto	45%
2	gcc, gcc, leslie3d, libquantum, mcf, sjeng, zeusmp, zeusmp	47%
3	bzip2, calculix, GemsFDTD, GemsFDTD, leslie3d, namd, omnetpp, wrf	49%
4	astar, gcc, leslie3d, libquantum, namd, soplex, soplex, tonto	51%
5	cactusADM, GemsFDTD, gromacs, lbm, lbm, mcf, sphinx3, wrf	53%
6	bzip2, GemsFDTD, gobmk, h264ref, leslie3d, leslie3d, sphinx3, xalancbmk	56%
7	dealII, leslie3d, mcf, namd, soplex, sphinx3, tonto, xalancbmk	58%
8	bzip2, calculix, lbm, libquantum, milc, namd, soplex, xalancbmk	60%
9	astar, h264ref, libquantum, libquantum, sphinx3, sphinx3, wrf, xalancbmk	62%
10	astar, bzip2, libquantum, omnetpp, sjeng, sjeng, soplex, xalancbmk	64%
11	astar, h264ref, lbm, lbm, libquantum, namd, omnetpp, soplex	66%
12	gamess, lbm, leslie3d, omnetpp, tonto, wrf, xalancbmk, xalancbmk	68%
13	cactusADM, GemsFDTD, hmmer, libquantum, sphinx3, xalancbmk, xalancbmk, zeusmp	70%
14	calculix, lbm, lbm, libquantum, milc, perlbench, soplex, xalancbmk	73%
15	astar, GemsFDTD, lbm, libquantum, libquantum, perlbench, xalancbmk, zeusmp	75%
16	leslie3d, mcf, sjeng, soplex, sphinx3, sphinx3, xalancbmk, xalancbmk	77%
17	cactusADM, gamess, libquantum, libquantum, mcf, milc, omnetpp, soplex	79%
18	cactusADM, lbm, lbm, lbm, libquantum, mcf, perlbench, xalancbmk	81%
19	lbm, mcf, milc, milc, omnetpp, perlbench, soplex, xalancbmk	83%
20	cactusADM, gamess, GemsFDTD, mcf, mcf, omnetpp, omnetpp, xalancbmk	85%
21	bwaves, GemsFDTD, leslie3d, leslie3d, mcf, omnetpp, soplex, xalancbmk	87%
22	cactusADM, dealII, gcc, libquantum, omnetpp, omnetpp, xalancbmk, xalancbmk	89%
23	gamess, lbm, mcf, omnetpp, omnetpp, soplex, sphinx3, xalancbmk	92%
24	gamess, lbm, libquantum, mcf, namd, omnetpp, xalancbmk, xalancbmk	94%
25	lbm, omnetpp, omnetpp, soplex, wrf, xalancbmk, xalancbmk, zeusmp	96%
26	bwaves, gromacs, libquantum, mcf, omnetpp, omnetpp, soplex, xalancbmk	98%
27	bwaves, GemsFDTD, libquantum, libquantum, libquantum, namd, omnetpp, xalancbmk	100%

order processor model from GEMS along with an in-house detailed memory subsystem model. In addition, we modified GEMS to add the hardware prefetching engine described in Section 4.2. Our memory controller model simulates a detailed DDR3 1333MHz DRAM using the appropriate memory controller policy for each experiment. Table 2 summarizes the full-system simulation parameters used in our study, while Table 3 includes the most important DDR3 parameters that we modeled in our toolset.

For our evaluation we utilize a set of multi-programmed workload mixes from the SPEC cpu2006 suite [1]. We randomly selected 27, 8-core benchmark mixes spanning from medium bus utilization levels to saturation. To accomplish this we summed the single core bandwidth requirements of a large number of randomly selected workloads. We then selected 27 sets by choosing the total bandwidth target to span for 45% to 100%. The sets are ordered from lower to higher bus utilization. The bandwidth range was selected to evaluate the proposal as the system transitions from medium load into saturation. Table 4 summarizes the selected workload sets along with their peak bandwidth use. Note, DDR DRAM timing constrains limit sustained utilization to ≈70%, thus 100% of the peak bandwidth produces a system beyond memory saturation.

For the evaluation, we fast-forwarded each experiment to its most representative execution phase; use the next 100M instructions to warm up the caches and memory controller structures; and then simulate the set until the slower benchmark completes 100M instructions. We only use the statistics gathered for the representative 100M instruction phase after the warming up period. For each experiment we present as a speed-up estimation the weighted speedup:

$$Throughput\_Speedup = \sum \left(\frac{IPC_i}{IPC_{i,FR-FCFS}} \right) \quad (1)$$

where $IPC_{i,FR-FCFS}$ is the IPC of the i-th application measured in our FR-FCFS baseline system using an open-page policy memory controller. In addition, to estimate the execution fairness of every proposal, we utilize the harmonic mean of weighted speedup, as was previously suggested by Luo *et al.* [12]:

$$Fairness = \frac{N}{\sum \left(\frac{IPC_{i,alone}}{IPC_i} \right)} \quad (2)$$

where $IPC_{i,alone}$ is the IPC of the i-th application when it was executed standalone on the system. For both metrics, IPC_i represents the IPC of the i-th application running concurrently with the rest of the applications of a workload set on the system under study.

Our *Minimalist Open-page* scheme is compared against three representative open-page memory controller policies: a) *Parallelism-aware Batch Scheduler* (PAR-BS) [18], b) *Adaptive per-Thread Least-Attained-Service* memory scheduler (ATLAS) [6], and c) *First-Ready, First-Come-First-Served* (FR-FCFS) [19] with open-page policy. Description

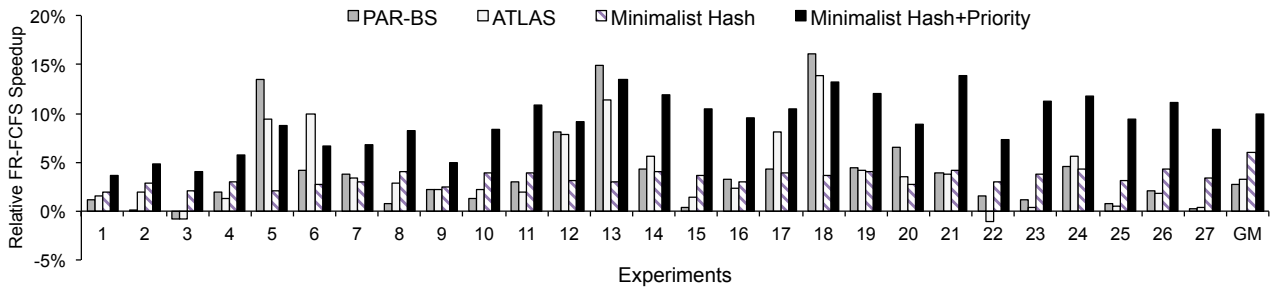


Figure 6: Speedup of PAR-BS, ATLAS, Minimalist with only the hashing scheme and Minimalist with hashing + priority scheme, relative to FR-FCFS

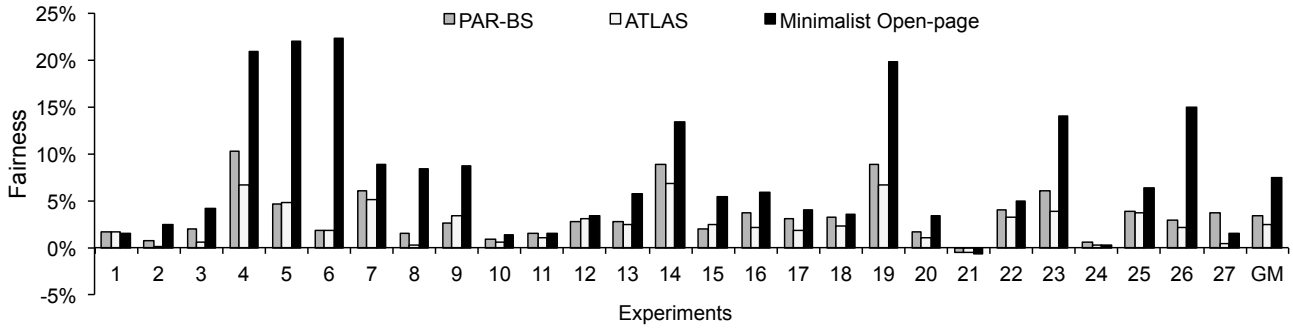


Figure 7: Fairness, compared to FR-FCFS

Table 5: Description of experiments

Scheme	Hash-function	Request Priority	Prefetch Drop delay
FR-FCFS	Fig. 5.a	FR-FCFS	400ns
PAR-BAS	Fig. 5.a	Batch	400ns
ATLAS	Fig. 5.a	ATLAS	400ns
Minimalist Hash	Fig. 5.b	FR-FCFS	400ns
Minimalist Hash+priority	Fig. 5.b	Per Request priority	100ns-400ns

of these policies can be found in our related work section (Section 6). Table 5 summarizes the key differences between the evaluation results presented in this section.

5.1 Throughput

To evaluate the throughput contributions of our scheme we evaluated the Minimalist scheme in two stages. The first, named "Minimalist Hash" is simply the FR-FCFS policy with the "Proposed Mapping" of Figure 5 replacing the "Typical Mapping" of the baseline policy. Secondly, the "Minimalist Hash+Priority" includes the scheduling priority enhancement described in section 4.3.1. The speedup results for all policies relative to the baseline FR-FCFS are shown in Figure 6.

The results can be divided in two categories. In the majority of the results, "Minimalist Hash" performs similarly to the PAR-BS and ATLAS. Workload sets 5, 6, 13, and 18 are notable exceptions, where both PAR-BS and ATLAS significantly outperform "Minimalist Hash". On the other hand, the speedup of the same sets is comparable to the "Minimalist Hash+Priority" policy. These workloads combinations benefited from the prioritization of newer more critical requests provided by the PAR-BS, ATLAS, and "Minimalist Hash+Priority" schemes. Such gains are due to the combination of high bandwidth workloads where at least one of the workloads contains high priority demand misses with low MLP. $GemsFDTD$ is such a workload. Note that not all workloads sets that include $GemsFDTD$ show such behavior, as the interactions between workloads are complicated and difficult to generalize.

An example of such complex interaction was observed when multiple copies of *libquantum* benchmark were concurrently running on the system. The memory access pattern of *libquantum* is very simple as it sequentially accesses a 32MB vector array. This streaming behavior resembles the pathological example of workload A in Figure 4. That said, the performance degradation described in Figure 4 does not always occur when multiple copies of *libquantum* execute on a system. We observed cases where the multiple copies reaches a stable execution phase where all workload copies sequentially accessed memory in lock-step without bank conflicts.

Overall, "Minimalist Hash+Priority" demonstrated the best throughput improvement over the other schemes, achieving a 10% improvement. This is compared against ATLAS and PAR-BS that achieved 3.2% and 2.8% throughput improvements over the whole workload suite. ATLAS and PAR-BS both improve system performance through prioritization of requests waiting to be serviced in the memory queue. Such significant queueing only occurs when the

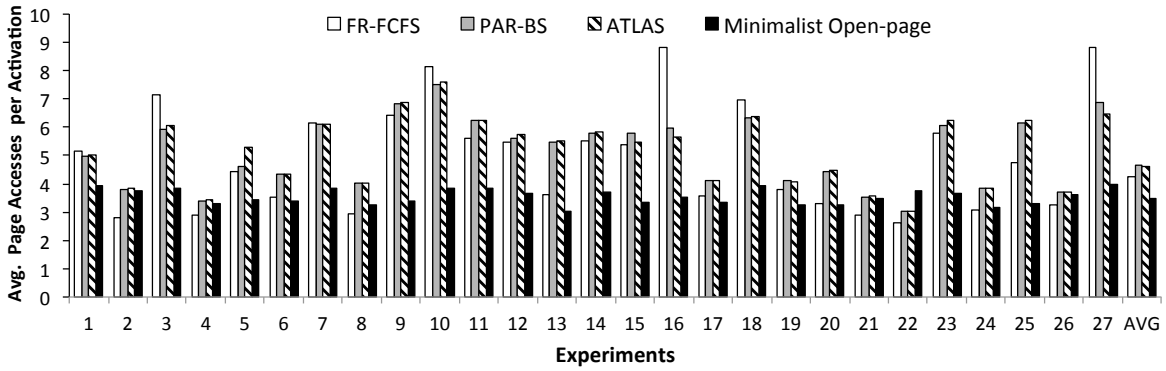


Figure 8: Average number of page-accesses per page activation for all schemes

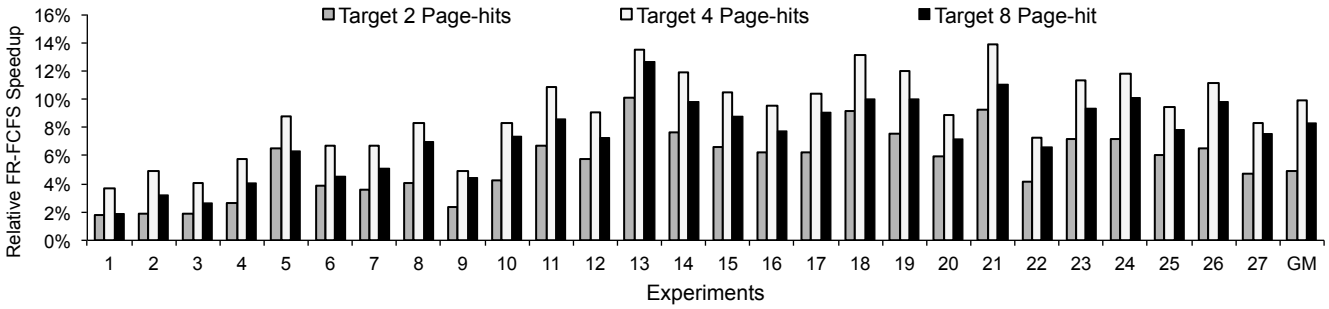


Figure 9: Speedup of targeting 2, 4, and 8 sequential page hits, compared to FR-FCFS

system has completely saturated the memory interface. This is compared to Minimalist scheme where bank conflicts are avoided through decreased bank utilization times provided by the DRAM address mapping. Our evaluation contains a mixture of medium to high bandwidth utilization, and therefore for a number of cases, ATLAS and PAR-BS degenerate to FR-FCFS. In addition, avoiding row buffer starvation through our address hashing enables the memory controller priority scheme to operate on a per memory request critically granularity, compared to a per processor priority scheme utilized in PAR-BS and ATLAS policies.

5.2 Fairness

Figure 7 shows the fairness improvement of all schemes relative to FR-FCFS baseline system using the harmonic mean of weighted speedup. It is important to note that the throughput gains Minimalist achieves are accompanied with improvements in the fairness. This is expected as the throughput gains are realized by alleviating unresolvable conflict cases associated with row buffer starvation. Essentially, Minimalist matches the throughput gains in cases without row buffer conflicts while significantly improves cases where row buffer conflicts exist.

As explained by Kim [6], ATLAS is less fair than PAR-BS, since ATLAS targets throughput over fairness (interestingly we saw similar throughput for both algorithms in our experiments). Minimalist improves fairness up to 15% with an overall improvement of 7.5%, 3.4% and 2.5% for FR-FCFS, PAR-BS and ATLAS, respectively.

5.3 Row Buffer Access per Activation

Figure 8 shows the average number of row buffer column reads for each activation. The observed page-access rate for the aggressive open-page policies fall significantly short of the ideal hit rate of 128, with average values of 4.25, 4.64, and 4.63 for FR-FCFS, PAR-BS, and ATLAS respectively. The high page hit rate is simply not possible given the interleaving of requests between the eight executing programs. With the Minimalist scheme, the achieved page-access rate is close to 3.5, compared to the ideal rate of four.

5.4 Target Page-hit Count Sensitivity

The Minimalist system requires a target number of page-hits to be selected that indicates the maximum number of pages hits the scheme attempts to achieve per row activation. As described in Section 3, for 1333MHz DDR3 DRAM, we found that a good number for page-mode hits is ≈four page hits. To validate this, we evaluated the performance of two, four, and eight page hits as out target number. This is implemented as one, two or three column bits above the cache block in the address mapping scheme (see Figure 5 where a target of four is shown). The results of our simulations, shown in Figure 9, verify that a target number of 4 pages hits provides the best results. Note that different system configuration may shift the optimal page-mode hit count.

5.5 DRAM Energy Consumption

To evaluate if the throughput and fairness gains adversely

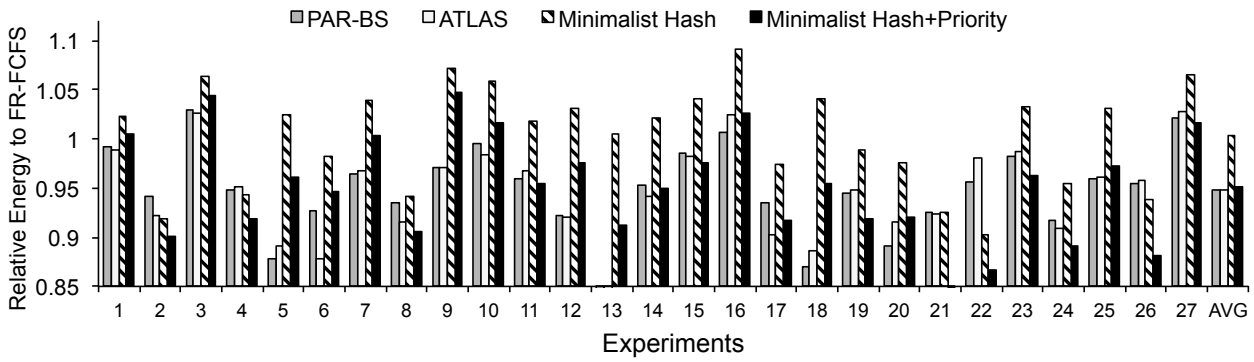


Figure 10: DRAM Energy Relative to FR-FCFS

affect the system energy consumption, we estimated the energy consumption of the various policies. Figure 10 shows the DRAM energy of the PAR-BS, ATLAS, and Minimalist policies relative to the FR-FCFS policy. To estimate the power consumption we used the Micron power calculator [14]. The geometric mean of the relative energy across all experiments of the policies is approximately the same as FR-FCFS. PAR-BS, ATLAS and "Minimalist Hash+Priority" provide a small decrease of approximately 5% to the overall energy consumption. The "Minimalist Hash" without the priority scheme shows an increase in energy as it has a similar page-hit rate with the "Minimalist Hash+Priority" but lower throughput. The energy results are essentially a balance between the decrease in page-mode hits (resulting in high DRAM activation power) and the increase in system performance (decreasing runtime). Note that decreasing runtime, by increasing performance, has the effect of decreasing the background power contribution on energy. Final, the specific results are based on a typical DDR3 row-buffer size. Assuming a Minimalist policy on the system, a DRAM device could be designed with much smaller row-buffers, which would result in energy reductions for the Minimalist policy as compared to current practices.

6. RELATED WORK

Rixner *et al.* [19] was the first to describe the *First-Ready First-Come-First-Serve* scheduling policy that prioritizes row-hit requests over other requests in the memory controller queue. Their proposal utilizes a combination of a column centric DRAM mapping scheme, similar to the one in Figure 5(a), combined with FR-FCFS policy. However, their approach creates starvation and throughput deficiencies when applied to multi-threaded systems as described by Moscibroda *et al.* [16]. Prior work attempts to mitigate these problems through memory requests scheduling priority, but is only able to select between the two suboptimal solutions of Scenario 1 and 2 of Figure 4. Solutions that bias row buffer hits such as FR-FCFS [19] map to Scenario 1. Mutlu *et al.* based their "Stall Time Fair Memory" (STFM) scheduler [17] on the observation that giving priority to requests with opened pages can lead to significant introduction unfairness in the system. As a solution they proposed a scheme that identifies threads that are stalled for a significant amount of time and prioritize them over requests to open-pages. On the average case, STFM will operate similarly to FR-FCFS mapping to Scenario 1 of Figure 4.

The *Adaptive per-Thread Least-Attained-Service* memory scheduler (ATLAS) [6] proposal, that tracks attained service over longer intervals of time, would follow Scenario 2, where the low bandwidth workload B would heavily penalize workload A. Following the same logic, *Parallelism-aware Batch Scheduler* (PAR-BS) [18] ranks lower the applications with larger overall number of requests stored in every "batch" formed in the memory queue. Since streaming workloads inherently have on average a large number of requests in the memory queue, they are scheduled with lower priority and therefore would also follow Scenario 2. The most recent work, *Thread Cluster Memory Scheduler* (TCM) [7] extends the general concept of the ATLAS approach. In TCM, unfriendly workloads with high row-buffer locality, that utilize a single DRAM bank for an extended period of time, are given less priority in the system, such that they interfere less frequently with the other workloads.

With the Minimalist hashing scheme such fairness problems are avoided as we limit the source of the unfairness, that is the row buffer hits per bank. This property is critical in the difference between the prior open-page work [18, 6, 7, 19] and Minimalist. In Minimalist, the memory system is inherently fair. Therefore, no invasive priority policies are introduced, allowing our controller to make good scheduling decisions without queuing of a significant number of memory read operations. Such queuing is detrimental to the observed latency of each individual memory request. In addition, while prior work has improved the problems with FR-FCFS, significant complexity is introduced and the priority selection forces some workloads to experience significant bank conflict delays. Our scheme is much simpler and allows us to focus our priority scheme on servicing the requests that are most critical to program execution.

Lee *et al.* [10] propose a *Prefetch-Aware* controller priority, where processors with a history of wasted prefetch requests are given lower priority. In our approach, prefetch confidence and latency criticality are estimated for each request, based on the state of the prefetch stream combined with the history of stream behavior. With this more precise per request information, more accurate decisions can be made. Lin *et al.* [11] proposed a memory hierarchy that coordinated the operation of the existing prefetch engine with the memory controller policy to improve bus utilization and throughput. In their hierarchy, the prefetch engine issues requests that

are spatially close to recent demand misses in L2 with the memory controller sending the requests to memory only when the memory bus is idle. Their prefetcher relies on a column-centric address hash which introduces significant unfairness in the system that is not directly addressed in their proposal.

7. CONCLUSIONS

In this paper we introduce a *Minimalist Open-page* memory scheduling policy. We show that page-mode gains can be realized with a relatively small number of page accesses for each page activation. By introducing "just enough" page-mode accesses per bank, our *Minimalist Open-page* policy can drastically reduce the negative aspects of page-mode, such as starvation and row buffer conflicts, while maintaining most of its gains. Using this policy, we are able to build intuitive memory scheduler priority policies based strictly on age and request criticality. We derive these request attributes through monitoring program Memory-level Parallelism (MLP) and request stream information within data prefetch engine.

Overall our scheme is effective in concurrently improving throughput and fairness across a wide range of memory utilization levels. It is particularly effective, compared to prior work, in improving workload combinations that contain streaming memory references (throughput increased by 10% on average). Compared to prior work, thread based priority information is not needed (or helpful), which enables workloads with requests of multiple priorities to be efficiently scheduled. In addition, no coordination between multiple memory controllers or operating system interaction is required. This alleviate overall system complexity, enabling other components of the system to be optimized.

Acknowledgements

The authors would like to thank Mattan Erez and Hillery Hunter for their valuable comments on improving the quality of this work. The authors acknowledge the use of the Archer infrastructure for their simulations. This work is sponsored in part by the National Science Foundation under award 0702694 and CRI collaborative awards: 0751112, 0750847, 0750851, 0750852, 0750860, 0750868, 0750884, 0751091, and by IBM. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or IBM.

References

[1] J. L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[2] I. Hur and C. Lin. Feedback mechanisms for improving probabilistic memory prefetching. In *15th International Symposium on High Performance Computer Architecture, 2009*, pages 443 –454, 2009.

[3] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[4] JEDEC Committee JC-42.3. JESD79-3D, Sept. 2009.

[5] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. POWER 7: IBM's next-generation server processor. *IEEE Micro*, 30:7–15, March 2010.

[6] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *16th International Symposium on High Performance Computer Architecture, 2010*, pages 1 –12, 2010.

[7] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *43rd International Symposium on Microarchitecture*, 2010.

[8] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the 8th International Symposium on Computer Architecture*, pages 81–87, 1981.

[9] H. Q. Le et al. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639 –662, 2007.

[10] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt. Prefetch-aware DRAM controllers. In *41st International Symposium on Microarchitecture*, pages 200–209, 2008.

[11] W. Lin, S. Reinhardt, and D. Burger. Designing a modern memory hierarchy with hardware prefetching. *IEEE Transactions on Computers*, 50(11):1202 –1218, 2001.

[12] K. Luo, J. Gummaraju, and M. Franklin. Balancing throughput and fairness in SMT processors. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2001.

[13] P. Magnusson et al. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[14] Micron Technologies, Inc. DDR3 SDRAM system-power calculator, revision 0.1, 2007.

[15] M. Milo et al. Multifacet: A general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News*, 33, 2005.

[16] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *16th USENIX Security Symposium on USENIX Security Symposium*, pages 18:1–18:18, 2007.

[17] O. Mutlu and T. Moscibroda. Stall-Time fair memory access scheduling for chip multiprocessors. In *40th International Symposium on Microarchitecture, 2007*, pages 146 –160, 2007.

[18] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *35th Annual International Symposium on Computer Architecture*, pages 63–74, 2008.

[19] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *27th Annual International Symposium on Computer Architecture*, pages 128–138, 2000.

[20] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter, and L. K. John. The virtual write queue: coordinating DRAM and last-level cache policies. In *37th annual international symposium on Computer architecture*, pages 72–82, 2010.

[21] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *33rd annual ACM/IEEE international symposium on Microarchitecture*, MICRO 33, pages 32–41, 2000.