Adaptive Row Addressing for Cost-Efficient Parallel Memory Protocols in Large-Capacity Memories

Dmitry Knyagin
Chalmers University of
Technology
Gothenburg, Sweden
dmitryk@chalmers.se

Vassilis Papaefstathiou
Chalmers University of
Technology
Gothenburg, Sweden
vaspap@chalmers.se

Per Stenstrom
Chalmers University of
Technology
Gothenburg, Sweden
pers@chalmers.se

ABSTRACT

Modern commercial workloads drive a continuous demand for larger and still low-latency main memories. JEDEC member companies indicate that parallel memory protocols will remain key to such memories, though widening the bus (increasing the pin count) to address larger capacities would cause multiple issues ultimately reducing the speed (the peak data rate) and cost-efficiency of the protocols. Thus to stay high-speed and cost-efficient, parallel memory protocols should address larger capacities using the available number of pins. This is accomplished by multiplexing the pins to transfer each address in multiple bus cycles, implementing Multi-Cycle Addressing (MCA). However, additional address-transfer cycles can significantly worsen performance and energy efficiency. This paper contributes with the concept of *adaptive row addressing* that comprises *row-address caching* to reduce the number of address-transfer cycles, enhanced by *row-address prefetching* and an *adaptive row-access priority policy* to improve state-of-the-art memory schedulers. For a case-study MCA protocol, the paper shows that the proposed concept improves: i) the read latency by 7.5% on average and up to 12.5%, and ii) the system-level performance and energy efficiency by 5.5% on average and up to 6.5%. This way, adaptive row addressing makes the MCA protocol as efficient as an idealistic protocol of the same speed but with enough pins to transfer each row address in a single bus cycle.

CCS Concepts

•Hardware → Semiconductor memory; Buses and high-speed links; •Computer systems organization → Architectures;

Keywords

Parallel memory protocols; address bus; row-address width; memory-request scheduling; performance; energy; fairness

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MEMSYS 2016, October 3–6, 2016, Washington, DC, USA

© 2016 ACM. ISBN 978-1-4503-4305-3...\$15.00

DOI: <http://dx.doi.org/10.1145/2989081.2989103>

1. INTRODUCTION

Parallel Double Data Rate (DDR) protocols with multi-drop buses have been de facto standard main memory protocols for about 15 years, and DDR4 [2] is the latest in the line. Although emerging point-to-point protocols like Hybrid Memory Cube (HMC) [13] offer significantly higher peak bandwidths, they fall short in terms of latency in large-capacity memories. For instance, HMC devices have to be connected into a “far memory” network to increase capacity, and each hop adds latency¹. Parallel protocols with multi-drop buses do not have this issue; thus they will remain key to large-capacity and still low-latency main memories [33].

Future memory capacity growth implies wider addresses. However, the address bus currently implemented in DDR4 has reached the limit of the feasible number of pins for high-speed (high-data-rate) parallel protocols: it is the widest bus of the protocol, and increasing its width would present a connectivity challenge leading to signal integrity issues, routing congestion, pad-limited designs, and increased manufacturing expense [6]. The burden of additional address pins would propagate through memory modules and channels down to the processor, requiring system-level changes. That is, widening the address bus would render parallel protocols slow and cost-inefficient.

So to meet large-capacity demands, parallel protocols should multiplex the available pins and transfer each address in multiple bus cycles², implementing Multi-Cycle Addressing (MCA). An idealistic protocol labeled DDR^{id} would have the same speed but enough pins to transfer each address in a single bus cycle, implementing Single-Cycle Addressing (SCA). Compared to DDR^{id} , MCA protocols can have significantly lower performance and energy efficiency, and so it is important to improve them.

This paper contributes *adaptive row addressing* as a general approach to close the performance and energy-efficiency gaps between MCA and idealistic SCA protocols. It does so by combining three techniques. A first technique is *row-address caching* that exploits row-address locality [31, 11] to reduce the number of cycles per address transfer by caching the most-significant row-address bits. We propose 2-way row-address caches with a custom organization for

¹Based on state-of-the-art Xilinx SERDES [37], our optimistic latency estimates for a 16-lane link with 15Gbps lanes are 15-20ns per hop, one way. Assuming 40-50ns internal DRAM latencies, such link latencies are inline with the read latencies of a single HMC measured by Gokhale et al. [12].

²Unless stated otherwise, the word “cycle” denotes an address-bus cycle throughout this paper.

high efficiency. To alleviate the performance penalty of address-cache misses, we propose a second technique: *row-address prefetching* that is effective yet simple to integrate with state-of-the-art memory schedulers. Further, our detailed analysis of memory-request scheduling reveals that row-address caching can negatively impact the request service order. This holds for state-of-the-art schedulers that reorder row accesses using the conventional first-ready policy [38, 32] (for instance, FRFCFS [38, 32] or BLISS [34], among many others). To eliminate the negative impact, we propose a third technique: an *adaptive row-access priority policy* that can simply replace the first-ready policy.

We study the effectiveness of the above techniques using a high-speed, cost-efficient MCA protocol based on DDR4 in large-capacity, low-latency main memories. The read latency gap between the MCA protocol and DDR^{id} is 7.5% on average and up to 12.5%; the system-level performance and energy efficiency gaps are 5.5% on average and up to 6.5%. Our evaluation shows that: i) the proposed 2-way row-address caches perform nearly as good as fully-associative ones; ii) the benefit of row-address prefetching exceeds the benefit of doubling the address-cache size; and iii) adaptive row-access priority policy cooperates with row-address prefetching to achieve the best performance. Combined, the three techniques of adaptive row addressing robustly close the gap between the MCA protocol and DDR^{id}.

The rest of this paper is organized as follows. Section 2 presents the background and motivation, Section 3 adaptive row addressing, and Section 4 describes the experimental setup. Section 5 presents the results, Section 6 discusses related work, and Section 7 concludes the paper.

2. BACKGROUND AND MOTIVATION

According to insights by JEDEC member companies, future systems will employ a point-to-point protocol for high bandwidth and a parallel protocol with multi-drop buses for large capacity and low latency [33]. Next, Section 2.1 provides relevant background information on large-capacity memories and DDR4, the latest parallel memory protocol. Section 2.2 motivates Multi-Cycle Addressing (MCA), and Section 2.3 motivates adaptive row addressing.

2.1 DDR4 DRAM Memory System

Device Organization. A DDR4 device is organized as up to eight 3D-stacked memory dies connected by through-silicon vias. The dies share the Command/Address (CA) bus and the data bus. A die contains four bank groups each comprising four banks, as shown in Figure 1 [29]. Each bank is organized as a memory array with up to 256K rows and 1K columns [2].

Device Operation. The precharge command (*PRE*) prepares the target bank for an activation and incurs a delay labeled t_{RP} . The activate command (*ACT*) opens a row in a bank, i.e., senses the target row into the bank’s *row buffer* (also known as *sense amplifiers*). *ACT* incurs a delay labeled t_{RCD} . Next, the target column is accessed in the row buffer by column read (*RD*) or write (*WR*) commands with respective delays CL and CWL . The row buffer is accessed in bursts of eight data-bus half-cycles, and so each column access occupies the data bus for four cycles. Activations

Figure 1: Simplified organization of one DDR4 die

Figure 2: RDIMM with 16 memory devices (D15-D0) and 64-bit data bus (DQ[63:0])

are destructive, thus the content of the row buffer has to be restored to the memory array. The restore starts automatically in the background and its latency is t_{RAS} . Hence, the minimum delay between consecutive ACTs to the same bank is $t_{RC} = t_{RAS} + t_{RP}$. The delay between consecutive ACTs to different banks in the same bank group is labeled $t_{RRD,L}$, and that to different banks in different groups is labeled $t_{RRD,S}$. Because of bank grouping, $t_{RRD,L} > t_{RRD,S}$. Likewise, there are two delays between consecutive column accesses: $t_{CCD,L}$ and $t_{CCD,S}$, where $t_{CCD,L} > t_{CCD,S}$ [2].

Device Address Pins. A typical DDR4 device used in large-capacity memories has 78 pins in total, half of which are power and ground and four are data pins [29]. Most of the remaining pins belong to the CA bus.

The address pins of the CA bus include three Chip-ID pins for addressing the 3D-stacked dies, two bank-group and two bank-address pins for addressing the banks, and 18 pins labeled **A**[17:0] for row addressing during ACT. Pins **A**[9:0] are multiplexed for column addressing within the open row during the column access commands (RD or WR).

The row address is significantly wider than the column address (18 vs. 10 bits), and in large-capacity memories this width gap has been growing. That is, the row-address width dictates the CA-bus width. As a result, some of the CA-bus pins are under-utilized: our analysis reveals that pins **A**[17] and **A**[13] are functionally used only during ACT³.

³Pins **A**[17] and **A**[13] are used during ACT and Mode Register Set (MRS), but during MRS they must be set to zero [2] and thus they are not functionally used (i.e., they can be internally hardwired to zero).

Figure 3: Read latency increase due to two-cycle row addressing (one data point per workload)

Channel Organization. A parallel memory channel is organized as one or more memory modules that share the CA bus and the data bus. A module is organized as a number of memory devices that share the CA bus, but each device connects to a private slice of the data bus. For instance, the Registered Dual In-line Memory Module (RDIMM) [23] in Figure 2 [24] holds 16 memory devices⁴ each with four data pins forming a 64-bit data bus.

Channel Operation. Each die of a memory device operates in lockstep with the respective dies across the other devices of the module forming a *rank*. For instance, the 16 devices in Figure 2 with eight dies per 3D stack form eight ranks. The number of consecutive ACTs to the same rank is limited to four per sliding window t_{FAW} . Consecutive column accesses to different ranks incur a switching delay labeled t_{RTRS} .

Electrical Constraints. Unlike the DDR data bus, the CA bus is Single Data Rate (SDR) due to a high electrical load in large-capacity memories, as follows. Each memory device appears as a single electrical load regardless of the number of dies per 3D stack, since the bottom die isolates the loads of the other dies. Thus an Unbuffered Dual In-line Memory Module (UDIMM) [22] with 16 devices appears as 16 loads on the CA bus and as one load on each 4-bit slice of the 64-bit data bus. Populating the channel with a second UDIMM doubles the respective numbers of loads. A heavily loaded CA bus can lead to low data rates, since: 1) the maximum operating clock rate (frequency) for reliable transmission on a bus decreases as the number of loads on the bus grows, and 2) the clock signal is shared between the CA bus and the data bus. Thus in large-capacity, high-performance memories UDIMMs have been superseded by RDIMMs, that register the CA bus as shown in Figure 2. An RDIMM appears as a single load on the channel’s CA bus (the *pre-register* CA bus). However, the number of loads on the RDIMM’s CA bus (the *post-register* CA bus) is still large, totaling half the number of memory devices per module. Load Reduced DIMMs (LRDIMMs) register both the CA bus and the data bus [20]. Though, they do not reduce the number of loads on the post-register CA bus [21]. Thus, even if RDIMMs or LRDIMMs are employed, the CA bus has a large number of loads and has to be SDR in order to guarantee the peak data rates.

2.2 Multi-Cycle Addressing (MCA)

Future memory capacities are expected to grow. For instance, the International Technology Roadmap for Semiconductors predicts 3D stacks taller than eight dies [15]. In addition, memory die densities continuously increase [16].

⁴We exclude memory devices used for error detection and correction, since it is outside the scope of the paper.

Figure 4: System-level performance loss due to two-cycle row addressing

Capacity growth implies wider addresses. However, widening the address bus beyond that of DDR4 would cause multiple issues manifesting themselves in lower speeds and higher costs [6]. Besides, widening the bus for row addressing would increase the number of already under-utilized pins. Thus to stay high-speed and cost-efficient, parallel memory protocols have to address larger capacities by using the available pins economically. In addition, the CA bus has to be SDR in order to comply with the stringent electrical constraints, required to guarantee the peak data rates⁵. This suggests MCA, where pins are multiplexed to transfer each address in multiple CA-bus cycles.

2.3 DDR4-Based Two-Cycle Row Addressing

To avoid costly disruptions of the well-established DDR4 ecosystem, we consider an MCA protocol that is based on DDR4 and has the same pin count and speed. To support future large capacities while using the available pins economically, we reassign the under-utilized A[17] and A[13] from row address to Chip ID thus enabling up to 32 dies per 3D stack. Row addresses are transferred over the remaining 16 pins (A[16:14], A[12:0]) in two cycles⁶.

Although some address-transfer cycles can be overlapped with bank-busy cycles, the opportunity to do so is limited. Additional address-transfer cycles interfere with other commands on the CA bus, causing the MCA protocol to perform significantly worse than DDR^{4d} (the idealistic protocol with enough pins for SCA). Figure 3 shows that two-cycle row addressing in 100 multi-program workloads and the system from Section 4 increases the read latency by about 7.5% on average and up to 12.5%. Figure 4 shows that two-cycle row addressing reduces the system-level performance by about 5.5% on average and up to 6.5%. Since we consider low-latency, high-performance memories, it is important to improve the efficiency of the MCA protocol.

3. ADAPTIVE ROW ADDRESSING

We propose the concept of *adaptive row addressing* to close the efficiency gap between MCA and idealistic SCA protocols. Adaptive row addressing comprises three techniques described in the following sections: *row-address*

⁵Making the CA bus DDR would reduce its maximum frequency and hence it would slow the data bus.

⁶An alternative optimization could reassign three pins—A[17], A[13], A[11]—from row address to Chip ID and, e.g., bank address (should the number of banks increase in the future). This optimization affects only Mode Register Set (MRS) and ACT [2]. The MRS opcodes would have to be sent in two cycles over pins A[12], A[10:0]. However, MRS is used only during initialization. During normal operation, pins A[16:14], A[12], A[10:0] would transfer row addresses in two cycles. We evaluate an equivalent of this optimization as part of sensitivity analysis in Section 5.2.4.

Figure 5: Address mappings

caching (Section 3.1), row-address prefetching (Section 3.2), and adaptive row-access priority policy (Section 3.3).

3.1 Row-Address Caching

The idea of address caching is to reduce the number of address-transfer cycles by exploiting address locality [31, 11]. The Most-Significant Portion (MSP) of each address can be cached on the memory-device side and later encoded by the memory controller with fewer bits, making it possible to transfer the entire address in fewer cycles.

We propose to employ one row-address cache per bank and to instantiate the address-cache update logic off the critical path on both the Memory Controller (MC) and the memory-device sides. Before issuing an ACT, the MC checks its respective cache for the MSP of the target row address. Upon a hit, it encodes the hit location and sends it to the target bank along with the Least-Significant Portion (LSP) of the row address, in one cycle. Upon a miss: 1) the MC encodes the miss and sends it with the LSP in one cycle, followed by the MSP in the second cycle; and 2) the MC updates its respective cache, and the memory dies of the target rank mirror the update in their own respective caches.

Row-address caches are most effective if consecutive memory accesses have the same MSPs. If MSPs are different, row-address caches can still be effective if the reuse distances [26] of the MSPs are short. In the following sections we discuss row-address locality, row-address cache organizations, and their implementation details.

3.1.1 Row-Address Locality

Row-address locality depends on a number of factors: 1) program row-access patterns, 2) virtual-to-physical address mapping, 3) physical-to-DRAM address mapping, and 4) interference among co-running programs. Figure 5 illustrates the virtual-to-physical and physical-to-DRAM address mappings in a system with a 48-bit virtual address space, 8-KB virtual pages (and 8-KB physical frames), four channels, 32 ranks per channel, and 64-B cache blocks⁷. Figure 5 shows a physical-to-DRAM address mapping that is considered baseline for the open-page row-buffer management policy [19]. Figure 5 also shows how the *row* field (the 18-bit row address) is split into two portions by the 16 row-address pins, with bits 41-40 being the MSP.

Virtual-to-physical address mapping is a major factor affecting row-address locality with two extremes: 1) the Operating System (OS) maps virtual pages to sequential physical frames yielding high locality, and 2) the OS maps pages to random frames yielding low locality. In real systems, an intermediate amount of row-address locality can be expected.

⁷The *block* field (bits 14-8) is 7-bit wide to address each 64-B cache block per aggregate 8-KB row buffer of 16 banks operating in lockstep, where each bank has a 512-B row buffer.

Figure 6: Address-caching schemes R-1, F-31, D-31, and W-31

3.1.2 Row-Address Cache Organizations

Unlike in conventional caches, in such row-address caches tags are the same as data and the address space to be cached increases with the cache size, as follows. Since each cache location has to be encoded together with the LSP, doubling the address-cache size pushes out one bit from the LSP to the MSP thus doubling the number of possible MSP values. We roughly estimate the *efficiency of encoding* by the ratio of the number of possible MSP values over the address-cache size (the lower the ratio, the higher the efficiency).

Figure 6 first shows the simplest address-caching scheme labeled *R-1* that employs one MSP register per bank. *R-1* uses one bit to encode a miss or a hit, and thus the MSP of the 18-bit row address is 3-bit wide and the ratio is $2^3/1 = 8$. A 3-entry cache would require two bits to encode a miss or a hit location, and so the MSP becomes 4-bit wide but the ratio improves to $2^4/3 = 5.33$.

Next, Figure 6 shows a scheme labeled *F-31* that employs 31-way fully-associative caches. Bits 15-11 encode a miss or a hit way, thus the MSP is 7-bit wide (bits 22-16) and the ratio is $2^7/31 = 4.13$. The entire MSP has to be cached, and so the MSP-storage size is $31 * 7 = 217$ bits per *F-31* cache.

The third scheme in Figure 6 is labeled *D-31* and employs 31-set direct-mapped caches. Upon a miss, the caches are indexed by bits 20-16, and since the number of sets is not a power of two, index 31 is wrapped to set 0. Since MSPs with different bits 20-16 (index 31 and index 0) can map to set 0, the entire MSP has to be cached. Thus, the MSP-storage size of *D-31* is the same as that of *F-31*. Making the number of sets a power of two simplifies indexing but reduces the efficiency of encoding: like *F-31*, *D-31* has the ratio of $2^7/31 = 4.13$, but a scheme with 32-set direct-mapped caches would have a much worse ratio of $2^8/32 = 8$.

The last scheme in Figure 6 is labeled *W-31* and employs 31-entry, 2-way set-associative caches. The number of sets is 16 which simplifies indexing, and to achieve the same encoding efficiency as *D-31* and *F-31*, we propose to disable the second way of the last set and use the all-ones combination to encode a miss. Thus upon a hit, bit 15 encodes

Figure 7: Implementation sketch of address-caching scheme W-31

Figure 8: Address-cache miss curves of comm2 [1]. Markers show best miss rates for respective MSP widths

the hit way and bits 14-11 the hit set, and in set 15 only the first way can be addressed. Upon a miss, bits 15-11 are set to one. W-31 requires a smaller MSP storage than F-31 and D-31, since MSPs with different bits 19-16 map to unique sets. That is, bits 19-16 come directly from the hit-set index (bits 14-11) and only three bits of the MSP (bits 22-20) have to be cached, reducing the MSP-storage size to $31 * 3 = 93$ bits per W-31 cache. Thus the advantages of the proposed W-31 are simplicity, high encoding efficiency, small MSP storage, and a conflict-miss rate likely between those of F-31 and D-31.

3.1.3 Implementation Details

On the device side, we instantiate row-address caches right before the Control Unit’s Command/Address Register (recall Figure 1). Figure 7 shows an implementation sketch of W-31. For brevity, we rename pins **A[16:14]**, **A[12:0]** to **A_int[15:0]**. Upon an address-cache hit, the row address is a concatenation of three cached MSP bits (block 1H in Figure 7) and 15 bits sent by the MC (**A_int[14:0]**).

The circuit operates transparently to the Control Unit (no changes to the Control Unit are needed). The row-address caches are tiny, and so decoding the hit location and reading out the MSP can be performed very fast, without a performance penalty on the SDR CA bus. The MSP-storage overhead is trivial and is dwarfed by the area of the banks.

3.2 Row-Address Prefetching

The tiny size of row-address caches implies that they might have high miss rates. In the following sections we discuss the miss rates, propose *row-address prefetching* to tackle them, and describe its implementation.

3.2.1 Address-Cache Miss Rates

We collect Least-Recently Used (LRU) stack-distance histograms using Mattson’s algorithm [26] at the granularity of one cache block (64B) for single-program workloads and the system from Section 4 but with just one bank per rank. We consider MSP widths from 3 to 9 bits that yield, respectively, 1 to 7 bits to encode a miss or a hit location. Thus, the largest possible address-cache sizes are 1 to 127 entries.

Figure 8 shows the miss curves of comm2 from the Memory Scheduling Championship (MSC) suite [1] for one of the banks (the miss curves for the other banks are very similar). The markers indicate the miss rates of the largest possible row-address caches for the respective MSP widths, i.e., the *best miss rates*. For instance, Figure 8a shows that when the virtual-to-physical mapping is sequential (the first extreme, yielding high locality) all of the MSP values can be cached, except when the MSP is 3-bit wide: then the best miss rate is 20%. Figure 8b shows that when the MSC frame numbers [1] are used, the best miss rate is about 50% for the 3-bit MSP (1-entry row-address cache) and about 5% for the 7-bit MSP (31-entry row-address cache). Figure 8c shows that when frames are allocated randomly (the second extreme, yielding low locality) the best miss rates increase dramatically. The MSC frame numbers [1] exhibit an intermediate amount of row-address locality that can be expected in real systems.

When the program is executed in a multi-program workload, the miss rates are likely to increase due to interference. Next we describe row-address prefetching, a technique to alleviate the penalty of address-cache misses.

3.2.2 Row-Address Prefetch Strategy

We propose row-address prefetching and implement it in the memory-request scheduler. We define a row-address

prefetch as a command that transfers a request’s row-address MSP to the row-address cache of its target bank.

Schedulers maintain request queues, where the *oldest request* is the one that got enqueued first. The *next command* for each request depends on the status of its target bank and can be ACT, PRE, RD or WR. Memory timing constraints define whether a command can be issued at a specific CA-bus cycle. A row-address prefetch can be issued at an idle cycle, i.e., if no other command can be issued at that cycle⁸.

We propose the following prefetch strategy. The scheduler tracks banks that are eligible for prefetch, and initially all banks are flagged as eligible. A bank is flagged as ineligible if the oldest request to that bank has a one-cycle row address and its next command is ACT or PRE. A prefetch is permitted for a request if: 1) its target bank is flagged as eligible and 2) the request has a two-cycle row address and its next command is ACT or PRE. All permitted prefetches are prioritized first by command (requests whose next command is ACT get the high priority) and then by age (the oldest request gets the high priority). At an idle CA-bus cycle, the prefetch with the highest priority is issued. This way we prefetch for as many requests as possible and avoid prefetch interference per bank, i.e., address-cache interference of later prefetches with earlier ones. This strategy is an efficient tradeoff between the simplest strategy that prefetches for the oldest request (regardless of the target bank) and the finest-grain strategy that tracks address-cache sets eligible for prefetch (and so avoids prefetch interference per set).

3.2.3 Implementation Details

The protocol has to have a reserved command available. For instance, DDR4 has one reserved command [2] and MSPs up to 13 bits can be sent over pins A[12:0]. The prefetch command is executed only by the address-cache control logic. An extension of the sketch in Figure 7 is straightforward: upon a prefetch command, update the row-address cache using A_int[6:0]. The storage overhead is one bit per bank to flag banks eligible for prefetch.

The proposed row-address prefetching is easy to integrate with state-of-the-art schedulers. For instance, BLISS [34] temporarily blacklists programs that recently issued four consecutive column accesses and prioritizes non-blacklisted programs. This simply adds one priority level to the proposed prefetch strategy: all permitted prefetches are first prioritized by program (requests of non-blacklisted programs get the high priority), then by command, and lastly by age.

3.3 Adaptive Row-Access Scheduling

Counter-intuitively, row-address caching and prefetching can increase the execution time compared to two-cycle row addressing in some cases⁹. We find that the problem is caused by the memory-request scheduler being agnostic to

⁸Idle cycles are expected since the CA bus is never utilized more than the data bus. Each request occupies the data bus for four cycles. Upon a row-buffer miss (PRE + ACT + RD/WR), a request with a two-cycle row address occupies the CA bus for four cycles. However, a request with a one-cycle row address occupies the CA bus for three cycles. Upon a row-buffer hit (RD/WR), each request occupies the CA bus for only one cycle. Hence, there is at least one idle CA-bus cycle per request with a one-cycle row address.

⁹For instance, R-1 with row-address prefetching slows the execution of comm1 [1] by 2% in the system from Section 4 but with one bank per rank, which emphasizes the problem.

Figure 9: Negative impact of row-address caching on request service order

the variable number of cycles per row-address transfer. Next we present a detailed memory-request scheduling analysis, propose *adaptive row-access priority policy*, and discuss its implementation details.

3.3.1 Analysis of Memory-Request Scheduling

Because of row-address caching, some ACTs get one-cycle row addresses (A1) while the other ACTs keep two-cycle row addresses (A2). Assuming that the address-cache control logic can register the row-address LSP while the target bank is busy, the MC can issue A2 one cycle earlier than A1. High-performance schedulers typically maintain read- and write-request queues and reorder requests according to a cascade of policies, where the last policy reorders row-access commands (ACT and PRE). FRFCFS [38, 32] and BLISS [34], among many other state-of-the-art schedulers, employ the conventional first-ready policy [38, 32] as the last in the cascade to prioritize the oldest row-access command that is ready to be issued. Since A2 can be issued one cycle earlier than A1, row-address caching can negatively impact the request service order produced by the first-ready policy, as illustrated in Figure 9. We assume RDIMMs and so A2 and A1 can be issued respectively three ($t_0 - 3$) and two ($t_0 - 2$) cycles before their target banks become ready (t_0). Figure 9 shows that if the older ACT gets a one-cycle address, the younger ACT gets unfairly issued first, solely because it has a two-cycle address. This delays the older ACT, potentially slowing the execution. In a particularly bad case the ACTs have the same target bank and the older ACT belongs to a load at the head of the Re-Order Buffer (ROB).

Note that the problem can slow the execution in both single- and multi-program cases, i.e., regardless if the ACTs belong to the same program/thread. The proposed row-address prefetch strategy is key to alleviate the penalty of address-cache misses. However, it prefetches for the oldest request, and thus increases the probability of the problem.

Table 1 lists the cases when a younger A2 gets unfairly issued ahead of an older row-access command. Figure 10 shows the respective schedules for each case, highlighting the minimum number of cycles that the service of the older read request gets delayed for, according to DDR4 DRAM timings [2, 29]. The width of one rectangle denotes one CA-bus cycle. The rectangles with dashed borders denote the cycles at which the older commands would get issued if there were no younger A2. For instance, in Case 1 the requests have the same target bank, and the delay totals $t_{RAS} + t_{RP} = 28 + 10 = 38$ cycles. The delay can be longer if the younger request is followed by other younger requests that hit in the row buffer. In Case 2 the requests have different target banks on the same channel, and the delay is at least $t_{RRD_S} = 5$ cycles (recall that $t_{RRD_S} < t_{RRD_L}$).

Table 1: Cases when younger A2 gets issued ahead of older row-access command

Target Rank	Target Bank	Older Command A1	Command PRE
Same	Same	Case 1	Impossible
Same	Different	Case 2	Case 4
Different	Any	Case 3	Case 5

Figure 10: Minimum delays experienced by older read request when younger read request gets serviced first

Since in Cases 1 and 2 the ACTs are to the same rank, the service delays can be longer if t_{FAW} is not met. In Case 3 the requests are to banks of different ranks, and the delay is at least $t_{CCD_S} + t_{RTRS} = 4 + 2 = 6$ cycles (recall that $t_{CCD_S} < t_{CCD_L}$). Finally, Cases 4 and 5 are equivalent: the delay is one cycle, since the target banks are different and the PRE can be issued right after the ACT.

We also observe that row-address caching can positively impact the request service order. Figure 11 illustrates it relative to one-cycle row addressing (DDR^{id}). The younger A1 in Figure 11 is ready one cycle before the older A1 and thus gets issued first. However, if due to row-address caching the younger ACT remains A1 but the older ACT becomes A2, both are ready at the same cycle, and so the older ACT gets issued, potentially speeding the execution. Thus it is important to design a row-access priority policy that would eliminate the negative impact and retain the positive impact.

3.3.2 Adaptive Row-Access Priority Policy

We propose an *Adaptive row-access Priority Policy (APP)* as follows. Since the service delays in Cases 1 to 3 are significant, we propose to postpone the younger A2 (that is, to not issue it at the current cycle) if there is an older A1 that will be ready at the next cycle. However, in Cases 4 and 5 there is a tradeoff. If we issue the A2, the service delay of the PRE is just one cycle. If we postpone the A2, its

Figure 11: Positive impact of row-address caching on request service order

service delay would be two cycles. Thus, we propose to not postpone a younger A2 if there is an older PRE that will be ready at the next cycle.

To summarize, APP prioritizes an older A1 ready at the next cycle over a younger A2 ready at the current cycle, regardless whether the ACTs have the same target bank. We find that this single change to the first-ready policy eliminates the negative impact of row-address caching.

The address-cache miss rate can be reduced by prioritizing a younger A1 over an older A2 ready at the same cycle. However, this would eliminate the positive impact of row-address caching (Figure 11) and could slow the execution. Thus, there is no benefit to reduce the address-cache miss rate via scheduling beyond what APP already does.

Note that in the simplest case the MC could ignore the opportunity to issue A2 one cycle earlier than A1. This would avoid the negative impact of row-address caching at the cost of missing the opportunity to overlap the second address-transfer cycle of A2 with a bank-busy cycle. That is, it would diminish the benefit of adaptive row addressing. On the contrary, APP helps to exploit its full potential.

3.3.3 Implementation Details

The proposed APP can be used in state-of-the-art schedulers by simply replacing the first-ready policy. For instance, BLISS [34] prioritizes requests first by program (non-blacklisted programs get the high priority), then by column access, and finally by row access, using the first-ready policy for both column and row accesses. The implementation of APP in BLISS is straightforward: 1) prioritize an older A1 of a non-blacklisted program ready at the next cycle over a younger A2 ready at the current cycle, regardless if its program is blacklisted or not; and 2) prioritize an older A1 of a blacklisted program ready at the next cycle over a younger A2 of a blacklisted program ready at the current cycle.

4. EXPERIMENTAL SETUP

We evaluate adaptive row addressing using a detailed memory system simulator USIMM [7], employed in recent memory-system research [30, 35, 5, 8]. We extended USIMM with: 1) the latency of CA transfers; 2) the latency of the RDIMM's register [23]; 3) DDR4 timing constraints; 4) DDR4 power model that implements Micron's DDR4 System Power Calculator [28] and in addition estimates the *dynamic power of the CA bus*¹⁰; 5) the *row:rank:bank:block:channel:block_offset* physical-to-DRAM address mapping (baseline for the open-page row-buffer management

¹⁰We estimate the CA-bus dynamic power as the termination power of CA transfers. We use the same termination power per CA pin as per data pin [28].

Table 2: System Configuration

Number of cores	32
Core:	
Clock speed	3.2GHz
ROB size	160
Retire width	4
Fetch width	4
Pipeline depth	10
Cache block size	64B
Last-level cache size, total	16MB
Last-level cache size, per core	512KB
Memory bus clock speed	800MHz
Memory channels	4
Ranks per channel	2
Banks per rank	16
Rows per bank	256K
Cache blocks per row	128
Physical address width	38 bits

Table 3: DRAM Device Parameters

Timing (800MHz cycles)	Power supply (V)
t_{RCD}	10
t_{RP}	10
CL	10
t_{RAS}	28
t_{RC}	38
t_{RRD_S}	5
t_{RRD_L}	6
CWL	9
t_{WR}	12
t_{WTR_S}	2
t_{WTR_L}	6
t_{RTP}	6
t_{CCD_S}	4
t_{CCD_L}	5
t_{RTS}	2
t_{REFI}	6240
t_{RFC}	208
t_{FAW}	16
	Current (mA)
	V_{DD} 1.2
	V_{PP} 2.5
	I_{DD0} 58
	I_{PP0} 4
	I_{DD2P} 30
	I_{DD2N} 44
	I_{DD3P} 44
	I_{DD3N} 61
	I_{PPSB} 3
	I_{DD4P} 140
	I_{DD4W} 144
	I_{DD5} 190
	I_{PP5} 22
	DQ pins 4

Table 4: MSC Programs

PARSEC	MPKI
<i>black</i> (blackscholes)	3.2
<i>face</i> (facesim)	6.2
<i>ferret</i>	6.7
<i>fluid</i> (fluidanimate)	3.0
<i>freq</i> (freqmine)	3.0
<i>stream</i> (streamcluster)	3.7
<i>swapt</i> (swaptions)	3.5
BioBench	
<i>mummer</i>	20.4
<i>tigr</i>	27.4
SPEC CPU2006	
<i>leslie</i> (leslie3d)	6.4
<i>libq</i> (libquantum)	14.0
Intel Commercial	
<i>comm1</i>	6.9
<i>comm2</i>	8.5
<i>comm3</i>	3.5
<i>comm4</i>	2.4
<i>comm5</i>	1.6

policy) [19]; and 6) an address extension that separates the physical address spaces of different programs by adding unique, random bits right after the *block* field¹¹. For sensitivity analysis we extended USIMM with: 1) page coloring such that each program gets its own bank [27, 25] and 2) sequential and random virtual-to-physical address mappings.

System Configuration. Table 2 shows key system parameters. We configure the system to have a similar number of cores and channels as Intel Xeon E7-8890 v3 [18], which has 36 threads (18 cores) and four channels. The baseline system has 32 cores and a 38-bit physical address space formed by four channels, two ranks per channel and 16 DDR4 DRAM dies per rank with parameters shown in Table 3 [29]. The last-level cache size is scaled down accordingly to the scaling of the program execution for simulation [1]. The system uses 8-KB OS pages, the default USIMM virtual-to-physical address mapping (the MSC frame numbers [1]), and the pessimistic address extension described above. The MC employs one read-request queue and one Write-Request Queue (WRQ) per channel, FRFCFS for each queue, and the default USIMM policy for WRQ draining. The WRQ size, high and low watermarks are 96, 60 and 20, respectively.

Workloads. Table 4 shows the single-thread programs from the MSC suite [1], where MPKI denotes the number of read Misses in the last-level cache Per Kilo Instruction. Using the programs we generate 100 unique, random, 32-program workloads with a uniform distribution of the average MPKI per core from five to 25.

Scaling Method. The program address spaces are limited to 32 bits [1, 7]. Thus a 32-program workload would exercise only $32 + 5 = 37$ address bits. Since the baseline system has a 38-bit physical address space, the most-significant bit of all row addresses would be fixed (e.g., zero). To make the evaluation pessimistic for adaptive row addressing, we inflate the workload address space using the following method: we

insert $\Delta = w_{PA} - w_{WA}$ zero bits into the least-significant bits of the *row* field, where w_{PA} is the physical address space width and w_{WA} is the workload address space width. Thus, we insert one zero bit to scale the 37-bit workload address space up to 38 bits. Note that for random virtual-to-physical address mapping such scaling is not needed.

System-Level Metrics. We assess weighted speedup [10], execution energy, and fairness [10]. Weighted speedup is given by $\sum_i (IPC_i^{MP} / IPC_i^{SP})$ for all programs i in the workload, where IPC_i^{MP} denotes the Instructions Per Cycle (IPC) of program i when it is executed in the workload, and IPC_i^{SP} when it is executed alone. Since we consider performance loss compared to DDR^{id} , we express it as *normalized weighted slowdown*, i.e., as the weighted speedup of DDR^{id} over that of two-cycle or adaptive row addressing. Execution energy is estimated by the USIMM system energy model with our extensions described above. Fairness is estimated as the maximum slowdown across the programs in the workload, given by $\max_i (IPC_i^{SP} / IPC_i^{MP})$.

Non-System-Level Metrics. We consider the read latency and *Address-Cache Miss rate (ACM)*, estimated as the percentage of two-cycle row addresses among all row addresses transferred. Each metric is averaged across the channels.

5. EXPERIMENTAL RESULTS

We evaluate two-cycle row addressing (*A2*) and adaptive row addressing with various address-caching schemes and schedulers built on FRFCFS and BLISS (*APP* adds suffix -A and row-address prefetching adds suffix -P). Performance, energy, and fairness results are normalized to those of DDR^{id} . Next, Section 5.1 presents the main evaluation and Section 5.2 the sensitivity analysis.

5.1 Main Evaluation

5.1.1 Benefit of APP

Figure 12 presents the ACM and fairness of A2 and R-1 employing the four possible versions of FRFCFS: baseline, FRFCFS-A (with APP), FRFCFS-P (with row-address prefetching), and FRFCFS-AP (with both APP and

¹¹This address extension is pessimistic for adaptive row addressing compared to the default address extension of USIMM, that inserts core-ID bits into the most-significant bits of the *row* field [7].

Figure 12: Cooperation of APP and row-address prefetching for best efficiency of FRFCFS and R-1

prefetching). The box plots [36] summarize the results for each configuration across 100 workloads from Section 4.

Figure 12a shows that FRFCFS-A slightly reduces the ACM spread compared to FRFCFS. This is because APP prioritizes some of the one-cycle row addresses. FRFCFS-P further reduces the ACM spread and mean. However, the lowest ACM is achieved by FRFCFS-AP, that combines APP and row-address prefetching.

Figure 12b shows that the ACM improvements correlate well with fairness gains. FRFCFS-P reduces the ACM but increases the probability of the negative impact of row-address caching on the request service order. APP eliminates the negative impact, and so FRFCFS-AP achieves the best fairness for R-1. Recall also that APP eliminates the negative impact in both single- and multi-program workloads. Thus, APP is a key technique to exploit the full potential of adaptive row addressing via cooperation with row-address prefetching. For brevity, we further present results only with APP.

5.1.2 Performance of FRFCFS-A

FRFCFS-A employs APP but not row-address prefetching. Figure 13 presents normalized weighted slowdown of A2, R-1, and various schemes with row-address caches of three to 63 entries: direct-mapped ($D-3$ to $D-63$), 2-way ($W-3$ to $W-63$), and fully-associative ($F-3$ to $F-63$). The $W-*$ and $F-*$ caches are LRU-managed. Figure 13 shows that the performance gap between A2 and DDR^{id} (the 1.00 guide line) is about 5.5% on average and up to 6.5%. The gap is significant because: i) the opportunity to overlap address-transfer cycles with bank-busy cycles is limited, and ii) additional address-transfer cycles interfere with other commands on the CA bus.

Figure 13 shows that FRFCFS-A significantly improves performance: the row-address caches with 15 or more entries perform within 1% of DDR^{id} . Counter-intuitively, for some workloads the MCA protocol performs even better than DDR^{id} (see the points below the 1.00 guide line). We find that FRFCFS-A significantly reduces the number of two-cycle row addresses and the positive impact of the remaining two-cycle addresses on the request service order outweighs the overhead of the extra address-transfer cycles.

Next, Figure 13 shows that the $W-*$ row-address caches perform almost as good as the respective $F-*$ caches. For instance, both $W-31$ and $F-31$ achieve performance within 0.5% of DDR^{id} . Hence, there is no clear benefit from associativity above two.

Figure 13: FRFCFS-A – Performance

Figure 14: FRFCFS-AP – Performance

5.1.3 Performance of FRFCFS-AP

FRFCFS-AP employs both APP and row-address prefetching. Figure 14 shows that the latter further improves performance. For instance, R-1 in Figure 14 outperforms D-3 in Figure 13. Likewise, D-15, W-15, and F-15 in Figure 14 respectively outperform D-63, W-31, and F-31 in Figure 13. Thus, FRFCFS-AP outperforms FRFCFS-A with two or more times larger row-address caches. In other words, the benefit of row-address prefetching exceeds the benefit of doubling the address-cache size. We consider 31-entry row-address caches as a reasonable design point. Although smaller caches also perform well, they lack robustness, as we discuss in Section 5.2.

5.1.4 Detailed Results for FRFCFS-A(P)

We find that the read-request latency of A2 is longer than that of DDR^{id} by 7.5% on average and up to 12.5% (Figure 3). FRFCFS-A and FRFCFS-AP improve the read latency in much the same way as they improve the system performance in Figures 13 and 14. For brevity, we omit the read-latency plots.

Figure 15 presents detailed results for A2 and D-31, W-31, F-31 using FRFCFS-A vs. D-15, W-15, F-15 using FRFCFS-AP. Figure 15a shows that row-address prefetching significantly reduces the ACM spread: the 15-entry row-address caches using FRFCFS-AP achieve a two times smaller ACM spread compared to the respective 31-entry caches using FRFCFS-A. Figures 15b, 15c and 15d show that thanks to the smaller ACM spread, the 15-entry caches outperform the respective 31-entry caches in terms of system-level performance, execution energy, and fairness. This emphasizes the benefit of row-address prefetching.

Figure 15c shows that F-31, W-15, and F-15 have lower system-level execution energies than DDR^{id} for half of the workloads (the medians rest on the 1.00 guide line). The energy-efficiency gain over DDR^{id} is due to: 1) the narrower CA bus and 2) the positive impact of row-address caching on the request service order.

The fairness results in Figure 15d are similar to the results in Figures 15b and 15c, though the spread is larger. Still, FRFCFS-AP closes the fairness gap between A2 and DDR^{id} from 5% on average to less than 1%.

Figure 15: FRFCFS-A and 31-entry caches vs. FRFCFS-AP and 15-entry caches

5.1.5 Results for BLISS-A and BLISS-AP

We find that the normalized results for BLISS-A(P) are very similar to the respective results for FRFCFS-A(P), and so we omit them for brevity. Despite that BLISS truncates long sequences of row-buffer hits (to improve fairness) [34] and thus can cause more ACTs, adaptive row addressing is still effective. Although BLISS is more fair than FRFCFS, it is vulnerable to the negative impact of row-address caching in single- and multi-program workloads. Thus, BLISS needs APP to eliminate the negative impact.

5.2 Sensitivity Analysis

5.2.1 OS Page Size

The most significant bits of the *page offset* field of the virtual address map to the *block* field of the DRAM address (Figure 5). Thus a larger OS page size could potentially reduce row-buffer miss rates. To separate the physical address spaces of the programs in the workload, we generate unique, random bits per page (Section 4). A larger OS page size means fewer pages and thus it could reduce the ACM as well. However, we observe that the results for 8-KB pages are very similar to those for 4-, 16-, and 32-KB pages. Thus, the OS page size is not a major factor in our evaluation.

5.2.2 Page Coloring

The OS can implement page coloring such that each core (program) gets its own bank [27, 25]. Such page coloring separates the physical address spaces of the programs by adding core-ID bits right after the *block* field of the DRAM address. This reduces bank-level interference among co-running programs, improving the efficiency of adaptive row addressing.

Figure 16 shows that under page coloring adaptive row addressing is very effective even without row-address prefetching: the ACM in Figure 16a is less than 2%, and the performance, energy, and fairness of FRFCFS-A are very similar to those of FRFCFS-AP. Figure 16d shows that A2 attains better fairness than DDR^{id} for some workloads (the bottom whisker crosses the 1.00 guide line). Though, the spread is large and for some workloads fairness is lower by almost 10%. On the contrary, adaptive row addressing significantly reduces the spread and attains nearly the same fairness as DDR^{id} for all of the workloads except a few outliers.

5.2.3 Random Virtual-to-Physical Address Mapping

The default USIMM virtual-to-physical address mapping, employed so far, exhibits an intermediate amount of row-

Figure 16: Page coloring applied to FRFCFS-A(P) and W-31

Figure 17: Random virtual-to-physical address mapping applied to FRFCFS-A(P) and W-31

address locality expected in real systems. Figure 17 shows that random virtual-to-physical address mapping degrades the efficiency of adaptive row addressing across the metrics. Such randomization destroys locality, increasing both row-buffer miss rates and the ACM. Thus it can be considered the worst case for adaptive row addressing.

However, Figure 17a shows that row-address prefetching manages to reduce the ACM of W-31 from about 50% on average down to less than 20% on average. Thanks to that W-31 with FRFCFS-AP reduces the gap between A2 and DDR^{id} to 1.5% on average, as Figures 17b to 17d show. We omit the results for row-address caches smaller than W-31 since they are less robust and perform poorly under random virtual-to-physical address mapping.

Figure 18: 11-bit MSPs applied to FRFCFS-A(P) and W-31

5.2.4 Wider MSPs

The encoding efficiency of a row-address cache of a fixed size decreases as the MSP width increases. Thus, wider MSPs make it more challenging for adaptive row addressing to close the gap between A2 and DDR^{id} . We assume that MSPs can be wider due to one or more of the following reasons: i) the continuous growth of memory die densities [16] can imply more rows per bank, requiring wider row addresses; ii) techniques like *rank multiplication* [14, 17] use row-address bits for addressing additional ranks; iii) should the rows become smaller in the future, wider row addresses will be required to address the same bank capacity; and iv) the alternative optimization from Section 2.3, that reassigns three row-address pins, would push one more bit out to the MSP. To stress adaptive row addressing, we assume a 16 times larger capacity and therefore 4-bit wider MSPs. We employ the scaling method from Section 4 to appropriately scale the 37-bit workload address space up to the 42-bit physical address space.

Figure 18 shows the results for W-31 using FRFCFS-A(P). The MSP is 11 bits, and the encoding efficiency of W-31 is 16 times worse than that of W-31 in Figure 6. Figure 18a shows that the ACM of W-31 without row-address prefetching is high, about 40% on average. However, W-31 with row-address prefetching achieves an average ACM of only 10%. This brings adaptive row addressing within 1% of DDR^{id} in terms of the average values of the system-level metrics (Figures 18b to 18d). Smaller row-address caches are less robust than W-31 and perform poorly under wide MSPs.

6. RELATED WORK

Multi-Part and Multi-Cycle Addressing. LPDDR3 [3] has a 10-pin CA bus and transfers each row address in two parts in one cycle (the CA bus is DDR), which is practical in small memories, where the number of devices (loads) on the CA bus is small. Currently LPDDR3 is being replaced by LPDDR4 [4], that has a 6-pin CA bus and transfers each row address in four parts in four cycles (the CA bus is SDR to support higher speeds). The protocols are optimized for relatively small memories. On the contrary, this paper considers future parallel protocols for large memories and proposes adaptive row addressing to boost their efficiency.

Large-Capacity Memory Modules. LRDIMMs can offer larger module capacities than RDIMMs operating at the

same speed. LRDIMMs implement rank multiplication, that uses row-address bits to address additional ranks [14, 17]. Thus, rank multiplication assumes that some of the row-address bits are unused, i.e., that the memory-device capacity is smaller than the maximum capacity defined by the respective standard. Main memories that employ future LRDIMMs designed for MCA protocols would benefit from the proposed adaptive row addressing.

Address Caching. The idea of address compression for off-chip transmission [31, 11] has been generalized to reduce total off-chip traffic [9], and multiple later works apply it to on-chip transmission. Unlike prior work, this paper: 1) removes the address-cache update logic from the critical path of the MC, i.e., it mirrors the update logic on the memory-device side, so that the MC does not need to send explicit address-cache update commands upon a miss; 2) proposes address-caching schemes with $(2^n - 1)$ -entry, 2-way caches where one way of one set is disabled for high encoding efficiency; and 3) studies row-address caching for large-capacity off-chip memories in contemporary multicore systems with state-of-the-art memory-request schedulers.

Prefetching. Although prefetching is well-studied in other contexts, to the best of our knowledge we are the first to propose row-address prefetching for off-chip memories.

Memory Scheduling. There exists a massive body of work about memory-request scheduling, and the first-ready policy [38, 32] is commonly used to reorder row accesses. However, prior work assumes a fixed number of cycles per row-address transfer. On the contrary, this paper tackles a variable number of cycles per row-address transfer and proposes an adaptive row-access priority policy. The proposed policy eliminates the negative impact of row-address caching in both single- and multi-program workloads and retains its positive impact.

7. CONCLUSION

Cost-efficient yet high-speed parallel memory protocols with multi-cycle addressing are key to large-capacity and still low-latency main memories. We propose the concept of adaptive row addressing as a general approach to close the efficiency gap between such protocols and an idealistic protocol of the same speed but with enough pins for single-cycle addressing. Adaptive row addressing comprises: 1) row-address caching, 2) row-address prefetching, and 3) adaptive row-access priority policy. This paper shows that adaptive row addressing robustly closes the efficiency gap by boosting system-level performance, energy efficiency, and fairness up to the level of the idealistic protocol and in some cases slightly above it.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments on the earlier versions of the paper. We also thank Alen Bardizbanyan and Madhavan Manivannan for their feedback. This work was funded by the European Union under the FP7 project EUROSERVER (No: 610456).

9. REFERENCES

- [1] Memory scheduling championship 2012. www.cs.utah.edu/~rajeew/jwac12.
- [2] DDR4 SDRAM JEDEC standard. www.jedec.org, Sept. 2012.
- [3] Low Power Double Data Rate 3 (LPDDR3) JEDEC standard. www.jedec.org, Aug. 2015.
- [4] Low Power Double Data Rate 4 (LPDDR4) JEDEC standard. www.jedec.org, Nov. 2015.
- [5] B. Akin, F. Franchetti, and J. C. Hoe. Data reorganization in memory using 3D-stacked DRAM. In *Proc. Int. Symp. on Computer Architecture*, pages 131–143, June 2015.
- [6] G. Allan. The past, present and future of DDR4 memory interfaces. Synopsys Insight Newsletter, www.synopsys.com, 2012.
- [7] N. Chatterjee, R. Balasubramanian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti. USIMM: the Utah Simulated Memory Module. University of Utah, Tech. Rep. UUCS-12-002, Feb. 2012.
- [8] J. Choi, W. Shin, J. Jang, J. Suh, Y. Kwon, Y. Moon, and L.-S. Kim. Multiple clone row DRAM: A low latency and area optimized DRAM. In *Proc. Int. Symp. on Computer Architecture*, pages 223–234, June 2015.
- [9] D. Citron and L. Rudolph. Creating a wider bus using caching techniques. In *Proc. Int. Symp. on High Performance Computer Architecture*, pages 90–99, Jan. 1995.
- [10] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *MICRO*, 28(3):42–53, May 2008.
- [11] M. Farrens and A. Park. Dynamic base register caching: A technique for reducing address bus width. In *Proc. Int. Symp. on Computer Architecture*, pages 128–137, May 1991.
- [12] M. Gokhale, S. Lloyd, and C. Macaraeg. Hybrid memory cube performance characterization on data-centric workloads. In *Proc. Workshop on Irregular Applications: Architectures and Algorithms*, pages 1–8, Nov. 2015.
- [13] Hybrid Memory Cube Consortium. Hybrid memory cube specification 2.1. www.hybridmemorycube.org, Oct. 2015.
- [14] Inphi Corp. Introducing LRDIMM – a new class of memory modules. White Paper, www.inphi.com, 2011.
- [15] Int. Technology Roadmap for Semiconductors. Assembly and Packaging. Tables, www.itrs.net, 2012.
- [16] Int. Technology Roadmap for Semiconductors. Process Integration, Devices, and Structures. www.itrs.net, 2013.
- [17] Integrated Device Technology, Inc. DDR4 LRDIMMs for both memory capacity and speed. White Paper, www.idt.com, 2014.
- [18] Intel Corp. Intel[®] Xeon[®] processor E7-8890 v3. ark.intel.com. Accessed: Mar. 2016.
- [19] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. 2007.
- [20] JEDEC. DDR4 SDRAM load reduced DIMM design specification. www.jedec.org, Sept. 2014.
- [21] JEDEC. DDR4 SDRAM load reduced DIMM design specification, Annex B – Raw Card B. www.jedec.org, Sept. 2014.
- [22] JEDEC. DDR4 SDRAM UDIMM design specification. www.jedec.org, Aug. 2014.
- [23] JEDEC. DDR4 SDRAM registered DIMM design specification. www.jedec.org, Aug. 2015.
- [24] JEDEC. DDR4 SDRAM registered DIMM design specification, Annex F – Raw Card F. www.jedec.org, Aug. 2015.
- [25] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, pages 367–376, Sept. 2012.
- [26] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Syst. J.*, 9(2):78–117, June 1970.
- [27] W. Mi, X. Feng, J. Xue, and Y. Jia. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors. In *Proc. Int. Conf. on Network and Parallel Computing*, pages 329–343, Sept. 2010.
- [28] Micron Technology, Inc. DDR4 system power calculator, Oct. 2014.
- [29] Micron Technology, Inc. Micron[®] 4Gb: x4, x8, x16 DDR4 SDRAM features. Datasheet, www.micron.com, 2014.
- [30] P. J. Nair, C.-C. Chou, and M. K. Qureshi. Refresh pausing in DRAM memory systems. *ACM Trans. on Architecture and Code Optimization*, 11(1):10:1–10:26, Feb. 2014.
- [31] A. Park and M. Farrens. Address compression through base register caching. In *Proc. Int. Symp. on Microarchitecture*, pages 193–199, Nov. 1990.
- [32] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens. Memory access scheduling. In *Proc. Int. Symp. on Computer Architecture*, pages 128–138, June 2000.
- [33] T. Schmitz. The rise of serial memory and the future of DDR. White Paper, www.xilinx.com, 2014.
- [34] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu. The blacklisting memory scheduler: Achieving high performance and fairness at low cost. In *Proc. Int. Conf. on Computer Design*, pages 8–15, Oct. 2014.
- [35] V. K. Tavva, R. Kasha, and M. Mutyam. EFGR: An enhanced fine granularity refresh feature for high-performance DDR4 DRAM devices. *ACM Trans. on Architecture and Code Optimization*, 11(3):31:1–31:26, Oct. 2014.
- [36] J. W. Tukey. *Exploratory Data Analysis*. 1977.
- [37] Xilinx, Inc. UG576 ultrascale architecture GTH transceivers. User Guide, www.xilinx.com, Nov. 2015.
- [38] W. Zuravleff and T. Robinson. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. Patent 5630096, May 1997.