A Performance & Power Comparison of Modern High-Speed DRAM Architectures

Shang Li
University of Maryland, College Park
shangli@umd.edu

Dhiraj Reddy
University of Maryland, College Park
dhiraj@umd.edu

Bruce Jacob
University of Maryland, College Park
blj@umd.edu

ABSTRACT

To feed the high degrees of parallelism in modern graphics processors and manycore CPU designs, DRAM manufacturers have created new DRAM architectures that deliver high bandwidth. This paper presents a simulation-based study of the most common forms of DRAM today: DDR3, DDR4, and LPDDR4 SDRAM; GDDR5 SGRAM; and two recent 3D-stacked architectures: High Bandwidth Memory (HBM1, HBM2), and Hybrid Memory Cube (HMC1, HMC2). Our simulations give both time and power/energy results and reveal several things: (a) current multi-channel DRAM technologies have succeeded in translating bandwidth into better execution time for all applications, turning memory-bound applications into compute-bound; (b) the inherent parallelism in the memory system is the critical enabling factor (high bandwidth alone is insufficient); (c) while all current DRAM architectures have addressed the memory-bandwidth problem, the memory-latency problem does still remain, dominated by queuing delays arising from lack of parallelism; and (d) the majority of power and energy is spent in the I/O interface, driving bits across the bus; DRAM-specific overhead beyond bandwidth has been reduced significantly, which is great news (an ideal memory technology would dissipate power *only* in bandwidth, all else would be free).

CCS CONCEPTS

• **General and reference** → **Performance**; • **Computer systems organization** → *Architectures*;

KEYWORDS

Memory Systems, DRAM Architectures, Cycle-accurate Simulation

ACM Reference Format:
Shang Li, Dhiraj Reddy, and Bruce Jacob. 2018. A Performance & Power Comparison of Modern High-Speed DRAM Architectures. In *The International Symposium on Memory Systems (MEMSYS), October 1–4, 2018, Old Town Alexandria, VA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3240302.3240315

1 INTRODUCTION

In response to the still-growing gap between memory access time and the rate at which processors can generate memory requests [46, 60], and especially in response to the growing number of on-chip cores (which only exacerbates the problem), manufacturers have created several new DRAM architectures that give today's system designers a wide range of memory-system options from low power, to high bandwidth, to high capacity. Many are multi-channel internally. This paper presents a simulation-based characterization of the most common DRAMs in use today, evaluating each in terms of its effect on total execution time and power dissipation.

We have updated DRAMsim2 [50] to simulate nine modern DRAM architectures: DDR3 [24], DDR4 [25], LPDDR3 [23], and LPDDR4 SDRAM [28]; GDDR5 SGRAM [29]; High Bandwidth Memory (both HBM1 [26] and HBM2 [27]); and Hybrid Memory Cube (both HMC1 [18] and HMC2 [19]). The DRAM command timings are validated, and the tool provides power and energy estimates for each architecture. To obtain accurate memory-request timing for a contemporary multicore out-of-order processor, we integrate our code into *gem5* and use its DerivO3 CPU model [3]. To highlight the differences inherent to the various DRAM protocols, we study single-channel (and single-package, for those that are multi-channeled within package) DRAM systems. Doing so exposes the fundamental behaviors of the different DRAM protocols & architectures that might otherwise be obscured in, for example, extremely large, parallel systems like Buffer-on-Board [9] or Fully Buffered DIMM [14] systems.

This study asks and answers the following questions:

- Previous DRAM studies have shown that the memory overhead can be well over 50% of total execution time (e.g., [10, 11, 53]); what is the overhead today, and how well do the recent DRAM architectures combat it? In particular, how well do they address the memory-latency and memory-bandwidth problems?
  As our results show, main memory overheads today, for single-rank organizations, are still 42–75% for nearly all applications, even given the relatively modest 4-core system that we study. However, when sufficient parallelism is added to the memory system to support the bandwidth, which can be as simple as using a dual-rank organization, this overhead drops significantly. In particular, the latest high-bandwidth 3D stacked architectures (HBM and HMC) do well for nearly all applications: these architectures reduce the memory-stall time significantly over single-rank DDRx and LPDDR4 architectures, reducing 42–75% overhead down to less than 30% of total execution time. These architectures combine into a single package all forms of parallelism in the memory system: multiple channels, each with multiple ranks/banks. The most important effect of these and other highly parallel architectures is to turn many memory-bound applications

MEMSYS, October 1–4, 2018, Old Town Alexandria, VA, USA

ACM ISBN 978-1-4503-6475-1/18/10.
https://doi.org/10.1145/3240302.3240315

to compute-bound applications, and the total execution time for some applications can be cut by factors of 2–3x.

- Where is time and power spent in the DRAM system?
  For all architectures but HBM and HMC, the majority of time is spent waiting in the controller's queues; this is true even though the workloads represent only a small number of cores. Larger systems with dozens or hundreds of cores would tend to exacerbate this problem, and this very phenomenon is seen, for example, in measured results of physical KNL systems [48]. For HBM and HMC systems, the time is more evenly distributed over queuing delays and internal DRAM operations such as row activation and column access.
  Power breakdowns are universal across the DRAM architectures studied: for each, the majority of the power is spent in the I/O interface, driving bits over the bus. This is an extremely good thing, because everything else is overhead, in terms of power; this result means that one pays for the bandwidth one needs, and the DRAM operations come along essentially for free. The most recent DRAMs, HMC especially, have been optimized internally to the point where the DRAM-specific operations are quite low, and in HMC represent only a minor fraction of the total. In terms of power, DRAM, at least at these capacities, has become a pay-for-bandwidth technology.
- How much locality is there in the address stream that reaches the primary memory system?
  The stream of addresses that miss the L2 cache contains a significant amount of locality, as measured by the hit rates in the DRAM row buffers. The hit rates for the applications studied range 0–90% and average 39%, for a last-level cache with 2MB per core. (This does not include hits to the row buffers when making multiple DRAM requests to read one cache line.) This relatively high hit rate is why optimized close-page scheduling policies, in which a page is kept open if matching requests are already in the controller's request queue (e.g., [30, 47]), are so effective.

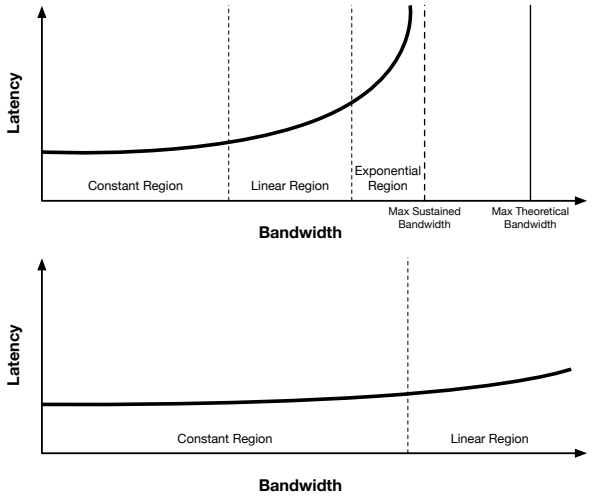


Figure 1: Top: as observed by Srinivasan [54], when plotting system behavior as latency per request vs. actual bandwidth usage (or requests per unit time), three distinct regions appear. At low bandwidth usage, latency is nearly constant. As bandwidth usage increases, the latency increases linearly. In high-performance systems, the maximum *sustainable* bandwidth is usually 75–80% of the maximum *theoretical* bandwidth. As an application's request rate approaches this value, the requests are arriving so frequently it strains the system's capacity, and latencies grow extremely high, extremely quickly. Bottom: when a second memory system with a much higher maximum bandwidth is evaluated running the same workload, the second system (bottom graph) will exhibit constant latency in regions where the lower-bandwidth system (top graph) experiences linear or even exponential latency.

In addition, we make several observations. First, "memory latency" and "DRAM latency" are two completely different things. *Memory latency* corresponds to the delay software experiences from issuing a load instruction to getting the result back. *DRAM latency* is often a small fraction of that: average memory latencies for DDRx and LPRDDRx systems are in the 80–100ns range, whereas typical DRAM latencies are in the 15–30ns range. The difference is in arbitration delays, resource management, and whether sufficient parallelism exists in the memory system to support the memory traffic of the desired workload. Insufficient parallelism leads to long queuing delays, with requests sitting in the controller's request buffers for tens to hundreds of cycles. If your memory latency is bad, it is likely not due to DRAM latency.

This is not a new concept. As has been shown before [10], more bandwidth is not always better, especially when it is allocated without enough concurrency in the memory system to maintain it. Execution time is reduced 21% when moving from single-rank DDR3 channels to dual-rank channels. Execution time is reduced 22% when moving from a single-channel LPDDR4 organization to a quad-channel organization. Execution time is reduced 25% for some apps when moving from a 4-channel organization of HBM to an 8-channel organization. And when one looks at the reason for the reduction, it is due to reduced time spent in queues waiting for memory resources to become free. Though it may sound counter-intuitive, average *latencies* decrease when one allocates enough *parallelism* in the memory system to handle the incoming request stream. Otherwise, requests back up, and queuing delays determine the average latency, as we see in DDRx, LPDDR4, and GDDR5 based systems. Consequently, if one's software is slow due to latency issues, consider improving your NoC, or increasing the number of controllers or channels to solve the problem.

Second, bandwidth is a critical and expensive resource, so its allocation is important. As mentioned above, having enough bandwidth with parallelism to support it can reduce execution time by 2–3x and turn some previously memory-bound apps into compute-bound apps. This is a welcome result: one can bring value to advanced processor-architecture design by simply spending *money* on the memory system. Critical rule of thumb to note: multicore/manycore architectures require at a minimum ~1GB/s of sustained memory bandwidth per core, otherwise the extra cores sit idle [54].

Third, for real-time systems, Hybrid Memory Cube is quite interesting, as it provides *highly* deterministic latencies. This characteristic of HMC has been noted before [46] and is due in part to the architecture's extremely high bandwidth, which pushes the exponential latency region out as far as possible (see Figure 1 for an illustration). However, high bandwidth alone does not provide such determinism, otherwise we would see similar deterministic latencies in HBM systems, which we do not. The effect is due not only to bandwidth but also to the internal scheduling algorithms of HMC, which use a close-page policy that does not opportunistically seek to keep a page open longer than required for the immediate request. While this may sacrifice some amount of performance, it provides predictable latencies and keeps the internal DRAM power down to a level below that of all other DRAM architectures studied, including power-optimized LPDDR4.

The following sections provide more background on the topic, describe our experimental setup, and compare & contrast the various DRAM architectures.

2 RELATED WORK

Many proposals have quantified the performance benefits of having a lower latency or higher bandwidth main memory. For instance, [35, 39, 42, 52] showcase the performance improvement obtainable by having a good fraction of memory requests served from on-package memories such as HBM.

Some studies [32, 33, 45] propose main memory system designs with alternative but not yet mainstream technologies such as Phase Change Memory (PCM) and Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM). These contrast the trade-offs between the emerging memory technologies and traditional DDR memory.

Some papers study the system-level performance impact caused by inadequately provisioned memory systems. Wang et al. [59] show the importance of providing proportionate and sufficient memory bandwidth for multi-threaded applications running on large multi-core systems. Subramanian et al. [55] measure the performance degradation caused due to interference in application memory access streams. Liu et al. [37] propose OS level techniques to mitigate performance degradation caused by memory interference in multi-core systems. Shingari et al. [51] and Narancic et al. [41] discuss the memory system behavior in the context of mobile workloads and LPDDR memory. Lee et al. [34] discuss the memory bandwidth challenges in implementing security features in multi-core processors. Abts et al. [1] discuss the impact of the physical location of the memory controllers on a multi-core chip on memory access latencies.

Designing memory schedulers that are fast and fair has long been an active area of research. Rixner et al. [47] perform an expansive design-space exploration of scheduling heuristics, analyzing various request reordering and scheduling strategies, row buffer open/close policies, etc. Yuan et al. [61] describe a similar study in the context of a GPU memory system. Kaseridis et al. [30] demonstrate a best-of-both-worlds heuristic that blends open page and close page, keeping pages open slightly longer than close-page, but not much longer. Cooper-Balis [8] and Chatterjee et al. [6] both argue for shortening the size of the DRAM row buffers to reduce

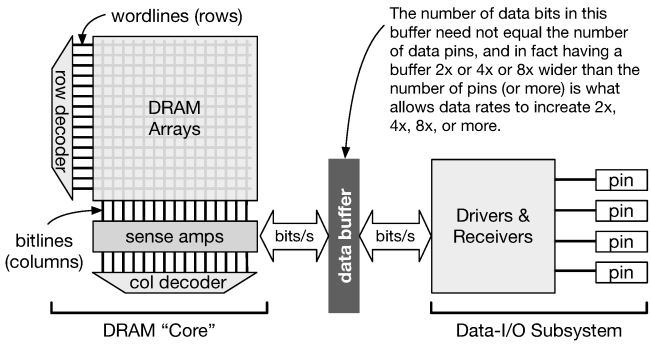


Figure 2: Stylized DRAM internals, showing the importance of the data buffer between DRAM core and I/O subsystem. Increasing the size of this buffer, i.e., the fetch width to/from the core, has enabled speed increases in the I/O subsystem that do not require commensurate speedups in the core.

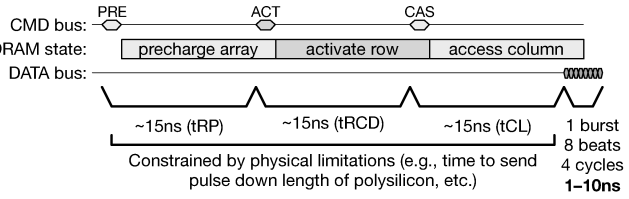


Figure 3: DRAM read timing, with values typical for today. The burst delivery time is not drawn to scale: it can be a *very* small fraction of the overall latency. *Note:* though precharge is shown as the first step, in practice it is performed at the end of each request to hide its overhead as much as possible, leaving the array in a precharged state for the next request.

memory power. Li et al. [36] propose prioritizing memory requests corresponding to cache lines shared by multiple cores. Chatterjee et al.[7] showcase the importance of prioritizing the memory requests from a GPU warp and to serve them together. Several proposals [5, 16, 63] discuss strategies to reduce the performance and energy impact associated with DRAM refresh.

Memory-system energy consumption is a growing concern in today's datacenters, as characterized in [56, 57]. Zhang et al. [62] further demonstrate the scale of this problem and propose ideas to partially power-off a part of the DRAM main memory to reduce standby power.

This paper revisits earlier comparative studies of DRAM architectures (e.g., [11, 12, 44]) and is the first to pit today's DRAMs (LP/DDR, GDDR, HBM, HMC) against each other.

3 BACKGROUND

Dynamic Random Access Memory (DRAM) uses a single transistor-capacitor pair to store each bit. A simplified internal organization is shown in Figure 2, which indicates the arrangement of rows and columns within the DRAM arrays and the internal core's connection to the external data pins through the I/O subsystem.

The use of capacitors as data cells has led to a relatively complex protocol for reading and writing the data, as illustrated in Figure 3. The main operations include *precharging* the bitlines of an array, *activating* an entire row of the array (which involves *discharging* the row's capacitors onto their bitlines and *sensing* the voltage changes on each), and then *reading/writing* the bits of a particular subset (a *column*) of that row [20].

Previous studies indicate that increasing DRAM bandwidth is far easier than decreasing DRAM latency [4, 10–12, 20], and this is because the determiners of DRAM latency (e.g., precharge, activation, and column operations) are tied to physical constants such as the resistivity of the materials involved and the capacitance of the storage cells and bitlines. Consequently, the timing parameters for these operations are measured in nanoseconds and not clock cycles; they are independent of the DRAM's external command and data transmission speeds; and they have only decreased by relatively small factors since DRAM was developed (the values have always been in the tens of nanoseconds).

The most significant changes to the DRAM architecture have come in the data interface, where it is easier to speed things up by designing low-swing signaling systems that are separate from the DRAM's inner core [44]. The result is the modern DDR architecture prevalent in today's memory systems, in which the interface and internal core are decoupled to allow the interface to run at speeds much higher than the internal array-access circuitry. The organization first appeared in JEDEC DRAMs at the first DDR generation, which introduced a *2n prefetch* design that allowed the internal and external bandwidths to remain the same, though their effective clock speeds differed by a factor of two, by "prefetching" twice the number of bits out of the array as the number of data pins on the package. The DDR2 generation then doubled the prefetch bits to 4x; the DDR3 generation doubled it to 8x, and so on. This is illustrated in Figure 2, which shows the decoupling data buffer that lies between the core and I/O subsystem. The left side of this buffer (the core side) runs slow and wide; the right side (the I/O side) runs fast and narrow; the two bandwidths are equal.

This decoupling has allowed the DRAM industry to focus heavily on improving interface speeds over the past two decades. As shown in Figure 3, the time to transmit one burst of data across the bus between controller and DRAM is measured in cycles and not nanoseconds, and, unlike the various operations on the internal core, the absolute time for transmission *has* changed significantly in recent years. For instance, asynchronous DRAMs as recent as the 1990s had bus speeds in the range of single-digit Mbps per pin; DDR SDRAM appeared in the late 1990s at speeds of 200 Mbps per pin, two orders of magnitude faster; and today's GDDR5X SGRAM speeds, at 12 Gbps per pin, are another two orders of magnitude faster than that. Note that every doubling of the bus speed reduces the burst time by a factor of two, thereby exacerbating the already asymmetric relationship between the data-access protocol (operations on the left) and the data-delivery time (the short burst on the right).

The result is that system designers have been scrambling for years to hide and amortize the data-access overhead, and the problem is never solved, as every doubling of the data-bus speed renders the access overhead effectively twice as large. This has put pressure in two places:

- **The controller design**. The controller determines how well one can separate requests out to use different resources (e.g., channels and banks) that can run independently, and also how well one can gather together multiple requests to be satisfied by a single resource (e.g., bank) during a single period of activation.
- **Parallelism in the back-end DRAM system**. The back-end DRAM system is exposed as a limitation when it fails to provide sufficient concurrency to support the controller. This concurrency comes in the form of parallel channels, each with multiple ranks and banks.

It is important for system design to balance application needs with resources available in the memory technology. As previous research has shown, not all applications can make use of the bandwidth that a memory system can provide [49], and even when an application *can* make use of significant bandwidth, allocating that resource without requisite parallelism renders the additional bandwidth useless [10]. In simpler terms, *more* does not immediately translate to *better*. This paper studies which DRAM architectures provide more, and which architectures do it better. The following sections describe the DRAM architectures under study in terms of their support for concurrency and parallel access.

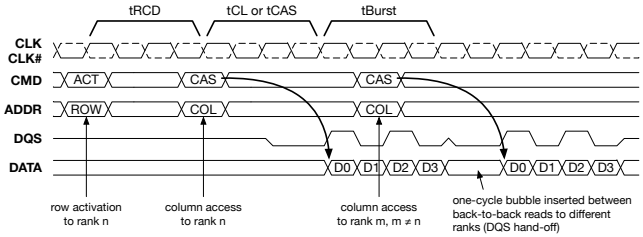


Figure 4: DDR SDRAM Read timing.

3.1 DDRx SDRAM

As mentioned above, the modern DDRx SDRAM protocol has become widespread and is based on the organization shown in Figure 2, which decouples the I/O interface speed from the core speed, requiring only that the two bandwidths on either side of the internal data buffer match. One of the distinguishing features of DDRx is its data transmission, which occurs on both edges of a data clock (double data rate, thus the name), the data clock named the *DQS* data-strobe signal. DQS is *source-synchronous*, i.e, it is generated by whomever is driving the data bus, and the signal travels in the same direction as the data. The signal is shown in Figure 4, which presents the timing for a read operation.

In our simulations, we use a DIMM organization as shown in Figure 5(a): a 64-bit data bus comprising eight x8 DRAMs.

3.2 LPDDRx SDRAM

Low Power DDR SDRAM makes numerous optimizations to achieve the same bandwidth as DDRx, in the same multi-drop organizations (e.g. multi-rank DIMMs), but at a significantly reduced power cost. Optimizations include removing the DLL, strict usage of the DQS strobe, and improved refresh.

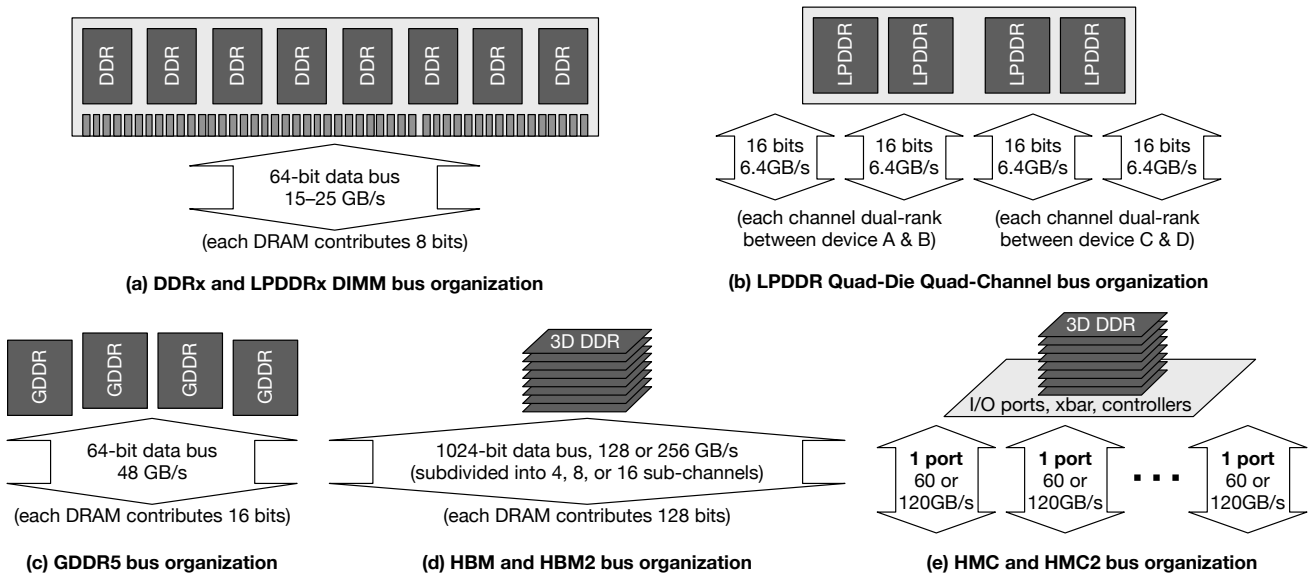


Figure 5: DRAM bus/interface organizations simulated in this study.

Another optimization for LPDDR4 is that each device is not only internally multi-banked, it is internally *multi-channel*[28]. Each device has two control/address buses, two data buses, and the specification describes the Quad-Die Quad-Channel Package: it has four dies and four separate buses, each 16 bits wide, with 16 bits coming from each of two devices in an overlapped, dual-rank configuration. This is an incredible amount of parallelism in a small, low-power package and approaches the parallelism (if not the bandwidth) of HBM.

In our simulations, we model LPDDR4 in two different ways that are common: first, we use a DIMM like that in Figure 5(a). Second, we simulate the the Quad-Die, Quad-Channel Package shown in Figure 5(b).

3.3 GDDR5 SGRAM

The DDRx standards have provided high bandwidth and high capacity to commodity systems (laptops, desktops), and the LPDDRx standards have offered similar bandwidths at lower power. These serve embedded systems as well as supercomputers and data centers that require high performance and high capacity but have strict power budgets.

The GDDRx SGRAM standards have been designed for graphics subsystems and have focused on even higher bandwidths than DDRx and LPDDRx, sacrificing channel capacity. SGRAMs are not specified to be packaged in DIMMs like the DDRx and LPDDRx SDRAMs. Each SGRAM is packaged as a wide-output device, typically coming in x16 or x32 datawidths, and they often require significant innovations in the interface to reach their aggressive speeds.

For example, GDDR5 runs up to 6Gbps per pin, GDDR5X is available at twice that, and the protocols require a new clock domain not seen in DDRx and LPDDRx standards: Addresses are sent at double-data-rate on the system clock, and the data strobe now runs a higher frequency than the system clock, as well as no longer being bi-directional. This has the beneficial effect of eliminating the dead bus cycle shown in Figure 4 as the "DQS hand-off," as the strobe line need not idle if it is never turned around. Instead of being source-synchronous, the data strobe is unidirectional and used by the DRAM for capturing data. For capturing data at the controller side during data-read operations, the controller trains each GDDR5 SGRAM separately to adjust its data timing at a fine granularity relative to its internal clock signal, so that the data for each SGRAM arrives at the controller in sync with the controller's internal data clock.

In our simulations, we use an organization as shown in Figure 5(c): a 64-bit data bus made from four x16 GDDR5 chips placed side-by-side.

3.4 High Bandwidth Memory (HBM)

JEDEC's High Bandwidth Memory uses 3D integration to package a set of DRAMs; it is similar to the DIMM package shown in Figure 5(d) in that it gathers together eight separate DRAM devices into a single parallel bus. The difference is that HBM uses through-silicon vias (TSVs) as internal communication busses, which enables *far* wider interfaces. Whereas a DDRx-based DIMM like that in Figure 5(a) gangs together eight x8 parts (each part has 8 data pins), creating a bus totaling 64 bits wide, HBM gangs together eight x128 parts, creating a bus totaling 1024 bits wide. This tremendous width is enabled by running the external communications over a silicon *interposer*, which supports wire spacing far denser than PCBs. This approach uses dollars to solve a bandwidth problem, which is always a good trade-off. JEDEC calls this form of packaging "2.5D integration."

The 8 channels of HBM can operate individually or cooperatively. HBM2 standard also introduced pseudo-channel, which further divide one channel into two pseudo channels.

In our simulations, we use the organization as shown in Figure 5(d): a 1024-bit data bus that is subdivided into four, eight, channels or sixteen pseudo-channels (8 for most studies). HBM1 gives 128GB/s total bandwidth; HBM2 gives 256GB/s total bandwidth.

3.5 Hybrid Memory Cube (HMC)

Hybrid Memory Cube is unique in that, unlike all the other DRAM architectures studied herein, the DRAM interface is not exported; instead, HMC packages internally its own DRAM controllers. As shown in Figure 5(e), it includes a 3D-integrated stack of DRAMs just like HBM, but it also has a non-DRAM logic die at the bottom of the stack that contains three important things:

(1) A set of memory controllers that control the DRAM. HMC1 has 16 internal controllers; HMC2 has up to 32.
(2) The interface to the external world: a set of two or four high-speed ports that are independent of each other and transmit a generic protocol, so that the external world need not use the DRAM protocol shown in Figure 3. Link speed and width can be chosen based on needs.
(3) An interconnect that connects the I/O ports to the controllers. Communication is symmetric: requests on any link can be directed to any controller, and back.

For most of the studies, we use two 15Gbps link configurations of HMC and HMC2 (120GB/s total bandwidth), because that is sufficient for the needs of the workload. At the end of the paper, we present a stress-test that significantly increases the memory-request rate, at which point we study 4-link and higher speed configurations as well.

4 EXPERIMENTAL SETUP

We run a newly updated version of DRAMsim [50, 58] within the gem5 simulator. The following sections elaborate.

Table 1: Gem5 Setup

CPU	Gem5 DerivO3 CPU model, x86 architecture, 4-core
Core	4GHz, Out-of-order, 8-fetch, 8-issue, 192 reorder buffer entries
L1 I-Cache	per-core, 32KB, 2-way associative, 64 Byte cache line, LRU
L1 D-Cache	per-core, 64KB, 2-way associative, 64 Byte cache line, LRU
L2 Cache	shared, MOESI protocol, 8MB, 8-way associative, 64 Byte cache line, LRU
Workloads	bzip2, gcc, GemsFDTD, lbm, mcf, milc, soplex, STREAM, GUPS, HPCG

4.1 Simulation Setup

We configure gem5 to simulate an average desktop processor: x86-based, 4-core, out-of-order. The detailed configuration is in Table 1. From several suites, we select benchmarks to exercise the memory system, including those from SPEC2006 [17] that are memory-bound according to [22]. These benchmarks have Cycles per Instruction (CPI) ranging from 2 to 14, representing moderate to relatively intensive memory workloads. We also simulate STREAM and GUPS from the HPCC benchmark suite [38]. STREAM tests the sustained bandwidth, while GUPS exercises the memory's ability to handle random requests. Finally we use HPCG [13], *high-performance conjugate gradients*, which represents memory-intensive scientific computing workloads. We ran four copies of each workload, one on each core of the simulated processor. Gem5 is configured to run in system-emulation mode, and all the benchmarks are fast-forwarded over the initialization phase of the program, and then simulated with the DerivO3 Gem5 CPU model for 2 billion instructions (500 million per core).

4.2 DRAM Simulator Overview

An updated version of DRAMsim2 models the new memory technologies studied herein.

As with DRAMsim2 [50], the simulator is validated against Verilog models for correctness. The following subsections provide some details of the simulator's inner workings.

Scheduling and Page Policy: A First-Ready–First-Come-First-Served (FR-FCFS) [47] scheduling policy combined with Minimalist Open-Page policy [30] is used in the memory controller design. FR-FCFS can reduce the latency and improve throughput by scheduling overlapped DRAM commands while Minimalist Open-Page prevents row-buffer starvation and thus improves fairness. We apply this scheme to all memory controllers except for HMC, because HMC only operates in strict close-page mode.

DRAM Address Mapping: To reduce row buffer conflicts and exploit parallelism among DRAM banks, we interleaved the DRAM addresses in the pattern of *row-bank-bankgroup-column* (from MSB to LSB). For configurations with multiple channels or ranks, we also interleaved the channel/rank bits in between the row-address bits and bank-ad-dress bits. Note that DDR3 has no bank group, and so this is ignored. Another exception: HMC enforces a close-page policy that does not take advantage of previously opened pages, and thus putting column address bits on the LSB side would not be beneficial. Therefore we adopt the address-mapping scheme recommended by the HMC specification, which is *row-column-bank-channel* (from MSB to LSB).

HMC Interface: Different from all other DRAM protocols, HMC uses high-speed links that transmit a generic protocol between the CPU and HMC's internal vault controllers.

The packets are broken down to flits to be sent across the internal crossbar, which has two layers: one for requests and another for responses, to avoid deadlocks.

Refresh Policy: All of the DRAM protocols simulated in this work use a per-rank auto-refresh scheme—that is, a refresh command is issued automatically by the controller to all the banks within a rank at the specified rate.

DRAM Parameters:

Several of the most important parameters are listed in Table 2, including tRCD, tRAS, tRP, and tCL/CWL.

Table 2: DRAM Parameters

DRAM Type	Density	Device Width	Page Size	# of Banks (per rank)	Pin Speed	Max. Bandwidth [3]	tRCD (ns)	tRAS (ns)	tRP (ns)	CL/CWL (ns)
DDR3	8Gb	8 bits	2KB	8	1.866Gbps	14.9GB/s	14	34	14	14/10
DDR4	8Gb	8 bits	1KB	16	3.2Gbps	25.6GB/s	14	33	14	14/10
LPDDR4	6Gb	16 bits	2KB	8	3.2Gbps	25.6GB/s	_[5]	_[5]	_[5]	_[5]
GDDR5	8Gb	16 bits	2KB	16	6Gbps	48GB/s	14/12[4]	28	12	16/5
HBM[1]	4Gbx8	128 bits	2KB	16	1Gbps	128GB/s	14	34	14	14/4
HBM2[1]	4Gbx8	128 bits	2KB	16	2Gbps	256GB/s	14	34	14	14/4
HMC[1]	2Gbx16	32 bits	256 Bytes	16	2.5Gbps[2]	120GB/s	14	27	14	14/14
HMC2[1]	2Gbx32	32 bits	256 Bytes	16	2.5Gbps[2]	320GB/s	14	27	14	14/14

[1] HBM and HMC have multiple channels per package, therefore the format here is channel density x channels.
[2] The speed here is HMC DRAM speed, simulated as 2.5Gbps according to [49]. HMC link speed can be 10–30Gbps.
[3] Bandwidths for DDR3/4, LPDDR4 and GDDR5 are based on 64-bit bus design; HBM and HBM2 are 8×128 bits wide; Bandwidth of HMC and HMC2 are maximum link bandwidth of all 4 links. We use 2 links 120GB/s in most simulations.
[4] GDDR5 has different values of tRCD for read and write commands.
[5] We are using numbers from a proprietary datasheet, and they are not publishable.

Most of the parameters are based on existing product datasheets or official specifications. Some parameters, however, are not publicly available—for example, some timing parameters of HBM and HMC are not specified in publicly available documentation. Previous studies [31, 49] have established reasonable estimations of such parameters, and so we adopt the values given in these studies.

5 EVALUATION

The following sections present our results and analysis.

5.1 Overall Performance Comparisons

Figure 6 shows performance results for the DRAM architectures across the applications studied, as average CPI. To understand the causes for the differences, e.g. whether from improved latency or improved bandwidth, we follow [11] and [4]: we run multiple simulations to distinguish between true execution time and memory overhead and distinguish between memory stalls that can be eliminated by simply increasing bandwidth and those that cannot.

The tops of the orange bars indicate the ideal CPI obtained with a perfect primary memory (zero latency, infinite bandwidth). The remaining portion above the orange bars is the overhead brought by primary memory, further broken down into stalls due to lack of bandwidth (red bar) and stalls due to latency (green bar). For example, the best CPI that could be obtained from a perfect memory for STREAM, as shown in Figure 6, is 4.3. With DDR3, the DRAM memory contributes another 4.6 cycles to the execution time, making the total CPI 8.9. Among these 4.6 cycles added by DDR3, only 0.3 cycles are stalls due to lack of memory bandwidth; the remaining 4.3 cycles are due to memory latency.

The first thing to note is that the CPI values are all quite high. Cores that should be able to retire 8 instructions per cycle are seeing on average one instruction retire every two cycles (bzip2), to 30 cycles (GUPS). The graphs are clear: more than half of the total overhead is memory.

As a group, the highest CPI values are single-rank (DDR3-1, DDR4-1) or single channel (LPDDR4-1) configurations . Single rank configurations exposes the tFAW protocol limitations [20, 21], because all requests must be satisfied by the same set of devices. Having only one rank to schedule into, the controller cannot move to another rank when the active one reaches the maximum activation window; thus the controller must idle the requisite time before continuing to the next request when this happens. The effect is seen when comparing DDR3-1 to DDR3, an average 21% improvement from simply using a dual-rank organization; or when comparing DDR4-1 to DDR4, an average 14% improvement from simply moving to dual-rank from single-rank.

LPDDR4-1 and LPDDR4 has the same bandwidth, but different configurations (64×1 vs 16×4 buses). There is a 22% improvement when using the quad-channel configuration, indicating that using more parallelism to hide longer data burst time works well in this case.

From there, the comparison is DDR3 to LPDDR4 to DDR4, and the improvement goes in that direction: LPDDR4 improves on DDR3 performance by an average of 8%, and DDR4 improves on LPDDR4 by an average of 6%. This comes from an increased number of internal banks (DDR4 has 16 per rank; LPDDR4 has 8 per rank but more channel/ranks, DDR3 has only 8 per rank), as well as increased bandwidth. The reason why LPDDR4, having more banks, does not outperform DDR4 is its slower DRAM timings, which was optimized for power but not performance.

Next in the graphs is GDDR5, which has almost twice the bandwidth of DDR4 and LPDDR4, but because it is a single-rank design (GDDR5 does not allow multi-drop bus configurations), it behaves like the other single-rank configurations: DDR3-1, DDR4-1, and LPDDR4-1, which is to say that it does not live up to its potential under our testing setup.

The best-performing DRAM architectures are HBM and HMC: the workloads are split roughly evenly on which DRAM is "best." One may be not impressed by the performance improvement here: though HBM and HMC have maximum bandwidths of 128GB/s and 120GB/s, roughly 8 and 13 times more than single-rank DDR3, they

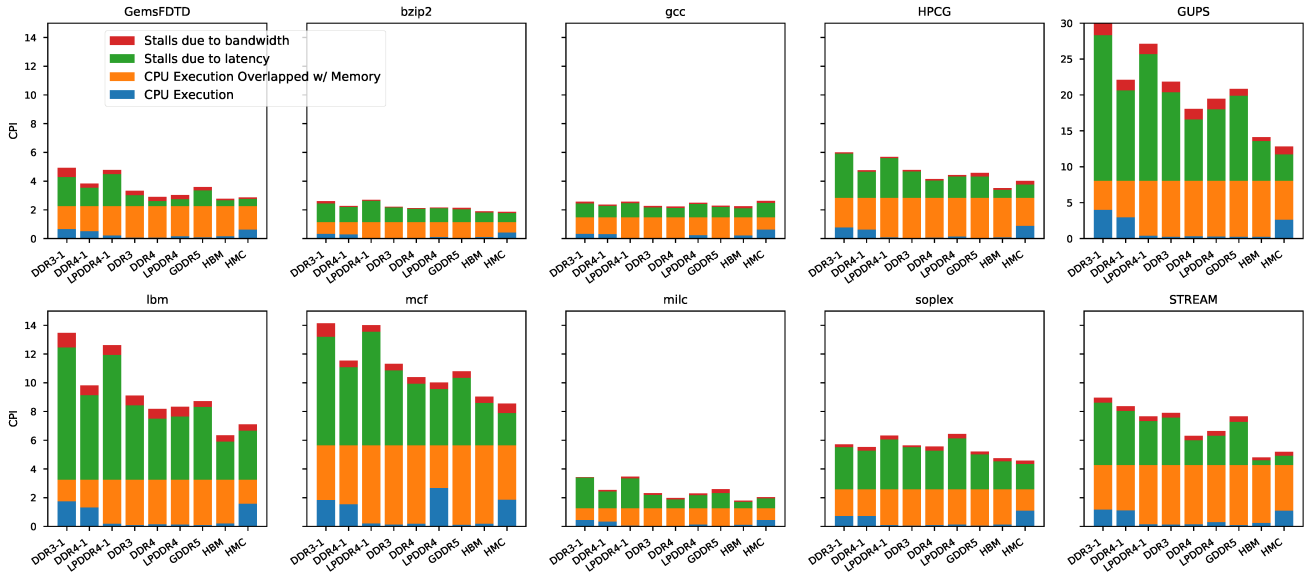


Figure 6: CPI breakdown for each benchmark. Note that we use a different y-axis scale for GUPS. Each stack from top to bottom are *stalls due to bandwidth, stalls due to latency, CPU execution overlapped with memory*, and *CPU execution*.

only achieve improvements of 2–3x over DDR3. Indeed, the performance improvement is less than the bandwidth increase; however, the total memory overhead decreases by a more significant amount, from being much more than half of the total CPI to accounting for less than 30% of the total CPI in many cases.

The net result is that the most advanced DRAMs, HBM and HMC, which combine all of the techniques previously shown to be important (multiple channels, and multiple ranks and/or banks per channel, and extremely high bandwidth), outperform all other DRAM architectures, often by a factor of two. The difference comes from virtually eliminating DRAM overhead, and the result is that half of the benchmarks go from being memory-bound to being compute-bound.

Lastly, it is clear from Figure 6 that the performance improvement brought by HBM and HMC is due to the significant reduction of latency stalls. In the following section, we break down the latency component to understand better.

5.2 Access Latency Analysis

Average memory latency is broken down in Figure 7, indicating the various operations that cause an operation not to proceed immediately.

Note that row access time varies with the different row buffer hit rates. In the worst case, where there is no row buffer hit, or the controller uses a close page policy (such as HMC), the row access time would reach its upper bound tRCD. Note also that the highly parallel, multi-channel DRAMs not only overlap DRAM-request latency with CPU execution but with other DRAM requests as well; therefore, these averages are tallied over individual requests.

At first glance, the reason HBM and HMC reduce the average access latency in Figure 6 is that they both tend to have shorter queuing delays than the other DRAMs. The reduced queuing delay comes from several sources, the most important of which is the degree of parallelism: HMC has 16 controllers internally; and HBM is configured with eight channels. This matches previous work [10], which shows that high bandwidth must be accompanied by high degrees of parallelism, and we also see it when comparing LPDDR as a DIMM (single-channel) with LPDDR in the quad-channel format: both have exactly the same bandwidth, but the increased parallelism reduces execution time significantly.

HBM and HMC also have additional parallelism from integrating many more banks than DDR3 and DDR4: they have 128 and 256 banks per package respectively, which is 8/16 times more than a DDR3 DIMM. Thus, potentially 8 times more requests can be served simultaneously, and the queuing delay for requests to each bank is reduced.

Further latency reduction in HMC is due to the controllers attached to it. In contrast with HMC, HBM can exploit open pages. In benchmarks such as *milc*, *HPCG*, *gcc*, *STREAM* and *lbm*, the row access time for HBM is reduced significantly, while HMC has the same constant row access time. Compared to DDR3, DDR4 and GDDR5, which also utilize open pages, HBM has more banks, meaning more potentially opened pages and thus higher row buffer hit rates, which further reduces the access latency. This happens in *STREAM* and *lbm*, where HBM has noticeably lower row access time than DDR3, DDR4 and GDDR5. This can be verified by looking at the row buffer hit rates shown in Figure 11.

Note that HMC exhibits a stable average access latency. From Figure 7 we see its average latency around 40ns, ranging from 39ns for *soplex* to 52ns for *STREAM*. The average is 41ns with a standard deviation of 3.9ns. HBM has the same average latency of 41ns, but a higher standard deviation of 8.6ns. This implies the behavior of HMC is more predictable in access latency, and it can be potentially useful in real-time systems, due to the deterministic nature of its

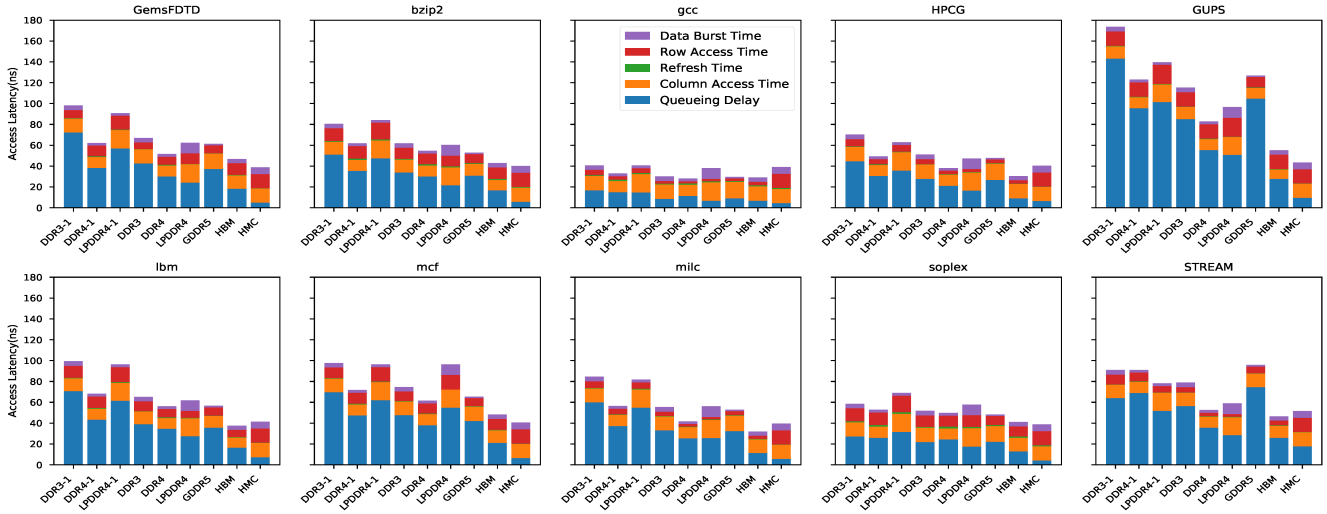


Figure 7: Average access latency breakdown. Each stack from top to bottom are *Data Burst Time, Row Access Time, Refresh Time, Column Access Time* and *Queuing Delay.*

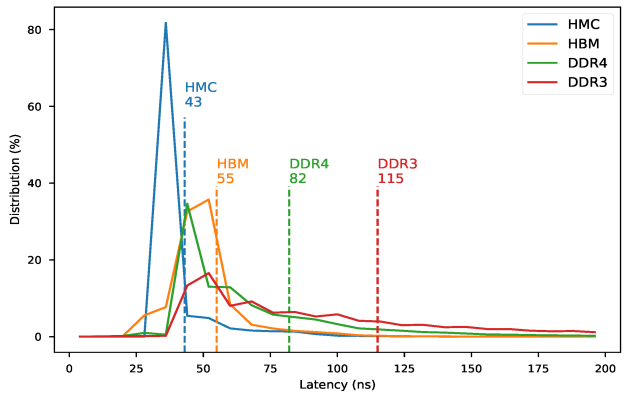


Figure 8: Access latency Distribution for GUPS. Dashed lines and annotation shows the average access latency.

close-page scheduling policy coupled with sufficient parallelism. Also note that the average latency of HMC is nearly pushed to its lower limit, as the row access and column access times contribute most of the latency. Queuing time can be improved by increasing parallelism, but improving row and column access time requires speeding up the internal DRAM arrays.

For further insight, we look at the access-latency distributions for several DRAMs, to give an idea of the quality of service and fairness of the DRAM protocols and DRAM-controller schedulers. Figure 8 shows probability densities for GUPS running on HMC, HBM, DDR4, and DDR3. The x-axis gives latency values; the y-axis gives the probability for each; and the area under each curve is the same. Thus, HMC, which spikes around 35ns, has an average that is near the spike, and the other DRAMs have average latencies that can be much longer.

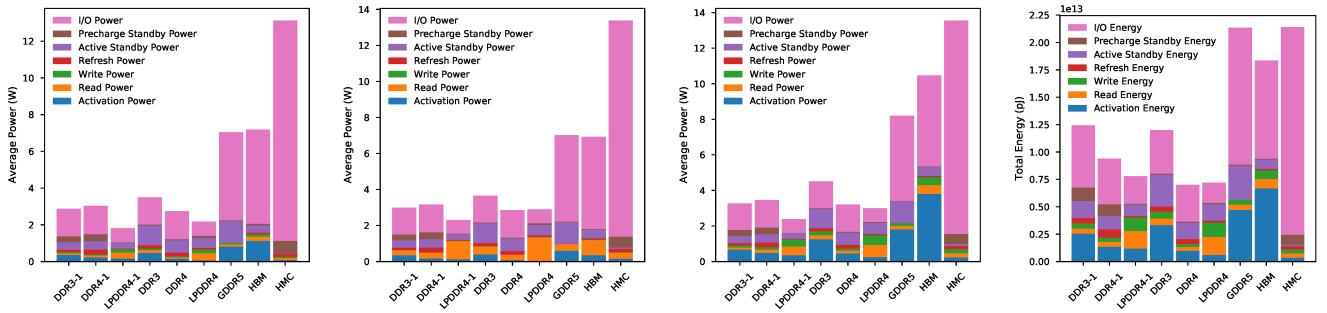
5.3 Power, Energy, and Cost-Performance

There are two major parts of a DRAM device's power: DRAM core power and I/O power. We use Micron's DRAM power model [40] to calculate the core power; we estimate I/O power based on various sources [2, 15, 43] and assume the I/O power for each DRAM is a constant while driving the bus.

Figure 9 shows the power estimation for a representative group of the benchmarks. I/O power tends to dominate all other fields. The high-speed interfaces of HBM and HMC are particularly power-hungry, driving the overall power dissipation of DRAM system upwards of 10W. HMC has the highest power dissipation, though its DRAM core only dissipates a small portion of its power. GDDR5 also has very high I/O power dissipation, considering its pin bandwidth is less than half that of HBM (48GB/s vs 128GB/s). DRAM-core power varies from application to application. HMC is still very steady, whereas others that adopt open-page policies see varying DRAM-core power. For instance, the core power of HBM can vary from 2 Watts to 5 Watts, with activation power the most significant variable.

Energy. Power is not the only story: energy-to-solution is another valuable metric, and we give the energy for GUPS in the far right graph. While the power numbers range from min to max over a factor of 7x, the energy numbers range only 3x from min to max, because the energy numbers represent not only power but also execution time, which we have already seen is 2–3x faster for the highest-performance and hottest DRAMs. Here we also see the effect of the single-rank vs. dual-rank systems: The single-rank configurations (DDR3-1, DDR4-1) have lower power numbers than the corresponding dual-rank configurations (DDR3, DDR4), because they have fewer DIMMs and thus fewer DRAMs dissipating power. However, the dual-rank configurations require lower energy-to-solution because they are significantly faster.

Power-Performance. Combining power with CPI values from earlier, we obtain a Pareto analysis of performance vs power, as



(a) bzip2 (b) HPCG (c) GUPS (d) GUPS(Energy)

Figure 9: Average power and energy. The left three figures show the average power breakdown of 3 benchmarks. The rightmost figure shows the energy breakdown of GUPS benchmark.

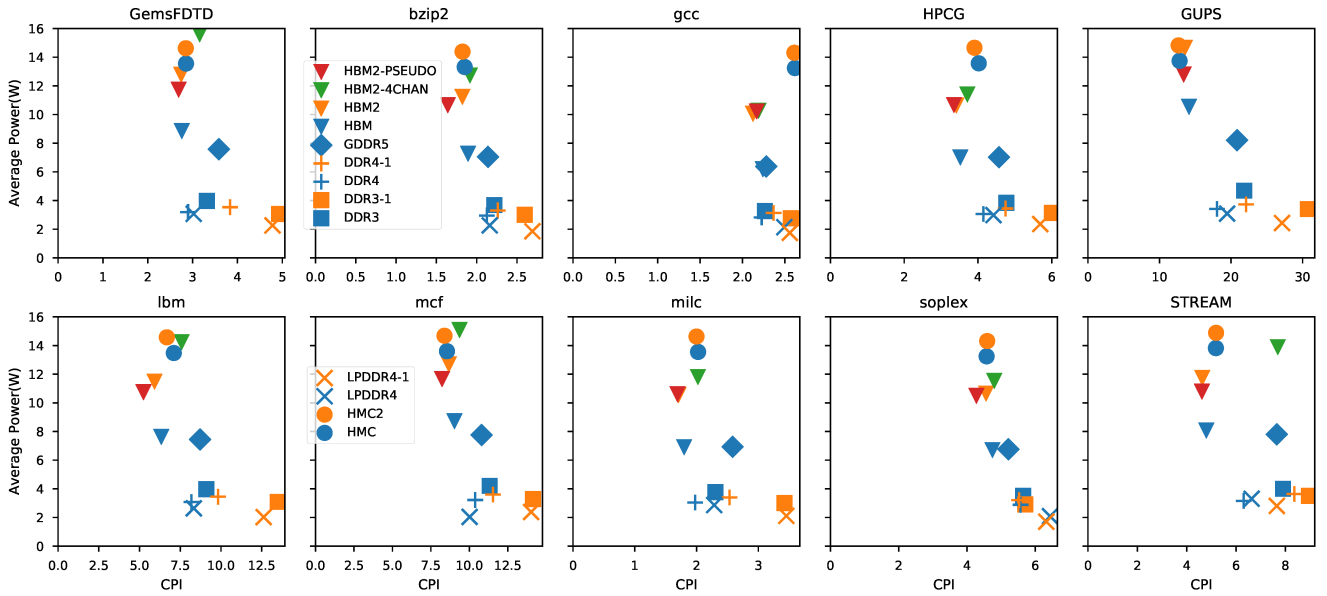


Figure 10: Average power vs CPI. Y-axis in each row has the same scale. Legends are split into 2 sets but apply to all sub-graphs.

shown in Figure 10. We add a few more simulation configurations that were not presented in previous sections: HBM2 running in pseudo channel mode (16 channels), labeled *HBM2-PSEUDO*; HBM2 configured as 4 channels, each 256 bits, labeled *HBM2-4CHAN*; and HMC2 which has 32 internal channels. Note also that, while HMC dissipates twice the power of HBM, HMC2 dissipates little more than HMC since they're configured using same links, whereas HBM2 dissipates noticeably more power than HBM.

In the graphs, the x-axis is CPI, and the y-axis is average power dissipation. By Pareto analysis, the optimal designs lie along a wavefront comprising all points closest to the origin (any design that is above, right of, or both above and to the right of another is "dominated" by that other point and is thus a worse design). The LPDDR4 designs are Pareto-optimal for nearly all applications, because no other design dissipates less power; and the HBM2-PSEUDO design is Pareto-optimal for nearly all applications, because it almost always has the lowest CPI value. HBM2-PSEUDO is a design from the HBM2 specification in which the 8-channel HBM2 is divided 2 pseudo channels each; as we have been discussing, this significant increase in parallelism is the type that one might expect to make HBM2 perform even better, and these results show that, indeed, it does.

Some interesting points to note: The improvements in design from 4-channel HBM2 to 8-channel to 16-channel almost always lie on a diagonal line, indicating that the 16-channel design dominates the others. HBM1 is almost always directly below HBM2, indicating that it has roughly the same performance but dissipates less power—this suggests that the 4-core processor is not fully exercising this DRAM, which we show to be the case in the following section. HMC1 and HMC2 have a similar vertical alignment, which suggests precisely the same thing. Comparing HBMs and HMCs,

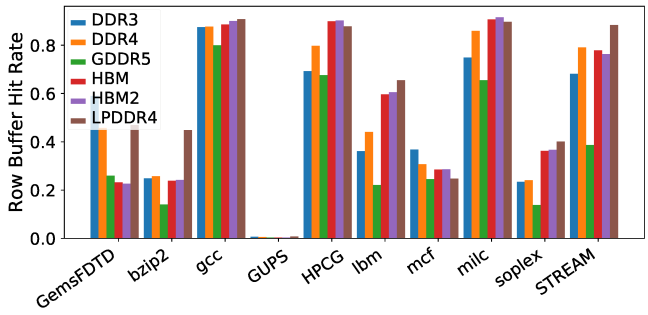


Figure 11: Row buffer hit rate. HMC is not shown here because it uses close page policy. GUPS has very few "lucky" row buffer hits.

they have similar CPI performance on these workloads, but HMCs tend to consume more power than HBMs in most cases. GDDR5 is almost always dominated by other designs (it is above and further to the right than many other DRAM designs), which indicates that its cost-performance is not as good as, for example, HBM1 (which exceeds it in performance) or DDR4 (which is more power-efficient). The relationship between DDR3 and DDR4 organizations is almost always a horizontal line, indicating that DDR4 improves performance significantly over DDR3, and a modest decrease in power dissipation.

While the power ranges from 2 Watts to 14 Watts for each benchmark, the changes in CPI are usually less significant (discussed in Section 5.1). Thus, to deliver the significant degree of performance improvement one would expect of high performance DRAMs like HBM and HMC, a disproportional amount of power is paid. Only in extremely memory-intensive cases, like GUPS and STREAM and manycore CPUs with high core counts (as we estimate in the next section), is the power-performance justified by using stacked memories like HBM and HMC. On the other hand, switching from DDR3 to DDR4 always seems to be beneficial in terms of both power and performance. LPDDR4 not surprisingly has the lowest power consumption and, in its quad-channel configuration, comparable performance to DDR4.

5.4 Row Buffer Hit Rate

The row buffer is the set of sense amps holding the data from an opened DRAM row. Accessing the data in a row buffer is much faster than accessing a closed row; therefore, most DRAM controller designers exploit this to improve performance. The only exception in the DRAMs studied in this work is HMC, which embraces a close-page design.

Figure 11 shows the row buffer hit rate for each application. GUPS has a near-zero row-buffer hit rate due to its completely random memory access patterns. Other than GUPS, the row buffer hit rates range from 13% to 90%, averaging 43% (39% if GUPS included). HBM and HBM2 have higher row hit rates than other DRAMs in most cases, because they have many more available rows. Note that high row buffer hit rate alone does not guarantee better performance. For example, DDR3 has highest row buffer hit rate in *GemsFDTD* benchmark, but the high row buffer hits only reduce the row access latency (which can be seen in Figure 7), but it also has much higher queuing delay as a result of having fewer banks.

5.5 High Bandwidth Stress Testing

Our studies so far are limited by the scale of the simulated 4-core system and only extract a fraction of the designed bandwidth out of the high-end memories like HBM and HMC. When we observe average inter-arrival times for the benchmarks (average time interval between each successive memory request), they are more than 10*ns* for most benchmarks—i.e., on average, only one memory request is sent to DRAM every 10*ns*. This does not come close to saturating these high-performance main memory systems.

Therefore, to explore more fully the potentials of the DRAM architectures, we run two contrived benchmarks designed to stress the memory systems as much as possible:

(1) We modify STREAM to use strides equal to the cache block size, which guarantees that every request is a cache miss.
(2) We use Cray's *tabletoy* benchmark. It generates random memory requests as fast as possible and stresses the memory system significantly more than GUPS, which is limited to pointer-chasing. If the average latency for the DRAM is 30ns, GUPS only issues an average of one request per 30ns. Tabletoy issues requests continuously.
(3) Lastly, we scale the cycle time of the processor to generate arbitrarily high memory-request rates.

In addition, we use higher-bandwidth HMC configurations. In the previous experiments, we used 2-link configurations for HMC and HMC2 at 120GB/s, because 4-link configurations would have been overkill. For the stress-test study, we upgrade links for both HMC and HMC2 for a maximum of 240GB/s and 320GB/s, respectively.

This should present two extremes of high-bandwidth request rates to the DRAM system: one sequential, one random, and as the request rates increase, these should give one a sense of the traffic that high-core-count manycore CPUs generate.

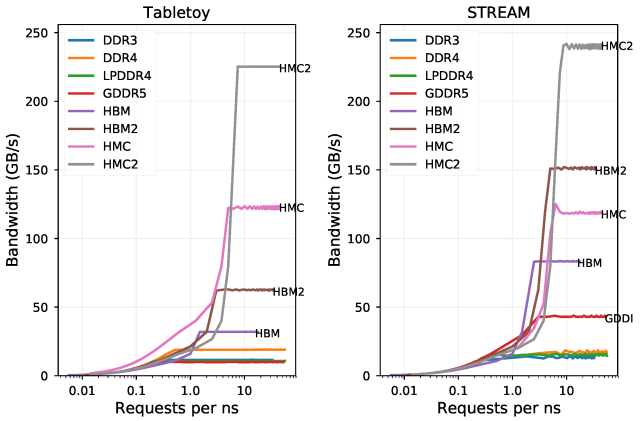


Figure 12: Tabletoy (random), left; STREAM (sequential), right. 64 bytes per request.

Figure 12 shows the results. The left-hand graph shows a random request stream; the right-hand graph shows a sequential stream. The x-axis is the frequency of requests being sent to the memory in log scale. Each request is for a 64-byte cache block. The request rate ranges from 100*ns* per request to more than 10 requests per *ns*. In the left-hand graph, one can see that all the traditional DDR

memories are saturated by 3*ns* per request. HBM and HBM2 reach their peak bandwidths at around 1 to 2 requests per *ns*, getting 32GB/s and 63GB/s respectively. HMC and HMC2 reach their peak bandwidth at the rate of 4 to 5 requests per *ns*, reaching 121GB/s and 225GB/s. The difference between HMC and HMC2 here is the number of vaults (channels) they have, 16 vs 32. The effective pin bandwidth of each vault is 10GB/s, meaning that both HMC and HMC2 reach about 75% of the peak internal bandwidths.

Looking at the sequential results in the right-hand graph, the high-speed DRAMs, e.g. GDDR5, HBM and HBM2, gain significantly more bandwidth than they do with the random stream. HMC and HMC2 only changes slightly, once again shows its steady performance regarding different types of worklaods.

The stress tests explain the bandwidth/latency relationship explained at the beginning of the paper in Figure 1. As the request rate is increased, the DRAMs go through the constant region into the linear region (where the curves start to increase noticeably; note that the x-axis is logarithmic, not linear). Where the stress test's bandwidth curves top out, the DRAM has reached its exponential region: it outputs its maximum bandwidth, no matter what the input request rate, and the higher the request rate, the longer that requests sit in the queue waiting to be serviced.

The stress-test results show that any processing node with numerous cores is going to do extremely well with the high-end, multi-channel, high-bandwidth DRAMs.

6 CONCLUSIONS

The commodity-DRAM space today has a wide range of options from low power, to low cost and large capacity, to high cost and high performance. For the single-channel (or single package) system sizes that we study, we see that modern DRAMs offer performance at whatever bandwidth one is willing to pay the power cost for, as the interface power dominates all other aspects of operation. Note that this would not be the case for extremely large systems: at large capacities, refresh power may also be very significant and dominate other activities. However, at the capacities we study, it is the transmission of bits that dominates power; thus, this provides an important first metric for system design: determine the bandwidth required, and get it.

Our studies show that bandwidth determines one's execution time, even for the modest 4-core CPU studied herein, as the higher bandwidths and, more importantly, the parallelism provided in the high-performance packages, assure that queuing delays are minimized. High-bandwidth designs such as HMC and HBM can reduce end-to-end application execution time by 2–3x over DDRx and LPDDR4 architectures. This translates to reducing the memory overhead from over half of the total execution time to less than 30% of total execution time. The net result: previously memory-bound problems are turned into compute-bound problems, bringing the focus back to architectural mechanisms in the processor that can improve CPI.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundationnder Award No. 1642424 and the Department of Defensender Contract No. FA8075-14-D-0002-0007, TAT 15-1158.

REFERENCES

[1] Dennis Abts, Natalie D. Enright Jerger, John Kim, Dan Gibson, and Mikko H. Lipasti. 2009. Achieving Predictable Performance Through Better Memory Controller Placement in Many-core CMPs. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 451–461. https://doi.org/10.1145/1555754.1555810

[2] AMD. 2015. High-Bandwidth Memory–Reinventing Memory Technology. https://www.amd.com/Documents/High-Bandwidth-Memory-HBM.pdf.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[4] Doug Burger, James R. Goodman, and Alain Kagi. 1996. Memory Bandwidth Limitations of Future Microprocessors. In *Proc. 23rd Annual International Symposium on Computer Architecture (ISCA'96)*. Philadelphia PA, 78–89.

[5] K. K. W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu. 2014. Improving DRAM performance by parallelizing refreshes with accesses. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 356–367. https://doi.org/10.1109/HPCA.2014.6835946

[6] N. Chatterjee, M. O'Connor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally. 2017. Architecting an Energy-Efficient DRAM System for GPUs. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*. 73–84. https://doi.org/10.1109/HPCA.2017.58

[7] N. Chatterjee, M. O'Connor, G. H. Loh, N. Jayasena, and R. Balasubramonia. 2014. Managing DRAM Latency Divergence in Irregular GPGPU Applications. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. 128–139. https://doi.org/10.1109/SC.2014.16

[8] Elliott Cooper-Balis and Bruce Jacob. 2010. Fine-Grained Activation for Power Reduction in DRAM. *IEEE Micro* 30, 3 (May 2010), 34–47. https://doi.org/10.1109/MM.2010.43

[9] Elliott Cooper-Balis, Paul Rosenfeld, and Bruce Jacob. 2012. Buffer On Board memory systems. In *Proc. 39th International Symposium on Computer Architecture (ISCA 2012)*. Portland OR, 392–403.

[10] Vinodh Cuppu and Bruce Jacob. 2001. Concurrency, Latency, or System Overhead: Which Has the Largest Impact on Uniprocessor DRAM-System Performance?. In *Proc. 28th Annual International Symposium on Computer Architecture (ISCA'01)*. Goteborg, Sweden, 62–71.

[11] Vinodh Cuppu, Bruce Jacob, Brian Davis, and Trevor Mudge. 1999. A performance comparison of contemporary DRAM architectures. In *Proc. International Symposium on Computer Architecture (ISCA)*. 222–233.

[12] Brian Dipert. 2000. The slammin, jammin, DRAM scramble. *EDN* 2000, 2 (Jan. 2000), 68–82.

[13] Jack Dongarra and Michael A Heroux. 2013. Toward a new metric for ranking high performance computing systems. *Sandia Report, SAND2013-4744* 312 (2013), 150.

[14] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob. 2007. Fully-Buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling. In *Proc. 13th International Symposium on High Performance Computer Architecture (HPCA 2007)*. Phoenix AZ, 109–120.

[15] Dean Gans. 2016. Low Power DRAM Evolution. In *JEDEC Mobile and IOT Forum*. http://server.semiconchina.org/downloadFile/1460449602275.pdf

[16] M. Ghosh and H. H. S. Lee. 2007. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 134–145. https://doi.org/10.1109/MICRO.2007.13

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

[18] HMC Consortium. 2013. *Hybrid Memory Cube Specification 1.0*. Hybrid Memory Cube Consortium.

[19] HMC Consortium. 2014. *Hybrid Memory Cube Specification 2.0*. Hybrid Memory Cube Consortium.

[20] Bruce Jacob, Spencer Ng, and David Wang. 2007. *Memory Systems: Cache, DRAM, and Disk*. Morgan Kaufmann.

[21] Bruce Jacob and David Tawei Wang. 2009. SYSTEM AND METHOD FOR PERFORMING MULTI-RANK COMMAND SCHEDULING IN DDR SDRAM MEMORY SYSTEMS. US Patent No. 7,543,102.

[22] Aamer Jaleel. 2010. Memory characterization of workloads using instrumentation-driven simulation. *Web Copy: http://www. glue. umd. edu/ajaleel/workload* (2010).

[23] JEDEC. 20012. *Low Power Double Data Rate 3 (LPDDR3), JESD209-3*. JEDEC Solid State Technology Association.

[24] JEDEC. 2007. *DDR3 SDRAM Standard, JESD79-3*. JEDEC Solid State Technology Association.

[25] JEDEC. 2012. *DDR4 SDRAM Standard, JESD79-4*. JEDEC Solid State Technology Association.

[26] JEDEC. 2013. *High Bandwidth Memory (HBM) DRAM, JESD235*. JEDEC Solid State Technology Association.

[27] JEDEC. 2015. *High Bandwidth Memory (HBM) DRAM, JESD235A.* JEDEC Solid State Technology Association.

[28] JEDEC. 2015. *Low Power Double Data Rate (LPDDR4), JESD209-4A.* JEDEC Solid State Technology Association.

[29] JEDEC. 2016. *Graphics Double Data Rate (GDDR5) SGRAM Standard, JESD212C.* JEDEC Solid State Technology Association.

[30] Dimitris Kaseridis, Jeffrey Stuecheli, and Lizy Kurian John. 2011. Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 24–35.

[31] Gwangsun Kim, John Kim, Jung Ho Ahn, and Jaeha Kim. 2013. Memory-centric system interconnect design with Hybrid Memory Cubes. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE Press, 145–156.

[32] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. In *Proc. International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 256–267. https://doi.org/10.1109/ISPASS.2013.6557176

[33] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting Phase Change Memory As a Scalable DRAM Alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 2–13. https://doi.org/10.1145/1555754.1555758

[34] J. Lee, T. Kim, and J. Huh. 2016. Reducing the Memory Bandwidth Overheads of Hardware Security Support for Multi-Core Processors. *IEEE Trans. Comput.* 65, 11 (Nov 2016), 3384–3397. https://doi.org/10.1109/TC.2016.2538218

[35] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee. 2015. A fully associative, tagless DRAM cache. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 211–222. https://doi.org/10.1145/2749469.2750383

[36] D. Li and T. M. Aamodt. 2016. Inter-Core Locality Aware Memory Scheduling. *IEEE Computer Architecture Letters* 15, 1 (Jan 2016), 25–28. https://doi.org/10.1109/LCA.2015.2435709

[37] Lei Liu, Zehan Cui, Yong Li, Yungang Bao, Mingyu Chen, and Chengyong Wu. 2014. BPM/BPM+: Software-based Dynamic Memory Partitioning Mechanisms for Mitigating DRAM Bank-/Channel-level Interferences in Multicore Systems. *ACM Trans. Archit. Code Optim.* 11, 1, Article 5 (Feb. 2014), 28 pages. https://doi.org/10.1145/2579672

[38] Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi. 2006. The HPC Challenge (HPCC) benchmark suite. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. 213.

[39] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh. 2015. Heterogeneous memory architectures: A Hardware/Software approach for mixing die-stacked and off-package memories. In *Proc. 21st International Symposium on High Performance Computer Architecture (HPCA)*. 126–136. https://doi.org/10.1109/HPCA.2015.7056027

[40] Micron. 2007. *TN-41-01 Technical Note—Calculating Memory System Power for DDR3.* Technical Report. Micron.

[41] G. Narancic, P. Judd, D. Wu, I. Atta, M. Elnacouzi, J. Zebchuk, J. Albericio, N. Enright Jerger, A. Moshovos, K. Kutulakos, and S. Gadelrab. 2014. Evaluating the memory system behavior of smartphone workloads. In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. 83–92. https://doi.org/10.1109/SAMOS.2014.6893198

[42] Mark Oskin and Gabriel H. Loh. 2015. A Software-Managed Approach to Die-Stacked DRAM. In *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT) (PACT '15)*. IEEE Computer Society, Washington, DC, USA, 188–200. https://doi.org/10.1109/PACT.2015.30

[43] J. Thomas Pawlowski. 2011. Hybrid Memory Cube (HMC). In *HotChips 23*. https://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.3-memory-FPGA/HC23.18.320-HybridCube-Pawlowski-Micron.pdf

[44] Steven Przybylski. 1996. *New DRAM Technologies: A Comprehensive Analysis of the New Architectures.* MicroDesign Resources, Sebastopol CA.

[45] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 24–33. https://doi.org/10.1145/1555754.1555760

[46] Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R. de Supinski, Sally A. McKee, Petar Radojković, and Eduard Ayguadé. 2015. Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?. In *Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS '15)*. ACM, Washington DC, DC, USA, 31–36. https://doi.org/10.1145/2818950.2818955

[47] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. 2000. Memory Access Scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*. ACM, New York, NY, USA, 128–138. https://doi.org/10.1145/339647.339668

[48] David Zaragoza Rodríguez, Darko Zivanović, Petar Radojković, and Eduard Ayguadé. 2016. *Memory Systems for High Performance Computing.* Barcelona Supercomputing Center.

[49] Paul Rosenfeld. 2014. *Performance Exploration of the Hybrid Memory Cube.* Ph.D. Dissertation. University of Maryland, Department of Electrical & Computer Engineering.

[50] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters* 10, 1 (2011), 16–19.

[51] Davesh Shingari, Akhil Arunkumar, and Carole-Jean Wu. 2015. Characterization and Throttling-Based Mitigation of Memory Interference for Heterogeneous Smartphones. In *Proceedings of the 2015 IEEE International Symposium on Workload Characterization (IISWC '15)*. IEEE Computer Society, Washington, DC, USA, 22–33. https://doi.org/10.1109/IISWC.2015.9

[52] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim. 2014. Transparent Hardware Management of Stacked DRAM as Part of Memory. In *Proc. 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 13–24. https://doi.org/10.1109/MICRO.2014.56

[53] Richard Sites. 1996. It's the Memory, Stupid! *Microprocessor Report* 10, 10 (Aug. 1996).

[54] Sadagopan Srinivasan. 2007. *Prefetching vs. the Memory System: Optimizations for Multi-core Server Platforms.* Ph.D. Dissertation. University of Maryland, Department of Electrical & Computer Engineering.

[55] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu. 2015. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *Proc. 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 62–75. https://doi.org/10.1145/2830772.2830803

[56] Aniruddha N. Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P. Jouppi. 2010. Rethinking DRAM Design and Organization for Energy-constrained Multi-cores. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. ACM, New York, NY, USA, 175–186. https://doi.org/10.1145/1815961.1815983

[57] S. Volos, J. Picorel, B. Falsafi, and B. Grot. 2014. BuMP: Bulk Memory Access Prediction and Streaming. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 545–557. https://doi.org/10.1109/MICRO.2014.44

[58] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. 2005. DRAMsim: A Memory System Simulator. *SIGARCH Computer Architecture News* 33, 4 (Nov. 2005), 100–107. https://doi.org/10.1145/1105734.1105748

[59] W. Wang, T. Dey, J. W. Davidson, and M. L. Soffa. 2014. DraMon: Predicting memory bandwidth usage of multi-threaded programs with high accuracy and low overhead. In *Proc. 20th International Symposium on High Performance Computer Architecture (HPCA)*. 380–391. https://doi.org/10.1109/HPCA.2014.6835948

[60] William A. Wulf and Sally A. McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News* 23, 1 (March 1995), 20–24.

[61] G. L. Yuan, A. Bakhoda, and T. M. Aamodt. 2009. Complexity effective memory access scheduling for many-core accelerator architectures. In *Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 34–44. https://doi.org/10.1145/1669112.1669119

[62] Dongli Zhang, Moussa Ehsan, Michael Ferdman, and Radu Sion. 2014. DIMMer: A Case for Turning off DIMMs in Clouds. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC '14)*. ACM, New York, NY, USA, Article 11, 8 pages. https://doi.org/10.1145/2670979.2670990

[63] T. Zhang, M. Poremba, C. Xu, G. Sun, and Y. Xie. 2014. CREAM: A Concurrent-Refresh-Aware DRAM Memory architecture. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 368–379. https://doi.org/10.1109/HPCA.2014.6835947