Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems

Mike O'Connor*†‡ Niladrish Chatterjee*† Donghyuk Lee† John Wilson†
Aditya Agrawal† Stephen W. Keckler†‡ William J. Dally†◇

†NVIDIA ‡The University of Texas at Austin ◇Stanford University

{moconnor, nchatterjee, donghyukl, jowilson, adityaa, skeckler, bdally} @nvidia.com

ABSTRACT

Future GPUs and other high-performance throughput processors will require multiple TB/s of bandwidth to DRAM. Satisfying this bandwidth demand within an acceptable energy budget is a challenge in these extreme bandwidth memory systems. We propose a new high-bandwidth DRAM architecture, Fine-Grained DRAM (FGDRAM), which improves bandwidth by 4× and improves the energy efficiency of DRAM by 2× relative to the highest-bandwidth, most energy-efficient contemporary DRAM, High Bandwidth Memory (HBM2). These benefits are in large measure achieved by partitioning the DRAM die into many independent units, called grains, each of which has a local, adjacent I/O. This approach unlocks the bandwidth of all the banks in the DRAM to be used simultaneously, eliminating shared buses interconnecting various banks. Furthermore, the on-DRAM data movement energy is significantly reduced due to the much shorter wiring distance between the cell array and the local I/O. This FGDRAM architecture readily lends itself to leveraging existing techniques to reducing the effective DRAM row size in an area efficient manner, reducing wasteful row activate energy in applications with low locality. In addition, when FGDRAM is paired with a memory controller optimized to exploit the additional concurrency provided by the independent grains, it improves GPU system performance by 19% over an iso-bandwidth and iso-capacity future HBM baseline. Thus, this energy-efficient, high-bandwidth FGDRAM architecture addresses the needs of future extreme-bandwidth memory systems.

CCS CONCEPTS

• **Hardware** → **Dynamic memory**; *Power and energy*; • **Computing methodologies** → Graphics processors; • **Computer systems organization** → Parallel architectures;

KEYWORDS

DRAM, Energy-Efficiency, High Bandwidth, GPU

ACM Reference format:
M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S.W. Keckler, and W.J. Dally. 2017. Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 14 pages.
https://doi.org/10.1145/3123939.3124545

1 INTRODUCTION

High bandwidth DRAM has been a key enabler of the continuous performance scaling of Graphics Processing Units (GPUs) and other throughput-oriented parallel processors. Successive generations of GPU-specific DRAMs, optimized primarily to maximize bandwidth rather than minimize cost per bit, have increased aggregate system bandwidth; first through high-frequency off-chip signaling with Graphics Double-Data Rate memories (GDDR3/5/5X [18, 21, 24]) and, most recently, through on-package integration of the processor die and wide, high-bandwidth interfaces to stacks of DRAM (*e.g.,* High Bandwidth Memory (HBM/HBM2) [20, 23] and Multi-Channel DRAM (MCDRAM) [15]). Future GPUs will demand multiple TB/s of DRAM bandwidth requiring further improvements in the bandwidth of GPU-specific DRAM devices.

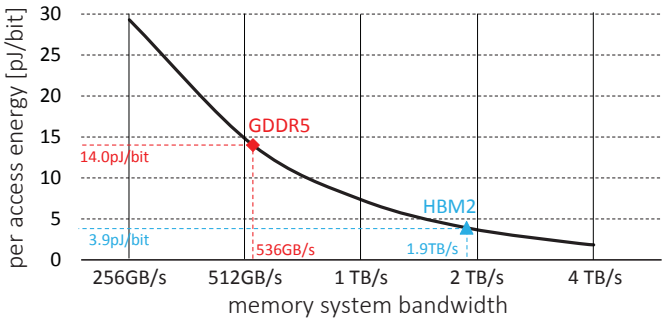
In this paper, we show that traditional techniques for extending the bandwidth of DRAMs will either add to the system energy, and/or add to the cost/area of DRAM devices. To meet the bandwidth objectives of the future, DRAM devices must be more energy-efficient than they are today without significantly sacrificing area-efficiency. To architect a DRAM device that meets these objectives, we carry out a detailed design space exploration of high-bandwidth DRAM microarchitectures. Using constraints imposed by practical DRAM layouts and insights from GPU memory access behaviors to inform the design process, we arrive at a DRAM and memory controller architecture, Fine-grained DRAM (FGDRAM), suited to future high-bandwidth GPUs.

The most formidable challenge to scaling the bandwidth of GPU DRAMs is the energy of DRAM accesses. Every system is designed to operate within a fixed maximum power envelope. The energy spent on DRAM access eats into the total power budget available for the rest of the system. Traditionally, high-end GPU cards have been limited to approximately 300W, of which no more than about 20% is budgeted to the DRAM when operating at peak bandwidth.

* Both authors contributed equally to the paper

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3124545



(a) Maximum DRAM access energy for given peak bandwidth within 60W DRAM power budget

(b) HBM2 energy consumption

Figure 1: GPU Memory Power and Energy

Figure 1a shows the DRAM energy per access that can be tolerated at a given peak DRAM bandwidth while remaining within a 60W DRAM power budget. We see that the energy improvements of on-die stacked High Bandwidth Memory (HBM2) over off-chip GDDR5 memories have allowed modern GPUs to approach a terabyte-per-second of memory bandwidth at comparable power to previous GPUs that provided less than half the bandwidth using GDDR5. This figure also demonstrates, however, that even with HBM2, systems with more than 2 TB/s of bandwidth won't be possible within this traditional power budget. A future exascale GPU with 4 TB/s of DRAM bandwidth would dissipate upwards of 120 W of DRAM power.

The energy to access a bit in HBM2 is approximately 3.97 pJ/bit, and, as shown in Figure 1b, it consists largely of *data movement energy* (the energy to move data from the row buffer to the I/O pins) and *activation energy* (the energy to precharge a bank and activate a row of cells into the row-buffer); the I/O energy accounts for the small remainder. The activation energy is a function of the row size and the row locality of the memory access stream, and it is a significant factor because most GPU workloads access only a small fraction of the 1KB row activated in HBM2. The data movement energy is determined primarily by the distance that the data moves on both the DRAM die and the base layer die to reach the I/O pins, the capacitance of these wires, and the rate of switching on this datapath. Since most current DRAM devices, including HBM2, send data from banks spread across the die to a common I/O interface, data may travel from the farthest corners of the device to the I/O PHYs on the base-layer, leading energy for data movement to dominate the overall energy. FGDRAM reduces both of these components of DRAM energy.

In FGDRAM, a DRAM die is a collection of small units called *grains*, with each grain having a local, dedicated, and narrow data interface. Much like a traditional DRAM channel, each grain serves a DRAM request in its entirety. However, there are two fundamental differences between an FGDRAM grain and an HBM2 channel. First, unlike a traditional HBM2 die where 16 DRAM banks share a single wide I/O interface, each FGDRAM grain fetches data from only a single DRAM bank. Second, each grain has a fraction of the bandwidth of a traditional HBM2 channel. These two architectural changes enable the main benefits of FGDRAM. First, eliminating the sharing of a DRAM channel by multiple banks eliminates the inter-bank global data bus on a DRAM die. This architecture reduces the distance moved by data from a row-buffer to the I/O hub, thereby reducing the on-DRAM data movement energy. Second, because each FGDRAM bank needs to provide less bandwidth than a traditional bank, FGDRAM is able to use techniques explained in Section 3.2 to achieve lower activation energy without significant area overheads. While these benefits combine synergistically to reduce the DRAM access energy, the allocation of private data channels to the individual banks on a die also exposes the entire bandwidth of the DRAM die to the GPU and paves the way for area-efficient bandwidth scaling. The throughput-optimized memory controllers on a GPU can easily exploit this architecture to provide high bandwidth to memory intensive applications.

In summary, this paper makes the following contributions:

- Based on a detailed analysis of GPU workloads (both compute and graphics) and practical DRAM architectures, we demonstrate that both data movement and row activation energies must be reduced to meet the energy target of future memories.
- We propose a new DRAM architecture, FGDRAM, which provides both 4× more bandwidth and 51% lower energy per access than HBM2, the highest bandwidth and most efficient contemporary DRAM.
- We develop an evolutionary approach to HBM2 which also provides 4× more bandwidth, but show FGDRAM is 49% lower energy than this iso-bandwidth baseline.
- The additional concurrency in our proposed FGDRAM architecture can be easily exploited by a GPU to improve the performance of a wide range of GPU compute workloads by 19% on average over the iso-bandwidth baseline.
- We also consider the iso-bandwidth baseline enhanced with two prior proposed techniques to improve DRAM performance and energy. We show that FGDRAM requires 34% less energy, is 1.5% less area, and is within 1.3% of the performance of this enhanced baseline.

2 BANDWIDTH SCALING CHALLENGES

This section examines the main challenges faced by conventional bandwidth scaling techniques when applied to high bandwidth DRAMs. We use the key insights gained from this analysis to guide the design of our proposed FGDRAM architecture.

2.1 DRAM Energy

As shown in Figure 1a, reducing DRAM energy is required to enable increased bandwidth without exceeding a reasonable energy budget for the DRAM in a system. The energy consumed by the DRAM is a function of the microarchitecture of the DRAM and the memory request pattern generated by the processor. Previous work [7] demonstrated that GPU workloads, both from the graphics and compute domains, incur high activation energy due

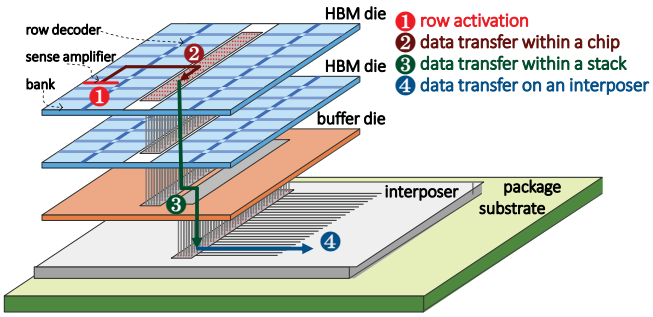


Figure 2: HBM2 access energy components. Reproduced from [6].

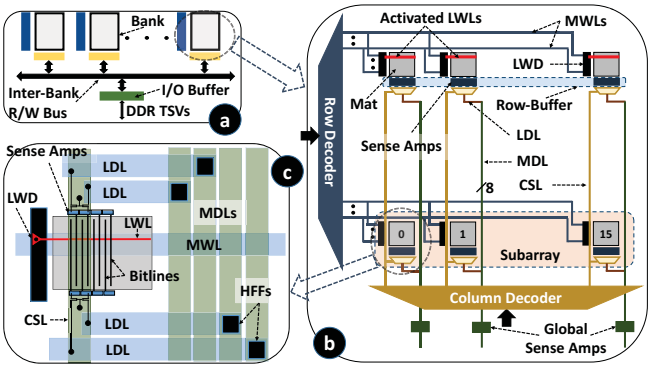


Figure 3: High Bandwidth Memory microarchitecture. Reproduced from [6].

to low row-access locality arising from irregular accesses and/or row-buffer interference between the thousands of concurrently executing threads. However, as shown in Figure 1b, the movement of data from the sense-amplifiers to the DRAM I/Os is another more significant contributor to overall energy. Figure 2 shows that in a stacked DRAM, the data first travels on the DRAM die from the bank's sense-amplifier to the central stripe, then down to the base layer using through-silicon vias (TSVs), and a short distance over the base-layer to reach the I/O bumps that connect the HBM2 stack to the GPU (approximately 9.9 mm). Using a detailed energy model based on physical floorplan and DRAM process characteristics (Section 4) and the actual data toggle rate of different applications, we found that switching the capacitance on this datapath requires 2.24 pJ/bit of energy on average. In contrast, transferring the data over the I/O wires on the silicon interposer requires an additional 0.3 pJ/bit, considering actual application data toggle rates. In total, with the average of 1.21 pJ/bit of activation energy, each HBM2 access incurs 3.92 pJ/bit of energy (including ECC overhead) which is a major impediment to increasing the aggregate bandwidth for a GPU.

Clearly, to reach a target 2 pJ/bit of overall energy consumption, future high-bandwidth DRAMs must reduce internal data movement energy. Opportunities to reduce activation energy should also be considered, as it is still a significant component of overall energy consumption.

2.2 Increasing Per-bank Bandwidth

Traditionally, bandwidth scaling of DRAM devices has been achieved by improvements in the I/O signaling frequency. However, while signal speed either on a PCB, an organic package substrate, or a silicon interposer can be increased with relative ease, scaling the frequency of the DRAM core and storage arrays at a commensurate pace is extremely difficult. Consequently, with internal DRAM frequencies remaining fairly similar over several product generations, DRAM vendors have turned to increasing the *prefetch width* to scale the internal DRAM bandwidth and match it with the bandwidth at the I/O. However, because current high-end GPU DRAMs are at the very limit of this scaling technique, continuing down this path will require either high energy or high area overhead.

In an HBM2 channel, the 64-bit I/O interface is operated at a 1GHz frequency, providing 16 GB/s of bandwidth on the DDR interface. The DRAM atom size (the size of a single DRAM request) is 32 Bytes, which is transferred with a burst length of four over the 64-bit DDR I/O interface. Throughout this paper, we consider an HBM2 stack to be operating in *pseudochannel* mode [41] with sixteen 64-bit wide channels per stack rather than legacy mode with eight 128-bit wide channels.

To support the 16 GB/s bandwidth per channel, an HBM2 bank operating at 500MHz outputs 128 bits in each internal clock; two banks operate in parallel to provide the required bandwidth (as described below in Section 2.3). Creating a future HBM stack with 4× the bandwidth at the same internal frequency with the traditional prefetch scaling method would require two changes. First, the internal prefetch from each bank must be increased to 512 bits and the DRAM atom size (the size of each DRAM request) must be correspondingly increased to 128 Bytes.

Increasing the prefetch width essentially requires a wider datapath from the DRAM bank to the I/O circuitry of the DRAM device. As shown in Figure 3, a DRAM bank is composed of several *subarrays*, with each subarray containing a subset of the rows in the bank and a set of sense-amplifiers that constitute the row-buffer of the subarray. As a typical example, the 16K rows in an HBM2 bank are organized as 32 subarrays, each containing 512 rows of DRAM cells. Each subarray is not monolithic, but consists of several 512×512 arrays of DRAM cells which are typically referred to as *mats*. When a row is activated in a subarray, each mat activates a section of the row into its local set of sense-amplifiers. This subarray performs this operations by first driving a Master Wordline (MWL) in a high level metal across the subarray, which in turn drives a Local Wordline (LWL) in every constituent mat. Subsequently, when a DRAM atom is read, *every* mat of the subarray outputs a few bits of the DRAM atom. In HBM2, a 1 KB wide row is split across 16 mats, each 512 bits wide; on a read command, each mat provides 16 bits over two internal cycles to return the 32 Byte DRAM atom. The read command drives a set of Column Select Lines (CSLs) in each mat which work as mux select signals, driving data from the target sense-amplifiers to the Master Datalines (MDLs) via the Local Data Lines (LDLs) [13, 25, 44]. Quadrupling the prefetch width of the bank requires either increasing the number of mats in a subarray to 64 or quadrupling the bandwidth per mat so that each mat outputs 64 bits over two cycles instead of 16 bits. The first option increases the row-size and consequently the activation energy. Given the

need to *reduce* energy in high bandwidth memories, this increase is a step in the wrong direction.

On the other hand, increasing the mat bandwidth requires large area overheads. The 3 metal layers in a typical DRAM process require a metal layer for the vertical bitlines in the mat, one for the MWLs and LDLs in the horizontal direction, and a third for the CSLs and MDLs in the vertical direction. The MWL and LDL metal layer has 4× the pitch of the LWL which is built using silicided polysilicon [16, 25, 36, 44]. Likewise, the CSLs and MDLs have 4× the pitch of a bitline. A DRAM mat's area is dictated by the number of wiring tracks required in these coarse-pitch metal layers. Quadrupling the mat bandwidth requires increasing the number of both the LDLs and MDLs from 16 to 64 (each signal being differential). This approach leads to a 77% increase in mat area due to increases in the wiring tracks in both the vertical and horizontal directions. While some of that area can be saved by trading off CSL count for increased MDL count, additional area is required to increase the width of the global inter-bank I/O bus that connects the banks to the I/O circuitry. Increasing the prefetch width thus significantly increases the cost of the DRAM device.

Furthermore, increasing the DRAM atom size is undesirable for multiple reasons. First, previous work has shown that GPU compute applications benefit from 32-byte sector sizes, and memory hierarchies designed to support such request sizes boost both performance and energy by avoiding data overfetch [35]. Second, graphics pipelines compress render surface tiles into 32-byte units to save DRAM bandwidth and amplify L2 capacity [31]. Increasing the atom size from 32 Bytes to 128 Bytes defeats the benefit of this important optimization and leads to a 17% degradation in performance for the graphics workloads we simulated. DRAM bandwidth scaling techniques must avoid increasing the DRAM mat bandwidth or the DRAM atom size.

2.3 Overlapping Accesses in Different Banks

Due to increasing I/O frequencies and stagnating DRAM internal frequencies in modern high-bandwidth DRAM devices, the time to transfer a single DRAM atom on the I/O bus (*t*BURST) is smaller than the minimum time between successive read requests to one DRAM bank. Thus, successive read accesses to the same DRAM bank cannot saturate the DRAM data interface.

To address this issue, recent DRAM standards, such as DDR4 [19], GDDR5, and HBM/HBM2 support *bank grouping*. The banks in a given DRAM channel are partitioned into several bank groups, typically with 4 banks per group. Accesses to different bank groups can be issued closely together, regardless of the cycle time of a single bank. This short delay between successive column commands (i.e. reads and writes) to different bank groups is the $tCCD_S$ timing parameter. This $tCCD_S$ parameter is equal to the *t*BURST time, ensuring "gapless" transmission on the data bus across successive accesses. The cycle time of a given bank (and possibly some structures shared with other banks in the same group) determines the rate at which successive column commands to the same group can be issued. This longer delay is the $tCCD_L$ timing parameter. To make efficient use of the full DRAM bandwidth, requests must alternate among different bank groups.

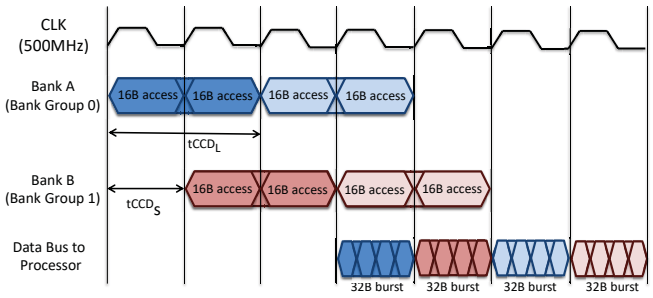


Figure 4: Overlapping multi-cycle accesses among bank groups. Each command requires two clock cycles to access the data for a 32B burst. Commands to the different bank groups can be separated by $tCCD_S$ =2 ns, while commands to the same bank group must be separated by $tCCD_L$ =4 ns.

Using this bank grouping approach, a DRAM can support higher bandwidth from a channel than a single bank can provide. In particular, if each bank accumulates the required data for a request over multiple internal cycles, the DRAM can support channel bandwidths many times that of a single bank's bandwidth. Figure 4 illustrates how a single 256-bit access is split over two 128-bit accesses to a single bank. Since a request to a different bank-group can overlap with the second phase access of the first bank, the total bandwidth available on the interface is twice that of a single bank. In fact, HBM2 employs this technique [23], allowing the bandwidth to double relative to first-generation HBM without requiring additional per-mat bandwidth. This approach can be further extended with more bank groups and more internal access cycles to enable a larger ratio between the bank and channel bandwidth.

While this approach enables higher channel bandwidths, it requires very fast switching on the internal DRAM global data bus that interconnects all the banks, particularly for high channel bandwidths. Furthermore, the high ratios required for a 4× bandwidth HBM2 derivative would require rotating read and write commands among at least 8 bank groups in each channel. Unfortunately, back-to-back accesses to the same bank would be very slow in this case, as each bank would require multiple cycles for a single access. In this example, $tCCD_L$ is 16ns, instead of 4 ns as it is today. We found that performance degrades by an average of 10.6% compared to an iso-bandwidth system with conventional inter- and intra-bankgroup timings.

2.4 Additional Parallel Channels

The complexities of increasing the bandwidth of a single channel can be avoided by simply increasing the number of channels in the device. Each channel remains the same baseline bandwidth, though possibly using a narrower, higher-speed I/O channel. All of the channel timing parameters remain the same.

Unfortunately, a straightforward replication of channels is area intensive. Even if the total storage capacity remains the same, the number of independent DRAM banks increases in proportion to the number of additional channels – each bank simply has proportionally fewer rows. However, increasing the channel count requires the replication of the row and column address decoders and the global sense-amplifiers, leading to 36% higher area.

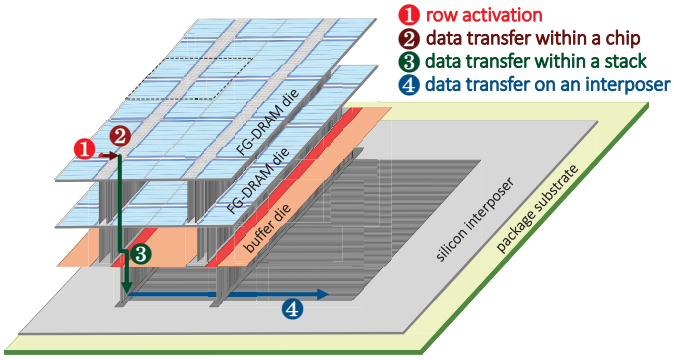


Figure 5: FGDRAM Die Stack Architecture

Alternatively, the number of banks per channel can be proportionally reduced to avoid most of this additional area penalty. For instance, a current 16-channel HBM2 stack with 16 banks per channel evolves into a 4× bandwidth 64-channel stack with 4 banks per channel. Each channel has the same bandwidth as a current HBM2 channel, using one quarter the number of 8 Gb/s I/O signals. The total number of banks remains constant. This evolutionary approach to scaling bandwidth has the fewest downsides, and we use this quad-bandwidth HBM stack (QB-HBM) as a baseline for comparison to our FGDRAM proposal.

By reducing the number of banks available in each channel, it becomes more difficult for the memory controller to always find work for all four banks so that it can hide activate/precharge latencies under data transfers from the other banks. Also, while this QB-HBM architecture addresses the target bandwidth demand for future systems, there is no significant reduction in the energy per access. As a result, we also consider an enhanced alternative to the QB-HBM baseline which incorporates prior published approaches to increase bank-level parallelism and reduce DRAM activation energy.

A potential solution to the reduction in exploitable bank-level parallelism can be found in a technique called Subarray Level Parallelism (SALP) [26]. This approach enables subarrays within a DRAM bank to be independently activated, effectively creating additional opportunities to perform activates/precharges while accesses to other subarrays take place. In effect, it creates a number of smaller semi-independent banks within a single bank structure. When SALP is enabled, bank-level parallelism is recovered, and the performance of the baseline 16-bank configuration is restored.

As shown in Figure 1b, the energy due to DRAM row activations is a significant portion of DRAM energy. The subchannels architecture [6] partitions each bank and the associated DRAM channel into narrow partitions, reducing the effective DRAM row size and DRAM activation energy. We apply both SALP and subchannels to create an enhanced baseline quad-bandwidth HBM design. We will compare these baseline alternatives to our proposed architecture in energy, area, and performance.

3 FINE-GRAINED DRAM

Based on the challenges faced in scaling bandwidth and reducing DRAM energy, three key objectives shape our DRAM architecture proposal:

(1) Additional bandwidth must be exposed via additional parallel channels.
(2) Data movement energy must reduced by limiting the distance between banks and I/Os.
(3) Activation energy must be reduced by limiting the effective row-size of each activate.

Our goal is to architect a DRAM stack with 4× the bandwidth of a current HBM2 stack while simultaneously reducing the energy per access by a factor of two. This 1 TB/s, 2 pJ/bit DRAM will enable future multi-TB/s GPU memory systems. Our proposed *Fine-Grained DRAM* (FGDRAM) stack architecture (Figure 5) partitions the DRAM into a number of small, independent *grains*. Each grain is essentially a narrow slice of a DRAM bank along with an adjacent associated and dedicated I/O interface. This architectural approach achieves the energy and bandwidth objectives by simultaneously addressing data movement energy and providing direct parallel access to every DRAM bank. Furthermore we apply an area-efficient technique to reduce the effective row-size, addressing activation energy. The finely partitioned FGDRAM architecture requires changes to the interface organization, the bank (grain) architecture, and the memory controller architecture. In contrast to Figure 2, the data movement energy (1) row activation energy and (2) data transfer within the chip are significantly reduced.

3.1 Interface Architecture

Parallel narrow channels. The proposed 1 TB/s FGDRAM stack architecture provides equivalent bandwidth to the proposed quad-bandwidth HBM (QB-HBM) baseline design. The QB-HBM design has 64 channels, each providing 16 GB/s of bandwidth. The FGDRAM architecture provides 512 grains in each stack, each providing 2 GB/s of bandwidth. The access granularity of each read or write request is still a 32 byte atom in both architectures. In FGDRAM, one request must be serialized over the narrower bus resulting in a *t*BURST of 16 ns. While this increases the minimum latency of each read request by several nanoseconds, this modest increase has a negligible impact on performance in highly threaded, bandwidth-oriented systems.

Using many narrow relatively low-bandwidth channels allows the FGDRAM architecture to provide direct connections to each bank in the stack. Figure 6 shows one QB-HBM channel with its 4 physical banks, and the equivalent 16 grains in the FGDRAM proposal. One key aspect of the FGDRAM architecture is unlocking bandwidth by allowing all banks to be accessed in parallel and eliminating the bottleneck of a shared bus interconnecting several banks. This approach reduces energy via the direct connection to nearby I/O, as well as provides the necessary bandwidth without an area penalty. Partitioning the interface into a large number of low bandwidth channels provides a number of other opportunities to optimize energy and simplify the memory controller, as discussed in Section 3.3.

Address/command interface. One command channel provides the commands to eight grains; there are 64 command channels for a 512-grain FGDRAM stack. As shown in Figure 6, the shared command logic sits between two physical banks to control the eight grains. Each command specifies the grain it targets. The command protocol is similar to a DDR/HBM2 interface with explicit activates, reads,

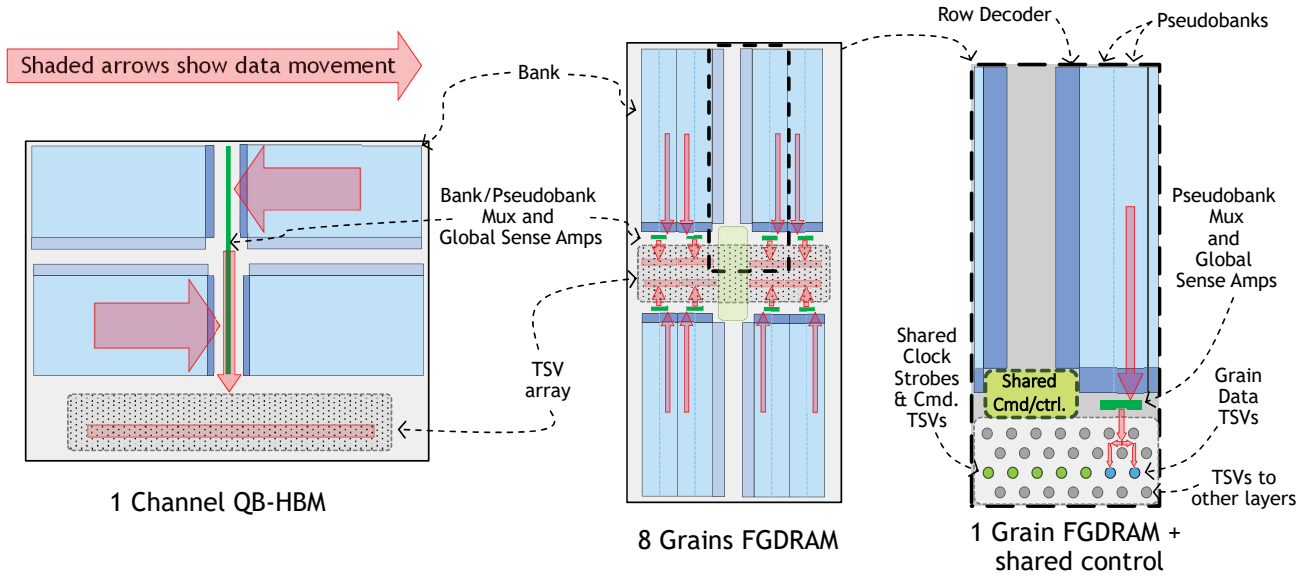


Figure 6: QB-HBM Channel and FGDRAM Grain architecture. Arrows illustrate data movement within the two architectures.

and writes sent to the DRAM. Sharing a single command interface among eight grains does not degrade performance. The long *t*BURST required from a grain allows commands for each of the other grains to be sent before a given grain needs another command. The overall ratio of command to data bandwidth remains the same as HBM2 and the QB-HBM baseline.

I/O signaling. As in the QB-HBM baseline, we are assuming a straightforward evolution of the existing HBM2 PHY technology to an 8 Gb/s Pseudo-Open Drain Logic (PODL) PHY operating at 1.2V similar to those in GDDR5 [18]. This provides 4× the bandwidth of the existing 2 Gb/s HBM2 PHY with a similar signal count. As a result, each grain transfers data over just two data signals.

3.2 Grain Architecture

One key objective guiding the architecture of a grain is reduced activation energy. Each grain in the FGDRAM architecture is the equivalent of a bank in HBM2 except that it has a private, serial I/O connection to the GPU. This section demonstrates how the existing HBM2 bank architecture can be modified to create *pseudobanks* in a grain, reducing the effective row-activation granularity.

Reducing row size. To reduce row size and the corresponding activation energy overhead, we leverage the bank architecture of the "subchannels" architecture described in [6]. This scheme partitions a bank into a number of semi-independent "subchannel" slices, each with a fraction of the original bank bandwidth. Importantly, each of these subchannels also can semi-independently activate a row that is a fraction of the baseline row size of the entire bank. We only use the bank architecture proposed in the subchannels paper, as the other aspects of the subchannels are pertinent just in the context of a conventional HBM2 architecture. The interface partitioning and command bandwidth issues are not applicable to FGDRAM.

The subchannels architecture exploits the existing hierarchical composition of a DRAM row assert signal. In the HBM2 baseline

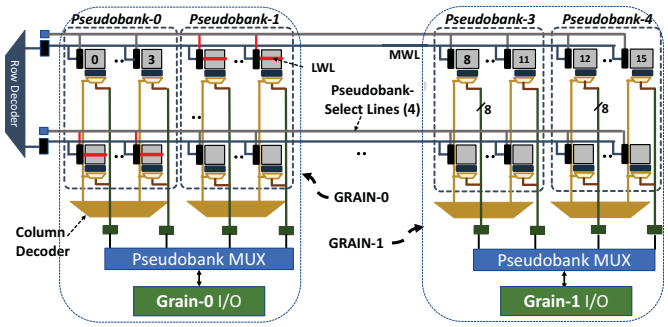


Figure 7: FGDRAM Grain. A bank is split into 2 grains, each with 2 pseudobanks.

(Figure 3(b)), on an activate, a MWL is driven across the entire subarray. The MWL, in turn, drives a local wordline (LWL) in each of the mats in the subarray by turning on each mat's local wordline driver (LWD). In the FGDRAM architecture, the assertion of the MWL leads to assertion of the LWLs in only a subset of the mats. As a result, the effective row size of an activation is reduced. Since only a subset of mats contain the active row, the bandwidth for a subchannel is correspondingly limited to a fraction of the original bank bandwidth.

FGDRAM pseudobanks. Like the QB-HBM baseline, the FGDRAM architecture delivers the required bandwidth by allowing many banks to be simultaneously active. There are 256 banks in a 64-channel QB-HBM stack, and each bank is capable of delivering 128 bits every 2 ns (8 GB/sec). Like current HBM2, the QB-HBM architecture keeps two banks active in each channel (via the bank-grouping technique described in Section 2.3). In contrast, the FGDRAM architecture allows all banks to be active simultaneously, transferring different

DRAM atoms, since each bank has access to its own dedicated I/O channel.

To match the 2 GB/s bandwidth of a grain, it is sufficient to involve only 4 mats in each grain, with each mat providing 8 bits per internal 500 MHz bank cycle. This is well matched to partitioning each bank using the subchannels bank architecture into 4 slices. Data is brought out of one of these slices in each grain over the 32-bit wide connection to the GSAs of the grain and pushed to the serialization I/O buffer over 8 internal cycles. The datapath is pipelined such that the external burst to the GPU can begin as soon as the first 32-bit section of the DRAM atom is received at the I/O buffer. For a single DRAM atom (32 Byte) read in the FGDRAM architecture, multiple column-select lines (CSLs) must be asserted sequentially, to select consecutive columns that store the entire DRAM atom with a slice. Instead of sending multiple column commands from the memory controller, the column address sent with the read command is incremented using a small 3-bit ripple carry adder inside the DRAM in successive cycles of a burst.

We call each such division of a grain a *pseudobank* as shown in Figure 7. A pseudobank is somewhat different from a subchannel because all of the subchannels in a bank are potentially active at once, and interleaving of requests among traditional banks is required for good performance.

Instead, the FGDRAM architecture divides a bank into 4 pseudobanks, but there are only 2 grains per bank. It is here that we take advantage of the fact that the total bank bandwidth in a baseline QB-HBM stack is twice what we need to deliver in our 1 TB/s FGDRAM stack. Each grain therefore consists of two pseudobanks. "Bank-level parallelism" (BLP) within a grain is achieved by allowing activates and precharges in one pseudobank, while a read or write operation is accessing the other. Thus, the FGDRAM architecture leverages the same technique to both reduce the effective row size and provide (pseudo)bank-level parallelism within a single bank.

Reducing data movement energy. As shown in Section 2.1, a large fraction of the HBM2 access energy in many applications is expended in moving data from the global sense amplifiers associated with a DRAM bank to the I/O pads. This energy is proportional to the distance the data must travel. Therefore, the FGDRAM grain architecture reduces this distance by providing each DRAM bank with adjacent local I/O. Figure 6 illustrates the reduction in data movement distance with FGDRAM. Data moves directly to an adjacent set of global sense amplifiers, and the small, narrow multiplexer that selects which of two adjacent pseudobanks is providing data to be sent through the TSVs to the I/O PHYs. This is in contrast to the traditional QB-HBM approach, where the data from all four banks must be multiplexed onto a central shared bus before being sent to the appropriate TSVs down to the base-layer PHYs. Also, rather than a central TSV area in the middle of the die as the QB-HBM requires, the TSV array is partitioned into two strips in the FGDRAM architecture. The QB-HBM design can't easily partition the TSV array in a similar manner because all the data from 4 banks must be muxed to a central shared bus, and splitting the group of 4 banks with a TSV array would force half of the per-bank data buses to traverse the TSV array to the mux, and then have the shared data bus deliver the selected data back to the TSVs. In FGDRAM, the data buses from a bank are directly routed to the immediately adjacent TSV area to be routed to the base layer. Adjacent to the TSV area on the base layer are the PHYs which connect to the host processor within the package.

Each grain is essentially a pair of DRAM pseudobanks along with an associated private serial I/O interface, and the grains can independently process different DRAM operations in parallel. By subdividing the DRAM die and constraining data movement (and to a certain extent command signals) to these small subdivisions, on-die data movement energy is significantly reduced.

3.3 Memory Controller Architecture

The FGDRAM architecture requires the memory controller to manage and schedule requests for a large number of independent grains. Rather than a single HBM2 command interface supporting two 16 GB/s channels, FGDRAM distributes this bandwidth among 16 grains. Because a group of 8 grains shares a command channel, there are twice as many command interfaces onto which the memory controller must schedule requests. The request rate on each command channel is one half that of the baseline HBM2 command interface. Thus, the memory controller must perform essentially the same amount of work per unit of bandwidth. Splitting the command interfaces allows a more energy efficient implementation in the DRAM. The commands a delivered near where they are needed and commands can be handled at the core 500 MHz clock rate. This architecture saves a small amount of data movement energy and clock power in the DRAM.

Command interface. The shared command interface is used to send commands to the eight associated grains. Just as in HBM2, separate row and column commands can be sent simultaneously. Each command specifies the 3-bit grain identifier with each command. Each command requires at least 2 ns to be transmitted across the interface. This latency allows each command to be processed at the low-speed 500 MHz internal DRAM core clock, thereby saving clock power in the DRAM command processing logic. The commands mirror those found in conventional HBM2. The commonplace commands like refresh, activate, precharge, read, and write (with optional auto-precharge) all specify which of the eight associated grains are targeted by the command. Some commands apply to all eight grains associated with a command channel; these commands are primarily associated with configuration or transitioning between low-power sleep/self-refresh modes.

Just as in a conventional DRAM, a number of timing parameters govern when certain commands can be sent relative to earlier commands. In FGDRAM, some of these timing parameters differ from the HBM2 baseline. For instance, because FGDRAM has a smaller DRAM row size and lower peak command issue rate, the *t*FAW (Four Activate Window) parameter due to power delivery constraints is effectively eliminated. Each command interface issuing an activate of a 256B row every 2 ns roughly matches the aggregate activation rate in number of bytes that an HBM2 die can sustain under the constraints of its *t*FAW parameter. The long 16 ns *t*BURST associated with each grain also simplifies the scheduling in the memory controller. With two pseudobanks per grain, and just two requests per activate, FGDRAM can hide all activation/precharge latencies and keep its interfaces 100% busy. While the memory controller must manage tracking requests to a number of grains and pseudobanks,

these are relatively simple directly indexed structures. Deep associative queues required to find multiple row-buffer hits are are much less important in the FGDRAM architecture, saving memory controller complexity.

One complexity FGDRAM introduces to the memory controller is the need to ensure that two different rows in different pseudobanks *within in the same subarray* are never simultaneously activated. The logic to track this is straightforward, but is new relative to the baseline HBM2 controller. Careful memory address layout and address swizzling makes this situation relatively unlikely to occur, but it must be prevented by delaying the activate or precharging the other pseudobank.

Command bandwidth. While the FGDRAM architecture has more independent command interfaces, the total aggregate bandwidth, both in command rate and raw I/O bandwidth of these interfaces, is the same as HBM2 relative to the total data bandwidth. In other words, a 1 TB/s FGDRAM stack has 4× more command I/O bandwidth and the memory controller must support 4× the command issue rate of an HBM2 stack. A new read or write command can only be sent to a grain once every 16 ns due to the length of the data burst on the per-grain data lines. Because eight grains share a command channel, a total of 8 read or write commands can be sent on this command channel every 16 ns. The worst-case scenario for command bandwidth is the case of one read or write command per activate. The row-command interface requires more than 2 ns to send an activate command due to the long row address. The 8:1 ratio of grains to command creates a bottleneck in this situation, but it is similar to the difficulty of keeping the bus utilized in HBM2 with only one access per activate. The FGDRAM architecture is balanced to sustain 100% utilization with two 32B accesses per activated 256B row, as long as each grain has requests distributed among both pseudobanks.

3.4 ECC

The current HBM2 architecture supports ECC bits stored within the DRAM array and sent across the interface. Each 32B DRAM burst is accompanied by 32-bits of additional ECC data sent on 16 additional data pins per channel. The QB-HBM architecture assumes a similar arrangement, sending the 32 additional bits per burst on 4 additional data signals per channel. This enables single-error correct, double-error detect (SECDED) on each of the two 64-bit word of a burst. ECC generation and checking is handled by the host processor.

With FGDRAM, storing the ECC data in the array is straightforward. As pointed out in [6], in structures like the narrow pseudobanks, additional ECC data is stored in slightly wider mats with 576 columns. Sending additional data in FGDRAM on extra pins is difficult since adding a single pin per grain would be a 50% overhead. As a result, two options are possible. First, the ECC data can be sent over additional cycles on the existing data bus. This would require the data I/Os to operate at 9 Gb/s rather than 8 Gb/s to compensate for having fewer data pins than an ECC-enabled HBM2-style interface. The energy overhead these approaches are roughly equivalent (though there is somewhat lower I/O energy if the data+ECC burst is sent on fewer pins at a slightly faster rate). Alternatively, if operating the data I/Os significantly faster is not a possibility, moving to in-DRAM ECC generation/checking like that employed in LPDDR4 [5] would provide protection within the DRAM array without the necessity to transfer the data across the interface. The long burst lengths in FGDRAM are well suited to this approach by providing a minimum access granularity to prevent the need for read-modify-writes to update the ECC data. A short 4-bit CRC code is then applied to each data transfer to detect errors on the I/O interface for retry. This 4-bit CRC can detect all 3-bit or fewer errors, and requires the I/O to operate only slightly faster at 8.125 Gb/s to transmit this incremental CRC data. The energy overheads of this approach are expected to be minor, with the I/O energy savings compensating for the ECC generation and checking being performed in a less power-efficient DRAM process than the host processor.

3.5 I/O Alternatives

HBM2 relies on simple unterminated signaling across a silicon interposer with small-geometry, high-density wiring. While these I/Os require very little energy, they have very short reach; the small-geometry wiring exhibits a physical bandwidth that is inversely proportional to the square of the length, similar to on-chip wiring. The HBM2 PHYs are located in a stripe near the middle of the base layer die and must be placed within a few millimeters of the processor. In practice, the I/O currently employed by HBM2 can travel roughly 5 to 7 mm on an interposer while maintaining unterminated signaling at 2 Gb/s. This distance depends strongly on the thickness, width and spacing of the wires. For example, a copper wire that is 1.5 μm thick, 1.0 μm wide, and spaced 2.5 μm from neighboring wires is limited to about 5 mm for 2 Gb/s unterminated signaling. Thicker and wider wires that use increased spacing can reach longer distances, but the achievable bandwidth density along the edge of the processor will decrease proportionally.

Since a key premise of the FGDRAM architecture is the placement of the DRAM PHYs close to the banks spread across the DRAM die, any practical I/O signaling technology must efficiently move data at least a centimeter from the far side of a DRAM die. Furthermore, this signaling technology should be more efficient over this distance than the on-die data transport energy; otherwise, any savings of on-die data transport energy simply becomes increased I/O energy.

Pseudo-Open Drain Signaling. In the baseline evaluation, we assume a conservative design similar to high-speed GDDR5 1.2V terminated pseudo-open drain logic (PODL) signaling technology. These existing DRAMs support 8 Gb/s data rates over the required distances, so it is a straightforward baseline. Since this paper focuses primarily on the DRAM architecture, we use this conservative I/O as the baseline in our evaluations.

Ground-Referenced Signaling. A promising alternative signaling technique is *Ground-Referenced Signaling* (GRS) [34]. This type of single-ended signaling has several advantages over PODL. For example, the current consumed by the line driver is essentially constant, except for a small ripple at the data rate. Since this ripple is at very high-frequencies it is easily filtered by on-chip bypass capacitance. The fact that this current is not data-dependent helps to mitigate simultaneous switching output (SSO) noise, which is often a limiting factor for single-ended memory interfaces [37]. This eliminates the need to use DBI coding [39] and the associated signals on

the data buses. In addition, GRS only uses the ground network to complete the signal return path by signaling "about" ground, which forces the signaling current to flow in the tight loop created by the lane wire and the nearest ground conductor. This also helps to reduce common-impedance return-path noise since the return path is well-defined and ground is almost always the lowest impedance network. Also, the ground network is easily the best reference for a single-ended interface because it is low impedance and common to all. Another important advantage of the GRS transceiver system is that the entire path for strobe and data are matched to have identical delay and delay-sensitivities. This ensures the relative timing between strobe and data is preserved in the presence of process, temperature, and voltage variation. It also means that the relative timing between strobe and data tracks together in the presence of power supply noise; this has been demonstrated at signaling rates up to 20 Gb/s with PHYs built in a logic process.

Higher Bandwidth Signaling. These signaling technologies may enable faster data rates than the 8 Gb/s assumed in this paper. Supporting higher data rates would allow fewer PHYs and signals between the processor and the DRAM. Ideally, the wire bandwidth would be sufficient to allow adequate memory bandwidth to be routed directly over an organic package substrate, enabling more traditional (less expensive) multi-chip module (MCM) assembly and obviating the requirement for assembly on a silicon interposer. Because the wires on an organic package have virtually distance-independent physical bandwidth (up to several centimeters) this is a viable approach. In other words, when using high-speed signaling over a terminated link on an organic package, energy consumption does not depend on wire length. This is due to the fact that the wire is relatively short (*e.g.,* less than 40 mm) and frequency-dependent attenuation is primarily due only to the parasitic capacitances of the micro-bump pads, ESD protection devices, and I/O devices at each end of the link. The additional high-frequency attenuation from the losses in the longest package traces is less than 1 dB (*i.e.,* 10%) for signaling rates up to 20 Gb/s, even when using low-cost packaging technology.

This can enable systems that require many DRAM devices (or stacks) co-populated with a processor in a package. Currently, it is impossible to use unterminated 2 Gb/s signaling over a silicon interposer and reach a second DRAM device on the far side of a closer DRAM device. The problem is that the bandwidth of the wires that connect the distant DRAM devices is significantly lower because they must be approximately 3× longer than the wires that connect the adjacent DRAM devices. Signaling over an organic substrate enables larger packages with more stacks of DRAM than current HBM2 systems.

The benefits of these alternative signaling technologies could apply to the alternative QB-HBM baseline as well. Thus, we consider only the conservative PODL signaling in out subsequent evaluation of the proposed FGDRAM architecture.

3.6 Design Assumptions

While we describe a 1 TB/s FGDRAM design with certain assumptions regarding the underlying process technology, the architectural approach is generally applicable under a range of different technology scaling assumptions. In general, we compare to a QB-HBM baseline that assumes identical aspects of a contemporary DRAM process. If different assumptions are made, they generally have a similar consequence in both our proposed FGDRAM design and the QB-HBM baseline.

Table 1: GPU Configuration

#SMs, Warps/SM, Threads/Warp	60, 64, 32
L2 cache (size, assoc., block, sector)	4MB, 16-way, 128B, 32B
DRAM	QB-HBM or FGDRAM

TSV scaling. We assume that we are able to drive signals through the TSVs at 4× the data rate of contemporary HBM2 DRAMs. As a result, we need the same number of signal TSVs as current HBM2 designs for both the QB-HBM and FGDRAM architectures. This assumption may be aggressive, but if additional TSVs are required, the impact would affect QB-HBM and FGDRAM equally. Future process technologies may reduce the TSV pitch, enabling more TSVs in a smaller area. Again, any area benefit of this approach would apply equally to FGDRAM and QB-HBM.

Stack height. We assume a 4-high DRAM stack in our analysis. It may be practical and economical to construct stacks with 8 or more devices going forward. Adding additional bandwidth to a stack via additional dies requires more TSVs and additional PHYs on the base layer. If additional dies in a stack are added only to provide additional capacity, then an approach similar to placing multiple DIMMs on a shared DRAM bus can be used. Additional dies in a stack act like another rank, with multiple dies sharing the TSVs much like a conventional DRAM bus. There may be some impact on peak data rates in this scheme, as there will be additional capacitive loading on the TSV channels. This may push tall stacks towards having more TSVs running at lower data rates to compensate.

Non-stacked DRAMs. All of the advantages of the FGDRAM architecture would apply to a non-staked DRAM as well. A single DRAM die built using the FGDRAM design could have the DRAM PHYs in the strips where the TSV arrays are located in the proposed stacked design. This approach can allow further efficient bandwidth scaling of traditional GDDR-class memories. Coupled with high-speed, distance-efficient I/O technologies like GRS, it could enable systems with large package substrates to simply use many non-stacked DRAMs to provide the bandwidth required, eliminating the overheads of stacked DRAMs.

4 METHODOLOGY

4.1 Simulation Details

GPU Model: We simulate a modern GPU-system based on the NVIDIA Tesla P100 chip [30] (configured as shown in Table 1) on a detailed GPU and memory simulator. We model the details of the compute and graphics pipelines, as well as the interconnect and caches. The caches are sectored (32B DRAM atom) for higher efficiency [35].

Memory Controller and DRAM: The baseline memory controller model is optimized for harvesting maximum bandwidth and thus deploys deep request buffers and aggressive request reordering to find row hits, batched write draining based on watermarks to reduce write-to-read and read-to-write turnarounds, and an address mapping policy designed to eliminate camping on banks and

Table 2: DRAM Configurations

Category		HBM2	QB-HBM	FGDRAM
channels/die (4-die stack)		4 (16)	16 (64)	128 (512)
banks/channel		16	4	2 pseudobanks
grains/bank		N/A	N/A	4
row-size/activate		1KB	1KB	256B
data I/Os/die (4-die stack)		256 (1024)	256 (1024)	256 (1024)
datarate/pin		2 Gb/s	8 Gb/s	8 Gb/s
bandwidth/channel or grain		16 GB/s	16 GB/s	2 GB/s
bandwidth/die		64 GB/s	256 GB/s	256 GB/s
bandwidth/4-die stack		256 GB/s	1 TB/s	1 TB/s
timing parameters in *ns* unless specified	common	tRC=45, tRCD=16, tRP=16, tRAS=29 tCL=16, tRRD=2, tWR=16, tFAW=12 tWTRl=8, tWTRs=3, tWL=2 clks		
	tBURST	2	2	16
	$tCCD_L$	4	4	16
	$tCCD_S$	2	2	2
	activates in tFAW	8	8	32

channels due to pathological access strides (similar to the baseline in [7]). The structural differences between QB-HBM and FGDRAM are reflected in the hierarchical composition of their building blocks and in the timing parameters in Table-2. The memory controller's internal state machine, and timing manager models these changes, and also models the fact that the FGDRAM command interface is arbitrated between eight grains.

Workloads: We evaluate memory intensive regions of 26 CUDA applications from the Rodinia [8] and Lonestar [4] suites, exascale workloads [1, 10, 14, 27, 29, 43] (*CoMD, HPGMG, lulesh, MCB, MiniAMR, Nekbone*), a hidden convolution layer from GoogLeNet [40], as well as two well-known memory-bound applications with disparate access patterns, STREAM and GUPS [11], to show the effect of our proposals on the spectrum of applications executed by a GPU. We also present results for 80 graphics workloads representing modern game engines and rendering pipelines (both shader- and render-backend-generated traffic is simulated). These applications are from the domains of professional graphics and games.

4.2 Energy and Area Model

For estimating the energy and area of HBM2, QB-HBM and FGDRAM devices, we use the methodology of previous work on subchannels [6] which was based on the model from Rambus [44]. In essence, we use detailed floorplans of DRAM dies and devices to estimate the areas of different blocks and the lengths of various datapath components traversed by bits from a DRAM cell to the GPU's pins, estimate the capacitive loading on these components using DRAM technology parameters for a 28nm node (scaled from 55nm [44]), and use appropriate switching activities to obtain the energy consumed to precharge a bank and activate a row, and in performing reads and writes. Table 3 enumerates the energy consumed for different operations for the different DRAM types. The row-activation energy, which is the sum of precharge and activate energies, differs based on the DRAM row activation granularity. The datapath energy has three components - i) traversing the LDLs and MDLs from the row-buffer to the GSAs (pre-GSA), ii) traversing the path from the GSAs to the DRAM I/Os (post-GSA), and iii) traversing the I/O channel between the DRAM and GPU. The first component is not data dependent as the LDLs and MDLs are precharged to a middle voltage before every bit transfer, the second depends on the data toggling

Table 3: DRAM Energy.

Component	HBM2	QB-HBM	FGDRAM
Row activation (*pJ*)	909	909	227
Pre-GSA data movement (*pJ/b*)	1.51	1.51	0.98
Post-GSA data movement (*pJ/b*)*	1.17	1.02	0.40
I/O (*pJ/b*)*	0.80	0.77	0.77

* at 50% activity

rate (Table 3 shows the value at 50% toggle rate), and their sum is the on-die data movement energy. Note that while FGDRAM has much lower overall data movement energy compared to HBM2 by design, even QB-HBM has more efficient data movement than HBM2 due to the reduced distance between the arrays and the I/O blocks. The final component is the I/O energy. In HBM2 it is influenced by the data switching activity, but in QB-HBM and FGDRAM the energy of the high-speed I/Os is determined by the termination energy and thus depends primarily on the number of 1 values in the data (Table 3 shows the values at 50% 1s).

5 RESULTS

This section, compares the energy-efficiency and performance of two iso-bandwidth DRAM systems: a QB-HBM stack and an FGDRAM stack. We also analyze the incremental area that would be required by these designs over current HBM2 devices.

5.1 Energy Improvement

Figure 8 shows the total DRAM energy consumed per bit by GPU compute applications using QB-HBM and FGDRAM . The compute workloads are grouped into two categories, with the first set (*dmr* to *pathfinder*) containing applications that use a small fraction (less than 60%) of the aggregate DRAM bandwidth, and the second consisting of memory intensive applications (*GUPS* to *STREAM*) which are more likely to be DRAM power (=*energy-per-bit*×*bandwidth*) limited. We find that the FGDRAM architecture significantly reduces energy across all the workloads by simultaneously reducing both the activation and data movement energies.

Naturally, the reduction in activation energy is most beneficial for applications with low row locality (*sssp*, *MCB*, *dmr*, *GUPS*, *nw*, *bfs*, *sp*, *kmeans*, *MiniAMR*). Similar to classic *GUPS*, applications *dmr*, *sssp*, *sp*, *bfs* and *MCB* have low *intrinsic* row-buffer locality as they perform many sparse data-dependent loads - *i.e.*, pointer chasing. On the other hand, *kmeans*, *nw*, and *MiniAMR* suffer from inter-thread interference at the row-buffers, and therefore have low *effective* row locality as a natural consequence of highly threaded processors [6]. The FGDRAM architecture reduces the fundamental activation granularity to 256 Bytes from 1 KB in QB-HBM. This reduces row-overfetch, and activation energy across the entire benchmark suite by 65% on average.

On the other hand, data movement energy is also either the largest or, at the least, a very significant fraction of overall energy in QB-HBM for many of the workloads. While data movement energy varies between applications depending on their data toggle rates (*e.g., HPGMG* has higher data movement energy compared to *srad_v2* in QB-HBM), it is primarily determined by the distance the data must travel between the sense-amplifiers and the I/O pads. By reducing this distance, FGDRAM reduces the average data movement

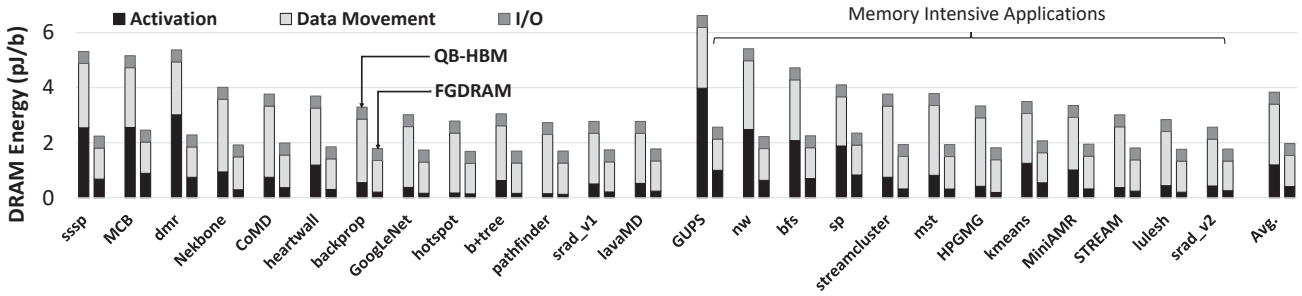


Figure 8: DRAM Access Energy Per Bit (lower is better). Applications are divided into two groups based on their bandwidth utilization in the baseline QB-HBM system. Within each group, the applications are sorted in descending order of energy-per-bit in the QB-HBM baseline.

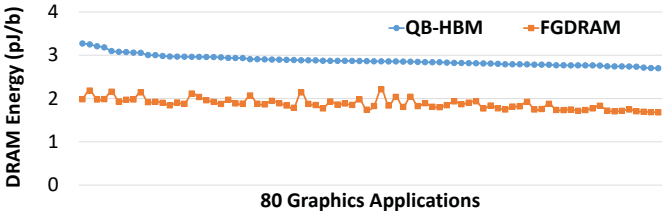


Figure 9: DRAM energy consumed by graphics applications (gaming, rendering, and professional graphics) (lower is better). Applications are sorted in descending order of energy consumption in the QB-HBM baseline system.

energy by 48%. This key benefit of the FGDRAM architecture is effective across the board for all applications, but is most significant for those memory intensive applications where data movement energy dwarfs the activation energy due to high row locality (*streamcluster*, *mst*, *HPGMG*, *STREAM*, *LULESH*). Previous work that focused solely on activation energy reduction ([6, 9, 12, 32, 42, 45]) is ineffective at addressing this important energy bottleneck.

The I/O energies are the same for both QB-HBM and FGDRAM architectures (as explained in Section 4). Using GRS technology for the I/O in FGDRAM and QB-HBM would slightly increase the I/O energy component from 0.43 pJ/bit to 0.54 pJ/bit, but would enable the benefits offered by GRS as outlined in Section 3.5.

Overall, on average, FGDRAM consumes only 1.95 pJ/bit, a 49% improvement over QB-HBM which consumes 3.83 pJ/bit. Notably, FGDRAM is able to reduce the DRAM energy consumption of applications at both ends of the row locality spectrum by addressing all sources of energy consumption in a holistic manner. Consequently, both *GUPS* and *STREAM* are able to meet the energy target of 2 pJ/bit needed for future high-bandwidth systems.

Figure 9 shows the per-access energy for graphics applications. Graphics applications tend to have higher row-buffer locality than compute applications, and thus lower energy in the QB-HBM baseline. Overall, FGDRAM reduces the baseline QB-HBM DRAM energy by 35%, primarily by reducing the data movement energy.

5.2 Performance

Figure 10 demonstrates the performance of a GPU with FGDRAM normalized to a baseline GPU with an iso-bandwidth QB-HBM system. Depending on access characteristics, applications either see improved or unchanged performance with FGDRAM. Memory intensive benchmarks that suffer from frequent row conflicts in QB-HBM and access a few bytes per activated row (*GUPS*, *nw*, *bfs*, *sp*, *kmeans* and *MiniAMR*) are helped by FGDRAM's i) increased concurrency that allows more requests to be overlapped in time across different pseudobanks in a grain, and across the grains on a die, and ii) higher row activate rates made possible by smaller activation granularity, *i.e.*, more activates in the same *t*FAW period. These two factors increase the performance of such irregular memory-intensive applications significantly — *GUPS* (3.4×), *nw* (2.1×), *bfs* (2.1×), *sp* (1.6×), *kmeans* (1.6×) and *MiniAMR* (1.5×).

On the other hand, memory-intensive applications with regular access patterns (*STREAM*, *streamcluster*, *LULESH*) that utilized a large fraction of the QB-HBM bandwidth, have very little change in performance when using the iso-bandwidth FGDRAM system as they continue to enjoy high utilization of the same available bandwidth. The small benefits observed in these applications with FGDRAM are due to a secondary benefit of the highly partitioned FGDRAM architecture that allows each grain to independently process reads and writes. This means that the write-to-read turnaround penalty on a grain is overlapped by data transfer from other grains, leading to more effective utilization of the same aggregate bandwidth compared to QB-HBM.

In general, applications that are not memory intensive remain unaffected by the change in memory technology. One exception is *MCB*, which is heavily bank-limited in the QB-HBM baseline. FGDRAM alleviates this bottleneck by using area-efficient pseudobanks, which improves bank-level-parallelism, and hence performance, compared to the iso-bandwidth QB-HBM. Across the simulated CUDA workloads, FGDRAM improves performance by 19% over an iso-bandwidth future QB-HBM system.

This outcome also shows that the increase in burst latency with FGDRAM (16ns) over that of QB-HBM (2ns), and the consequent increase in the empty-pipe memory access latency, has no impact on the performance of GPU applications. In theory, for a 1TB/s memory system, we need an extra 109 128-byte full cache line requests to cover the additional bandwidth-delay product resulting from the 14ns increase in the unloaded memory latency. This would mean an additional 2.8% more warps for a NVIDIA P100 machine that has 3840 warps already. In practice, however, by increasing

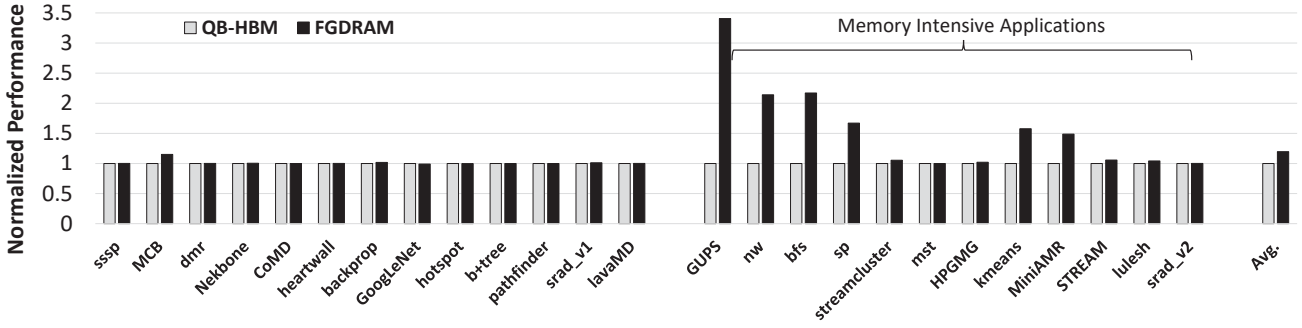


Figure 10: Performance normalized to QB-HBM baseline (higher is better). Applications are divided into two groups based on their bandwidth utilization in the baseline QB-HBM system.

parallelism through finer-grained channels that alleviates bank contention, FGDRAM reduces the queuing delay encountered by memory intensive applications in the QB-HBM architecture. This lowers the average DRAM access latency by 40% across the simulated workloads. In fact, a GPU already spends several hundred nanoseconds to fetch a request from DRAM [3]. The minor increase in burst latency to achieve better bandwidth utilization is easily justifiable in a throughput architecture like a GPU where most of the memory latency is from queuing delay rather than the unloaded DRAM latency.

Graphics applications running on GPUs leverage tiled accesses to improve cache utilization and compression to reduce DRAM bandwidth demand. As a result, most graphics applications are unable to fully utilize the bandwidth of the baseline QB-HBM system. In fact, even applications like raytracing, which one could presume to have sparse, random access patterns owing to incoherent rays, are heavily optimized to maximally utilize the on-die SRAM memory structures and produce sequential DRAM access streams that have high row-buffer locality [2]. Consequently, none of the graphics applications are activation rate limited and perform similarly on the two iso-bandwidth systems, with less than 1% different between QB-HBM and FGDRAM.

5.3 DRAM Area

To analyze the area overhead of the QB-HBM and FGDRAM stacks relative to an HBM2 stack, we use the detailed area model outlined in Section 4.

QB-HBM overhead: The QB-HBM (and FGDRAM) I/Os are operated at 4× the datarate of the HBM2 I/Os, and as a result there is no additional TSV area overhead. Also, the total number of banks on a die remains unchanged between QB-HBM and HBM2 (64), but the banks are rearranged so that each QB-HBM channel has 4 banks. To provide 4× the bandwidth of HBM2, 4× more banks have to be accessed in parallel in QB-HBM, which requires a proportional increase in GSA count and the wiring that connects the GSAs to the I/O buffers. These two factors increase the HBM2 area by 3.20% and 5.11% respectively. The additional control logic needed to manage the increased number of channels can be placed under the global wires, and only a small (0.26%) area is needed for additional decoding logic. Overall, the QB-HBM die is 8.57% larger than an HBM2 die.

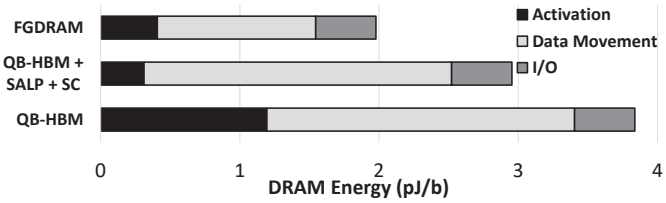


Figure 11: Average DRAM access energy per bit of baseline QB-HBM, enhanced QB-HBM+SALP+SC, and FGDRAM architectures.

FGDRAM overhead: The FGDRAM stack has the same bandwidth as QB-HBM and, thus, has an equal number of TSVs and GSAs. Like QB-HBM, FGDRAM has 3.20% more area than HBM2 due to the GSAs alone. However, since the data wires from the pseudobanks only have to be routed to local data TSVs, the routing is considerably simpler and thus the routing channel area overhead for data wires is almost negligible. A consequence of the reduced routing area is that the area for the additional control logic is not overlapped with these wires and increases the die size by 3.41% relative to HBM2. The next set of overheads for FGDRAM comes from the pseudobank organization. The overheads (3.47% over HBM2) are similar to those required for subchannels [6], and are due to extra LWD stripes, address decoupling latches, and control routing from the grain periphery to the pseudobanks. Overall, the FGDRAM stack is 10.36% larger than an HBM2 stack, and only 1.65% larger than the iso-bandwidth QB-HBM baseline.

The area overheads are derived with the assumption that the I/O and TSV frequency can be scaled to provide higher per pin bandwidth in QB-HBM and FGDRAM compared to HBM2. If TSV frequency cannot be increased, then both the baseline QB-HBM and FGDRAM would require 4× the number of TSVs to deliver the data from the banks to the I/O circuitry on the base die. Without any improvements in TSV pitch, this would make a QB-HBM die 23.69% larger than an HBM2 die, and our proposed FGDRAM die would be 1.45% larger than QB-HBM.

5.4 Comparison to Prior Work

Two significant drawbacks of the QB-HBM baseline system are the limited number of banks per channel (required to limit area overhead) which restricts bank-level parallelism, and the relatively

large 1 KB row-size. Two previous proposals, subarray-level parallelism (SALP) [26], and subchannels (SC) [6] can address these shortcomings. We modify a bank in QB-HBM to allow access to all row buffers belonging to the 32 constituent subarrays. Note that adjacent subarrays share a sense-amplifier stripe which limits parallelism somewhat. We also vertically segment the row and bank datapath into subchannels, so that the minimum activation granularity is 256 Bytes. This configuration, QB-HBM+SALP+SC, thus incorporates the best of prior proposals with the most feasible future scaling of the HBM architecture. The QB-HBM+SALP+SC approach has the area overheads of both QB-HBM (4× data wires and GSAs), as well as the overhead for maintaining access to each subchannel and subarray in a bank. Consequently, QB-HBM+SALP+SC is 3.2% larger than the QB-HBM baseline and 1.54% larger than our FGDRAM proposal.

Through our simulations, we found that the overlapping of activation and precharges to different subarrays from SALP and the increased activation rate possible from the smaller rows due to subchannels improves the performance of the baseline QB-HBM system to nearly identical levels as FGDRAM for all benchmarks. A few benchmarks, like *streamcluster*, have slightly improved performance over FGDRAM due to the slightly higher available bank-level parallelism from SALP. Overall, QB-HBM+SALP+SC has 1.3% better performance than FGDRAM. However, the energy efficiency of QB-HBM+SALP+SC is worse than that of FGDRAM. As shown in Figure 11, it fails to meet the the desired energy target of 2 pJ/b necessary to enable a 4 TB/s memory system. The subchannel technique reduces the minimum activation granularity of QB-HBM+SALP+SC to 256 bytes, and SALP additionally reduces the row conflict rate by keeping multiple rows open in a bank simultaneously. These mechanisms lead to 74% lower activation energy than QB-HBM (compared to 65% reduction with FGDRAM). Without any reduction in data movement energy, however, QB-HBM+SALP+SC is only able to improve energy by 23% over QB-HBM. This result falls well short of the 49% savings provided by FGDRAM. The benefits of SALP are orthogonal to FGDRAM, and could be applied to our proposed architecture as well if the incremental benefits warranted the additional area and complexity. To our knowledge, no prior work has focused on reducing the data movement energy in DRAM devices.

6 RELATED WORK

Previous work on DRAM energy reduction [9, 12, 28, 38, 42, 45], focused solely on reducing activation energy. Since, on average, more than 50% of the average DRAM access energy is attributable to on-die data movement, these techniques achieve a small fraction of the energy-efficiency of FGDRAM. The FGDRAM architecture readily lends itself to reducing the row-size by allocating lower-bandwidth I/Os to individual banks, which allows using the recently proposed area-efficient subchannels technique [6] to complement FGDRAM's low data movement energy.

Some DRAMs do have somewhat reduced data movement energy. The LPDDR4 die architecture [22], where the I/O pads for the two channels on the die are placed at opposite edges sees some reduction in data movement energy as a side-effect of the split interface. The Hybrid Memory Cube (HMC) [17] splits the DRAM die into several vaults with the banks stack above each other within the die stack. This technique educes the data movement within a vault since the data traveling from the far banks must only move through a few short TSVs. Overall, however, there is little savings since the data from a vault must travel through several buffers in the vault controller on the base layer and is then routed through a network on the base-layer to the appropriate I/O interface, finally traversing the serial channel on the PCB to the processor. These overheads make the HMC solution (10pJ/b [33]) less energy-efficient than FGDRAM. More importantly, scaling the bandwidth of HMC beyond its current capability will require addressing the same set of challenges as faced by HBM today. The FGDRAM architecture provides a roadmap for bandwidth scaling that can thus benefit the HMC as well.

7 CONCLUSION

Future GPUs and throughput CPUs that demand multiple TB/s of bandwidth require higher bandwidth and more energy efficient DRAMs. Our proposed FGDRAM architecture addresses bandwidth scalability in a performant and area efficient manner by unlocking the internal bandwidth of each bank in the stack. At the same time, we address the most significant sources of DRAM energy consumption: data movement energy and activation energy. Partitioning the DRAM into independent grains, each with adjacent local I/O, significantly reduces data movement energy within the DRAM die. Furthermore, we leverage an area-efficient technique to reduce the DRAM row size, saving activation energy. Synergistically, this technique also allows us to provide a mechanism to overlap activations and accesses within a grain inside a single DRAM bank. Overall, we reduce DRAM energy to 1.97 pJ/bit, a 49% improvement over an improved 4× bandwidth HBM2 variant. We show that the increase in concurrency within FGDRAM devices can improve performance relative to an iso-bandwidth QB-HBM system, particularly for activate-rate limited workloads. Overall, the proposed FGDRAM architecture provides a solution for the energy-efficient, high-bandwidth DRAM required for exascale systems and a range of other applications.

REFERENCES

[1] M. F. Adams, J. Brown, J. Shalf, B. V. Straalen, E. Strohmaier, and S. Williams. 2014. *HPGMG 1.0: A Benchmark for Ranking High Performance Computing Systems.* Technical Report. Lawrence Berkley National Laboratory. LBNL-6630E.

[2] T. Aila and T. Karras. 2010. Architecture Considerations for Tracing Incoherent Rays. In *Proceedings of High Performance Graphics.*

[3] M. Andersch, J. Lucas, M. Alvarez-Mesa, and B. Juurlink. 2015. On Latency in GPU Throughput Microarchitectures. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS).* 169–170.

[4] M. Burtscher, R. Nasre, and K. Pingali. 2012. A Quantitative Study of Irregular Programs on GPUs. In *Proceedings of the International Symposium on Workload Characterization (IISWC).* 141–151.

[5] S. Cha, S. O, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, J. H. Ahn, and N. S. Kim. 2017. Defect Analysis and Cost-Effective Resilience Architecture for Future DRAM Devices. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA).*

[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 *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA).*

[7] N. Chatterjee, M. O'Connor, G. H. Loh, N. Jayasena, and R. Balasubramonian. 2014. Managing DRAM Latency Divergence in Irregular GPGPU Applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC).*

[8] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S-.H. Lee, and K. Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the International Symposium on Workload Characterization (IISWC).* 44–54.

[9] E. Cooper-Balis and B. Jacob. 2010. Fine-Grained Activation for Power Reduction in DRAM. *IEEE Micro* 30, 3 (May/June 2010), 34–47.

[10] Coral 2014. CORAL Benchmarks. https://asc.llnl.gov/CORAL-benchmarks/. (2014).

[11] J. Dongarra and P. Luszczek. 2005. Introduction to the HPCChallenege Benchmark Suite. ICL Technical Report ICL-UT-05-01. (2005).

[12] H. Ha, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz. 2016. Improving Energy Efficiency of DRAM by Exploiting Half Page Row Access. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[13] Q. Harvard and R. J. Baker. 2011. A Scalable I/O Architecture for Wide I/O DRAM. In *Proceedings of the International Midwest Symposium on Circuits and Systems (MWSCAS)*.

[14] M. A Heroux, D. W. Doerfler, Paul S. Crozier, J. M. Wilenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. 2009. Improving Performance via Mini-applications. Sandia Report SAND2008-5574. (2009).

[15] Intel. 2016. An Intro to MCDRAM (High Bandwidth Memory) on Knights Landing. (2016). https://software.intel.com/en-us/blogs/2016/01/20/an-intro-to-mcdram-high-bandwidth-memory-on-knights-landing.

[16] D. James. 2010. Recent Advances in DRAM Manufacturing. In *Proceedings of the SEMI Advanced Semiconductor Manufacturing Conference*. 264–269.

[17] J. Jeddeloh and B. Keeth. 2012. Hybrid Memory Cube – New DRAM Architecture Increases Density and Performance. In *Symposium on VLSI Technology*.

[18] JEDEC. 2009. *JEDEC Standard JESD212: GDDR5 SGRAM*. JEDEC Solid State Technology Association, Virginia, USA.

[19] JEDEC. 2012. *JESD79-4: JEDEC Standard DDR4 SDRAM*. JEDEC Solid State Technology Association, Virginia, USA.

[20] JEDEC. 2013. *JEDEC Standard JESD235: High Bandwidth Memory (HBM) DRAM*. JEDEC Solid State Technology Association, Virginia, USA.

[21] JEDEC. 2014. *GDDR3 Specific SGRAM Functions in JEDEC Standard JESD21-C: JEDEC Configurations for Solid State Memories*. JEDEC Solid State Technology Association, Virginia, USA.

[22] JEDEC. 2014. *JESD209-4: Low Power Double Data Rate 4 (LPDDR4)*. JEDEC Solid State Technology Association, Virginia, USA.

[23] JEDEC. 2015. *JEDEC Standard JESD235A: High Bandwidth Memory (HBM) DRAM*. JEDEC Solid State Technology Association, Virginia, USA.

[24] JEDEC. 2016. *JEDEC Standard JESD232A: Graphics Double Data Rate (GDDR5X) SGRAM Standard*. JEDEC Solid State Technology Association, Virginia, USA.

[25] B. Keeth, R. J. Baker, B. Johnson, and F. Lin. 2008. *DRAM Circuit Design - Fundamental and High-Speed Topics*. IEEE Press.

[26] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. 2012. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 368–379.

[27] S. Layton, N. Sakharnykh, and K. Clark. 2015. GPU Implementation of HPGMG-FV. In *HPGMG BoF, Supercomputing*.

[28] Y. Lee, H. Kim, S. Hong, S. Hong, and S. Kim. 2017. Partial Row Activation for Low-Power DRAM System. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*.

[29] J. Mohd-Yusof and N. Sakharnykh. 2014. Optimizing CoMD: A Molecular Dynamics Proxy Application Study. In *GPU Technology Conference (GTC)*.

[30] NVIDIA. 2016. NVIDIA Tesla P100 Whitepaper. (2016). https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[31] NVIDIA 2017. NVIDIA GeForce GTX 1080: Gaming Perfected. (2017). http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf.

[32] S. O, Y. H. Son, N. S. Kim, and J. H. Ahn. 2014. Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 337–348.

[33] T. Pawlowski. 2011. Hybrid Memory Cube (HMC). In *HotChips 23*.

[34] J. Poulton, W. Dally, X. Chen, J. Eyles, T. Greer, S. Tell, J. Wilson, and T. Gray. 2013. A 0.54pJ/b 20Gb/s Ground-Referenced Single-Ended Short-Reach Serial Link in 28nm CMOS for Advanced Packaging Applications. *IEEE Journal of Solid-State Circuits* 48, 12 (December 2013), 3206–3218.

[35] M. Rhu, M. Sullivan, J. Leng, and M. Erez. 2013. A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 86–98.

[36] T. Schloesser, F. Jakubowski, J. v. Kluge, A. Graham, S. Selsazeck, M. Popp, P. Baars, K. Muemmler, P. Moll, K. Wilson, A. Buerke, D. Koehler, J. Radecker, E. Erben, U. Zimmerman, T. vorrath, B. Fischer, G. Aichmayr, R. Agaiby, W. Pamler, and T. Scheuster. 2008. A $6f^2$ Buried Wordline DRAM Cell for 40nm and Beyond. In *Proceedings of the International Electron Devices Meeting (IEDM)*. 1–4.

[37] R. Schmitt, J.-H. Kim, W. Kim, D. Oh, J. Feng, C. Yuan, L. Luo, and J. Wilson. 2008. Analyzing the Impact of Simultaneous Switching Noise on System Margin in Gigabit Single-Ended Memory Systems. In *DesignCon*.

[38] Y. H. Son, S. O, H. Yang, D. Jung, J. H. Ahn, J. Kim, J. Kim, and J. W. Lee. 2014. Microbank: Architecting Through-Silicon Interposer-Based Main Memory Systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.

[39] M. R. Stan and W. P. Burleson. 1995. Bus-Invert Coding for Low-Power I/O. *IEEE Transactions on Very Large Scale Integraion (VLSI) Systems* 3, 1 (March 1995), 49–58.

[40] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erha, V. Vanhoucke, and A. Rabinovich. 2015. Going Deeper With Convolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*.

[41] K. Tran and J. Ahn. 2014. HBM: Memory Solution for High Performance Processors. In *Proceedings of MemCon*.

[42] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. Jouppi. 2010. Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 175–186.

[43] O. Villa, D. R. Johnson, M. O'Connor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, S. W. Keckler, and W. J. Dally. 2014. Scaling the Power Wall: A Path to Exascale. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.

[44] T. Vogelsang. 2010. Understanding the Energy Consumption of Dynamic Random Access Memories. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 363–374.

[45] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie. 2014. Half-DRAM: a High-bandwdith and Low-power DRAM System from the Rethinking of Fine-grained Activation. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 349–360.