Architecting an Energy-Efficient DRAM System For GPUs

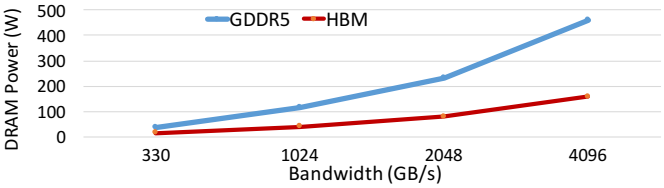
Niladrish Chatterjee*, Mike O'Connor*†, Donghyuk Lee*, Daniel R. Johnson*,
Stephen W. Keckler*†, Minsoo Rhu*, William J. Dally*
*NVIDIA †The University of Texas at Austin
{nchatterjee, moconnor, donghyukl, djohnson, skeckler, mrhu, bdally}@nvidia.com

***Abstract*—This paper proposes an energy-efficient, high-throughput DRAM architecture for GPUs and throughput processors. In these systems, requests from thousands of concurrent threads compete for a limited number of DRAM row buffers. As a result, only a fraction of the data fetched into a row buffer is used, leading to significant energy overheads. Our proposed DRAM architecture exploits the hierarchical organization of a DRAM bank to reduce the minimum row activation granularity. To avoid significant incremental area with this approach, we must partition the DRAM datapath into a number of semi-independent *subchannels*. These narrow subchannels increase data toggling energy which we mitigate using a static data reordering scheme designed to lower the toggle rate. This design has 35% lower energy consumption than a die-stacked DRAM with 2.6% area overhead. The resulting architecture, when augmented with an improved memory access protocol, can support parallel operations across the semi-independent subchannels, thereby improving system performance by 13% on average for a range of workloads.**

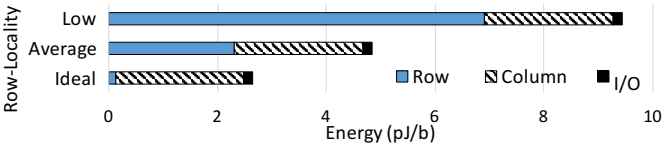
I. Introduction

Graphics Processing Units (GPUs) and other throughput processing architectures have scaled performance through simultaneous improvements in compute capability and aggregate memory bandwidth. To continue on this trajectory, future systems will soon require more than 1 TB/s of bandwidth, and systems used in Exascale systems are projected to require 4 TB/s of bandwidth for each processor [1]. Satisfying this increasing bandwidth demand, without a significant increase in the power budget for the DRAM, is a key challenge. As Figure 1a shows, the off-package high-speed signaling across a PCB used by traditional bandwidth-optimized GDDR5 memories can consume a significant portion of the system energy budget, becoming prohibitive as bandwidths scale beyond 1 TB/s. The highest bandwidth chips have recently adopted on-package stacked DRAM [2]–[4] to, in part, address this energy consumption challenge. These stacked memories, such as High Bandwidth Memory (HBM), allow the processor and memory to communicate via short links within a package, thereby reducing the cost of data transfer on the interface between the DRAM stack and the processor die. While this improved signaling reduces the I/O energy (the energy on the link between the DRAM and processor dies), the energy required to move data from the DRAM bit cells to the I/O interface remains similar. Consequently, as we project the bandwidth demands of future GPUs, further energy reductions within the DRAM itself are required to enable future high-bandwidth systems.

In addition to the I/O energy used to transfer the data from the DRAM to the host processor, the energy for a DRAM access can be decomposed into two primary components, the *row energy* or the energy to precharge a bank and activate a row, and the *column energy* or the energy to access a subset of data from the activated row and move it to the I/O pins. The column energy of a DRAM access primarily depends on the distance over which the data is moved, the rate of switching on this data path, and the capacitance of these wires. The row energy is determined by the fixed activation and precharge energy required to activate a DRAM row, sense and latch its data into the row buffer, restore the bit cell values, and precharge the bit lines. The per-access component of row energy is the sum of activation and precharge energy averaged over the number of bits accessed within a row. As a result, the row energy is highly dependent on the spatial locality of the memory access stream. With higher spatial locality, the cost of opening a new row is amortized over a larger number of bit transfers. As shown in Figure 1b, row energy accounts for nearly half of DRAM energy on average in the workloads we study, and it can dominate DRAM energy in low-locality workloads. Low row locality can also cause performance problems, limiting utilization of DRAM bandwidth and leading to workloads limited by the DRAM row activation rate.



(a) GPU memory system power vs bandwidth



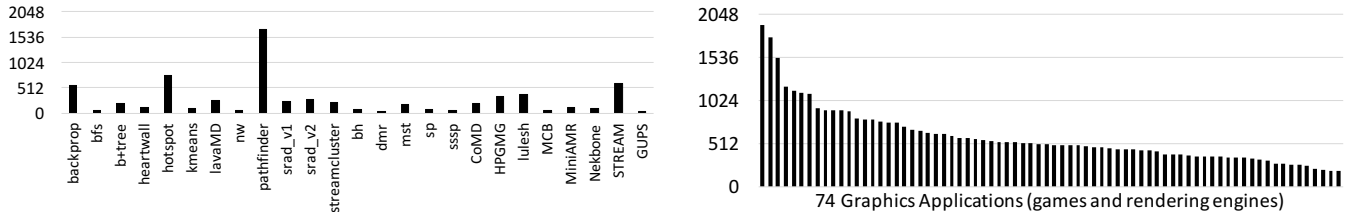
(b) Stacked memory energy vs. locality

Figure 1. GPU memory energy consumption.

To address these problems, we propose an energy-optimized DRAM and memory controller architecture, called `subchannels`, which can also boost the performance of many memory-intensive GPU applications. The key idea is to partition the DRAM storage in an area-efficient manner to reduce wasted activation energy *and* partition the datapath in the DRAM array and the periphery to enable parallel operations to continue to utilize the full bandwidth of the DRAM.

This paper makes the following contributions:

- We present a detailed analysis of memory energy consumption in modern GPUs. This is comprised of a study



(a) Compute applications

(b) Graphics applications

Figure 2. Bytes accessed per activate from a 2KB row.

of the row access and data toggling patterns in compute and graphics workloads, and a detailed energy model for 3-D stacked high-bandwidth memory devices.

- We propose an area-efficient DRAM architecture, `subchannels`, to reduce row energy, and augment it with a data-burst reordering mechanism to cut down column energy. The proposed design reduces DRAM energy consumption by 35% over the baseline.
- We leverage the abundant parallelism in GPU applications to design a DRAM and memory controller architecture that provides an average of 13% improvement in system performance compared to an HBM solution across a wide range of workloads.

GPUs and other throughput processors routinely use specialized memory devices to meet their high bandwidth needs. In this paper, we show that future DRAM designs require us to consider both the nuances of DRAM microarchitecture and the GPU's memory access behavior. The `subchannels` architecture leverages such co-design to provide significant energy and performance benefits with minimal additional area.

II. Row Locality in GPUs

Figure 2a shows the average number of bytes accessed per activated row in high-performance compute benchmarks (see Section VII for methodology details). In the majority of applications, fewer than 256 bytes are accessed from the 2KB row before the bank is precharged. There are two reasons for this behavior.

First, several GPU applications have intrinsically low locality. Applications that often perform data dependent memory accesses (pointer chasing) or sparse, irregular accesses to a data structure (*bfs, bh, dmr, mst, sp, sssp, MCB* and *MiniAMR*) naturally have low spatial correlation between successive accesses. *GUPS*, which is designed to randomly access memory, is the most extreme example of this behavior. All of these applications access one or two DRAM atoms (32B each) per activate.

Second, even when applications tend to access dense data structures, there is limited row-buffer locality due to interference between the large number of simultaneously executing threads. For example, the NVIDIA Tesla P100 [2], based on the Pascal architecture, contains 122,880 threads across all 60 Streaming Multiprocessors (SMs) accessing data in a total of 32 HBM2 channels with 16 banks each. Thus, on average, 240 threads contend for the single row-buffer in each bank. Consequently, even with deep memory controller queues and aggressive request reordering to harvest row locality, GPUs suffer from frequent bank conflicts. Even applications like *STREAM* that scan sequentially through input and output arrays are able to utilize only approximately a quarter of the row after activating it. *HPGMG, lulesh, Nekbone, kmeans, nw*, and *streamcluster* also exemplify this behavior. Two applications, *pathfinder* and *hotspot*, access the majority an of an activated row because they either have a very small working set size or very few active threads and, thus, do not have these problems.

Even though graphics applications demonstrate higher row-access locality than the compute applications (Figure 2b), in the majority of the cases, only a quarter of the row is utilized on an activate. Emerging graphics applications, such as ray tracing and particle systems, more closely resemble compute workloads in terms of row locality.

Since a row-activate operation is very energy-intensive (Section VII) a direct consequence of the observed low spatial locality is high effective row-energy. The energy cost of activating the whole row is averaged over only the small number of bits that are read or written. Few accesses-per-activate can also reduce bandwidth utilization in memory intensive benchmarks. Before we explain our proposal for mitigating these issues, we present a brief overview of the main building blocks of modern DRAMs.

III. High Bandwidth Graphics DRAM

A DRAM die is a collection of independent storage arrays called banks, each with its own row and column decoders. The data signals from the banks are connected via a shared bus to an I/O buffer (Figure 3a). The I/O buffer is the same size as a DRAM *atom*, the unit of a single memory transaction (32B in HBM and GDDR5), and is connected to the external I/O interface using a DDR-style interconnect (128-bits wide in HBM). In stacked memories, the I/O buffer connects to the external interface through on-die wires, inter-die TSVs, and finally wires on the base-layer to the stack's edge. Each bank is organized as a 2-D array of *mats* (Figure 3b). Each mat is a 512×512 array of DRAM cells, and has a 512-bit sense-amplifier or *row-buffer*. A group of 32 mats in the horizontal direction, called a *subarray*, is activated in unison. Thus the 2KB row-buffer of the subarray is physically spread across the mats, and a DRAM atom is spread out over the entire subarray, with each mat contributing 8 bits to the 32B atom. A row-activation signal is hierarchical [5]–[7]; a Master Wordline (`MWL`) drives a Local Wordline (`LWL`) in each of the mats in the subarray, and each `LWL` turns on the access transistors of a row of cells in a mat. The `MWL` is fashioned in higher level metal (M2) and has $4\times$ the pitch of the `LWL` which is in silicided polysilicon. After a row is activated and the bits are read into

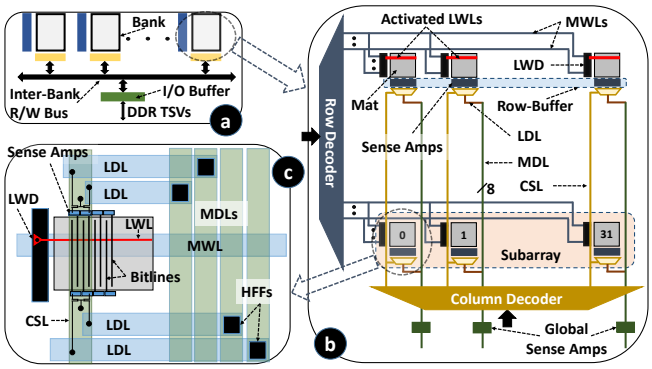


Figure 3. High Bandwidth Memory architecture, showing a single channel in a stack.

the local sense-amplifiers, a column command triggers a set of column select lines (CSLs) in each mat, which enables the data latched in the sense amplifier to be delivered to the DRAM periphery. The same CSL lines are asserted for each mat in the subarray. As shown in Figure 3(c), a CSL connects the sense-amplifier to the Master Data Lines (MDLs) via the Local Data Lines (LDLs) (both differential) [5], [8]. At the connection of each LDL-MDL pair is a helper flip-flop (HFF) which helps drive the MDL down to the Global Sense Amps (GSAs).

The 3 metal layers in a typical DRAM process include M1 for the vertical bitlines in the mat, M2 for the MWLs and LDLs in the horizontal direction, and M3 for the CSLs and MDLs. The LWLs are in silicided polysilicon. The M2 pitch is 4× the LWL pitch, while the M3 pitch is 4× the bitline pitch. Thus the height and width of the mat is dictated by the number of wiring tracks in these layers.

IV. Energy-efficient DRAM

To reduce DRAM row energy, we reduce the row-activation granularity, *i.e.*, the number of bits accessed by an activate command. We employ two techniques to achieve this without significant area overheads. First, a technique called *Master Wordline Segmentation* leverages the hierarchical structure of the DRAM wordline to turn on only part of a row [9]–[11]. Second, we map a DRAM atom such that it is contained entirely within a single activated subset of the row (instead of being dispersed over the whole row). Crucially, to maintain density, we choose to read out the DRAM atom from the activated region of the row using *only* the fraction of the total bank datapath that is available in that region. In effect, we create several narrow slices of the DRAM bank storage, as well as datapath from the cells to the I/O pins, each of which is called a *subchannel*. In the rest of this section we describe the architectural details of this subchannels architecture.

A. Reducing Activation Granularity

In existing DRAMs, when the master wordline (MWL) is asserted, it drives a local wordline (LWL) in *each* of the mats of a subarray by activating the corresponding local-wordline driver (LWD) in the mat's stripe. To achieve partial row activation, the wordline signal must reach the access transistor of only those cells that we wish to activate. One

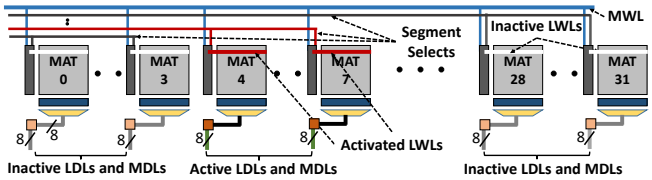
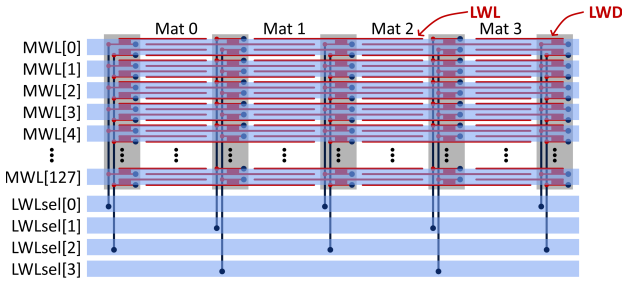
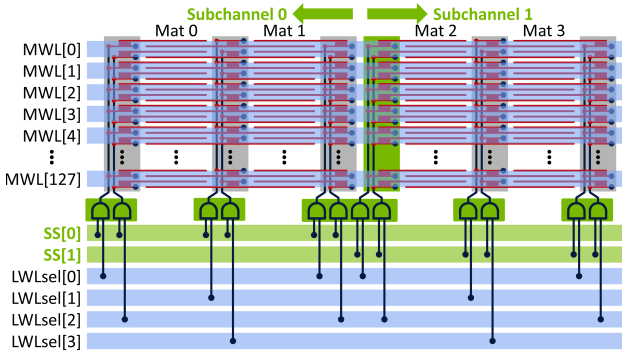


Figure 4. Master Wordline Segmentation. The LWLs are activated only in mats 4 through 7 for a 256-byte effective row size. Per-mat bandwidth is same as the baseline HBM architecture.



(a) Baseline - four mats in a subarray activated in unison



(b) Proposed - subarray partitioned into two groups of mats that can be activated independently

Figure 5. Implementation of a segmented wordline. We show a simplified example with four mats per subarray (32 in reality).

way to achieve this would be to segment the LWL, such that only a part of the LWL in every mat is turned on. However, since the LWL is laid out in silicided polysilicon inside the densest part of the DRAM chip, intrusive changes that disrupt the LWL layout are very area inefficient. Instead, we modify the relationship between a MWL and its subservient LWLs such that asserting a MWL leads to the assertion of the LWLs in only a few of the mats in the subarray while the other mats remain unaffected. Figure 4 shows this technique being employed to activate only 1/8th of a row. Even though the entire MWL has been driven, the corresponding LWLs have been activated only in mats 4 through 7 consuming an eighth of the original row-energy. When the MWL is driven, the segment-select signals (SS) are appropriately asserted to select the desired section of the row to activate.

To illustrate the implementation of this architecture in a density-optimized modern DRAM, we use the simple example in Figure 5a which shows an unmodified subarray consisting of four mats and four MWLs. Since a MWL is 4× wider than a LWL, each MWL is connected to four LWLs in a mat. In conjunction with the assertion of the MWL signal, a set of local wordline select signals, LWLsel [0:3], are

asserted to select one of the 4 LWLs connected to the MWL in question. To activate the LWLs in the first two mats (mats 0 and 1), and not in the other two, we modify the subarray as shown in Figure 5b. Since each LWD drives a LWL in a mat on either side in the baseline (Figure 5a), we introduce an extra LWD stripe between mats 1 and 2 to make sure that these two mats can be driven independently. Second, we add two Segment Select (SS) signals, each of which is responsible for a single group of mats. The SS signal is used as an enable for the LWD stripes belonging to a single group of mats, and is AND-ed to the decoded LWLsel signals. Thus if SS [0] is asserted, it turns on the LWDs of mats 0 and 1, in turn allowing the LWL selected by the LWLsel signals to be driven by the MWL. To divide a 32 mat subarray into 8 such groups, we will need 7 extra LWD stripes, one each at the boundary between two consecutive groups of mats and eight SS lines. The area overheads of this architecture, estimated to be 2.6%, are discussed in Section IV-C.

To use this architecture, the memory controller provides not just the row address with an activate, but also information about which segment must be activated (similar to Posted-Column commands [12]) in the form of a mask. This mask information is used to assert the SS signals and activate part(s) of the row. Sending a mask also makes it possible to activate multiple segments from a row simultaneously. Thus not only can we turn on different non-contiguous segments of a row, we can also turn on the entire row as in the baseline.

B. Subchannels

In the baseline, each 32B DRAM atom is interleaved across all the mats in a subarray and is read out into the 256 global sense-amplifiers (GSAs) in one internal DRAM cycle using all the MDLs. In our proposed architecture, this layout is modified such that a DRAM atom is interleaved across *only* the mats that are activated in unison, so that it can be retrieved in its entirety from these mats alone. Reading the DRAM atom from 1/8th of the mats in one cycle would require increasing the output width of each mat from 8 bits to 64 bits. However, increasing the datapath width (LDLs and MDLs) by $8\times$ increases the wiring tracks in the M2 and M3 layers, thereby increasing the height and width of the mat (Section III) and proportionally increasing the GSA area needed in the periphery. This leads to prohibitively high area overhead of 34.5% for high bandwidth mats (Section X), which was ignored by previous work that read an atom from a subset of mats [10], [11], [13]. Instead, we propose to keep the per-mat bandwidth unaltered and transfer each DRAM atom from the row-buffer over several cycles instead of one internal DRAM cycle. From the point of view of a DRAM transfer, it now appears that the data is stored and fetched from a channel which is 1/8th the bandwidth of a baseline channel with proportionally fewer mats and datapath resources (LDLs, MDLs, GSAs, and I/Os). We call each such narrow channel a *subchannel*. Section V-A details mechanisms to enhance the subchannels architecture by enabling parallel operations across subchannels to overcome the increased data transfer time.

Pipelined Burst: In the baseline HBM architecture, the whole DRAM atom is fetched from the row-buffer into the I/O buffer in a single internal cycle. However, with 8 subchannels, the same transfer takes 8 internal cycles due to the narrow datapath. Instead of waiting for the entire DRAM atom to fill up the I/O buffer before the data burst is initiated on the external I/O bus, we pipeline the internal and external burst. Thus the first beat of the data appears on the external I/O when the CAS latency time, tCAS, has elapsed after the column-read command, the same as in the baseline. However, the burst now takes 8 DDR cycles instead of 1 DDR cycle. For a single burst, multiple Column-Select-Lines (CSLs) must be asserted sequentially in a subchannel. To generate the sequence of CSLs, the column address sent with the CAS command is incremented using a small 3-bit ripple carry adder inside the DRAM in successive cycles of a burst, avoiding any extra traffic on the external column-address bus. The internal DRAM cycle time must be sufficiently short to enable back-to-back column accesses to each element of the burst in successive cycles. In current devices, the CAS-to-CAS delay within the same bank (or bank-group), tCCDL, defines this cycle time. This period is generally limited by the time required for the assertion of the CSL signal, the delay through the column-select mux enables, the time required for selecting bits from the local sense-amplifiers and for sending the selected bits to the GSAs. We assume that the DRAM can be engineered to meet the required cycle time on this path to allow gap-less reads of a full burst of data from a subchannel. Modern GDDR5 parts already support a tCCDL of 2ns [14], enabling gap-less bursts in a subchannel with HBM-like timings.

C. DRAM Overheads

There are two sources of area overheads for the subchannels architecture. First, the seven additional Local Wordline Driver (LWD) stripes required to split a row into 8 segments increases the subarray area by 1.19% (each LWD stripe is 5.7% the area of a mat). Second, additional metal wires are required for the segment selection signals. We place eight segment-select signals (SS) for every other subarray (required for parallel activates as described in Section V-A). Since the segment select signals are routed in the same metal layer (M2) as the Master Wordlines (MWLs) and Local Datalines (LDLs) in the horizontal direction, we estimate the area overhead by counting the number wiring tracks needed in this layer. Each subarray with 512 rows or Local Wordlines (LWLs) has 128 M2 routing tracks for MWLs (M2 pitch is $4\times$ the LWL pitch) and 16 M2 routing tracks for the 8 differential LDLs and 4 LWLsels. Thus the additional M2 tracks for SS increases DRAM area by 2.68%. Combining these two overheads (from additional LWDs and segment selection signals), each subarray is increased by 3.87%. Assuming the bank storage area to be 60% of the total die area, the effective area overhead turns to be 2.3%. The area overheads of the row and column address registers (also required to enable subchannel-level parallelism as described in Section V-A) is 0.3% of the DRAM die [15], [16], which puts the total area overhead of 8 subchannels at 2.6%.

Sending the subchannel mask from the memory controller to the DRAM requires six additional command I/O signals. The HBM architecture maintains two separate command interfaces. The column-command interface sends read/write

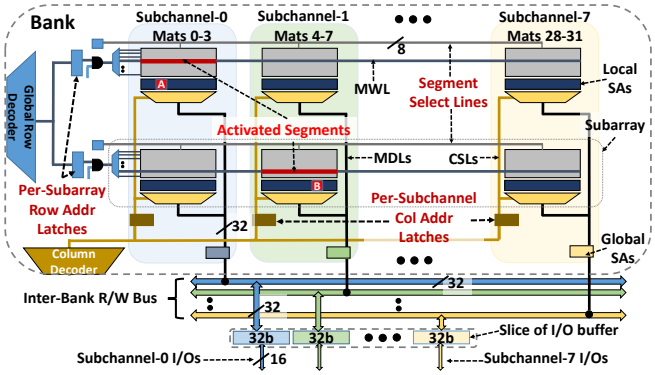


Figure 6. Subchannel Architecture. 8-subchannels per conventional DRAM bank. Two different 256 B rows have been activated in parallel in different subchannels, and different columns are being read in each. The row and column address latches, and the segment-select lines have been added to the baseline design.

commands and column addresses on two edges of the clock. Therefore, the column-command interface requires 4 additional signals to send the 8-bit subchannel mask with each command. The row-command interface sends activation commands and the row address over 4 edges of the clock. Thus, the 8-bit subchannel mask requires two additional signals to send the mask over 4 edges. The precharge commands have four unused bits which, coupled with the two additional signals, can send the subchannel mask over the 2 edges of a precharge command. Thus an 8-channel HBM stack requires 48 extra signal bumps. These extra bumps represent a 3.1% increase in the number I/O signals required for each stack.

V. Managing Narrow Subchannels

This section describes techniques to deal with two problems due to the reduced datapath width in `subchannels`. First, the effective bandwidth from a bank is reduced to 1/8th of the baseline, creating a major performance bottleneck (as shown by the `SC-8_No_Parallelism` bar in Figure 11). The second problem is less obvious. We found that the reduced datapath width also leads to higher switching activity on the internal and external data buses during a DRAM atom transfer, causing higher column and I/O energy (Figure 7).

A. Architecting Parallel Subchannels

The performance degradation when only one subchannel is active can be mitigated if all 8 subchannels are operated in parallel. We describe the changes needed in the DRAM and memory controller architecture to achieve this functionality.

Figure 6 shows the architecture of parallel subchannels in a bank. Each subchannel already has a subset of the mats and the datapath from the bank to the I/O buffer, including a subset of the Master Data Lines (`MDL`s), Global Sense Amplifiers (`GSA`s), the global I/O bus wires in the periphery. Also, due to pipelining of the burst, each subchannel only needs 1/8th the capacity of the I/O buffer and the I/O pins. Thus the datapaths of the 8 subchannels are completely independent. Also notice that if a 256B row is activated in a subchannel, a different row can potentially be activated in a different subchannel by driving another MWL and selecting a different segment as long as the rows are not in the same subarray. Since we assign a set of 8 Segment Select lines (`SS`) to every group of two consecutive subarrays, we can potentially activate a different row in different subchannels, as long as they are not in the same subarray-group. The main impediment to such parallel operation, however, is the row and column address decoding and control circuitry that is shared by the subchannels. So if a certain Column Select Line (`CSL`) is asserted in one subchannel, no other `CSL`s can be asserted in the other subchannels until that column operation completes, preventing parallel reads and writes across subchannels. Similarly, only one Master Wordline (`MWL`) can be active preventing parallel activates across subarrays.

Address Decoupling Registers: Adding a row-address latch per subarray and a column-address latch per subchannel solves this issue. As shown in Figure 6, the row-address latch decouples the driving of the `MWL` from the output of the row decoder. Once a row address is decoded and driven into the row-address latch of the corresponding subarray, the latch can drive the `MWL`, freeing up the row decoder to accept, decode and drive a different row address into the latch of another subarray in the next cycle leading to simultaneous activates of different rows. Similarly, the column-address latches allow the column decoder to assert different `CSL`s in different subchannels in successive cycles to enable parallel reads/writes from across subchannels.

Increased Performance: Creating parallel subchannels can not only recover the performance loss from narrower channels, it can improve the performance of many memory intensive workloads beyond the baseline. Most GPU workloads are latency insensitive [17] and are bottlenecked by the delivered bandwidth of the system. In memory-intensive applications with low spatial locality, the delivered bandwidth can be low when the bank-level parallelism (BLP) is not sufficient to cover for the tRC delay of precharging a bank and activating a row. Even if there is high BLP, if the accesses-per-activate is very low, the bandwidth utilization is restricted by the activate-rate limits posed by current delivery constraints (tFAW and tRRD). The `subchannels` architecture with parallel operation across subchannels alleviates both of these problems. First, due to more parallel subchannels, effectively there are more banks in the system making it possible to overlap row-activations to different rows of the same original bank. Second, due to the small activation granularity, each activate places a lower burden (approximately 1/8th the baseline) on the charge pumps that supply the high Vpp voltage to the `MWL`. Thus in the same tFAW period, the `subchannels` architecture can perform 32 activate operations while the baseline can only perform 4, accelerating activate-rate limited applications like the ones that exhibit data dependent irregular accesses. Even though the empty-pipe (unloaded) latency experienced by a DRAM request is 7ns higher with 8 subchannels than the baseline, the overall performance is improved by increasing the effective throughput of the DRAM system.

Command Coalescing: The address-decoupling registers and partitioned datapath allow overlapping commands to the same bank in different subchannels. However all subchannels in a channel share the address/command bus, and the same bank in different subchannels share the row

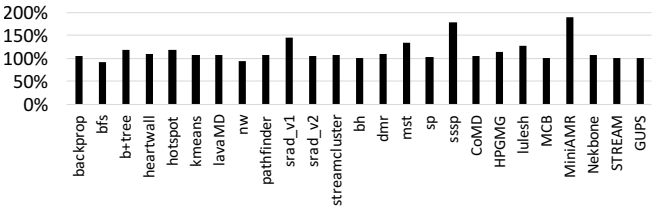


Figure 7. Datapath energy (column and I/O) for 8 Subchannels normalized to the baseline.

and column decoders of that physical bank. We propose a memory-controller optimization, *command coalescing*, that can alleviate the contention for these shared resources to improve subchannel performance.

When an application has high row-buffer locality, it might require activating the same row in different subchannels. Not only does this require more address/command bandwidth compared to the baseline HBM system, the activates to different subchannels must be separated by the inter-activate command timing delay (tRRD) and constrained by the four-activate window (tFAW). While more activates can be performed in the tFAW period due to the reduced row size of subchannels, activates are still subject to tRRD preventing back to back issue of activates to different subchannels. Today there is little pressure for DRAMs to reduce tRRD to the limit of the row-decoder cycle time since the rate of activates is also limited by tFAW. Since tRRD is more than a single cycle, it introduces a constant overhead between activates and consequently idle cycles on the data bus. Thus when applications access several DRAM atoms across multiple subchannels on the same row (high spatial locality), tRRD introduces a performance overhead that is not there in a baseline HBM channel, limiting the overlapping of activates in subchannels.

To circumvent this problem, we perform *activate coalescing* in the memory controller. In this mechanism, the memory controller scans the command queues to find activate commands directed to the same row in different subchannels, and issues a single activate command for all of these requests. The subchannel mask sent with the activate command is set appropriately such that the same MWL is turned on across all the subchannels needed. Activate coalescing thus allows subchannels to mimic the performance of the HBM baseline when there is high locality.

Similarly, the memory controller also performs *read coalescing* and *write coalescing*, issuing a single RD (or WR) command to multiple subchannels if the same column and bank is accessed in each. The output of the column-decoder is used along with the supplied subchannel mask to drive the desired CSLs in each of the selected subchannels (via the column-address registers). Thus the reads can progress completely in parallel across subchannels. Compared to the 64 columns in the original row, a 8-subchannel system will have only 8 columns per subrow, increasing the probability of finding such coalescing opportunities.

B. Column Energy in Narrow Subchannels

Column energy for a DRAM part can be calculated using the current drawn by column read (IDD4R) and write (IDD4W) operations. However, these values are specified by DRAM vendors assuming a 50% toggle rate of the datapath. With higher toggle rates, the capacitive load on the datapath has to be switched more frequently, leading to higher power dissipation. Using our energy model (Section VII), we found that for HBM, the column-energy is can vary between 1.5 pJ/bit when there is no toggling and 5.7 pJ/bit when the toggle rate is 100%. Taking this data toggling energy into account, we found that a design with 8 subchannels increases the column-energy of the Exascale, Rodinia, and Lonestar benchmarks by 23%, 10% and 25%, respectively, over the baseline case, with the *MiniAMR* benchmark suffering a 90% increase (Figure 7). Clearly, the narrow datapath in subchannels increases the switching activity on the wires.

Cause of Increased Toggling: The increase in switching activity with subchannels is due to the structure of the data elements in a DRAM atom. Often a 32B atom consists of eight 32-bit values or four 64-bit values for which the corresponding bytes of adjacent words are strongly correlated. This characteristic forms the basis for many memory compression schemes, *e.g.* [18]. With 8 subchannels, the internal 256-bit wide datapath is initially partitioned into 8 slices 32-bits wide. As the data moves to the I/O bus, the datapath narrows to 16-bits and the frequency doubles. Figure 8 shows how a given 32-byte DRAM atom is transferred over datapaths of different widths. In the baseline memory, the 32B DRAM atom initially moves across a 256-bit internal bus. All switching activity on this bus is due to switching between successive DRAM atoms. As the data moves to the DDR I/Os, the datapath narrows, and Figure 8a shows a straightforward mapping of how data is serialized onto the narrower 128-bit interface. As seen, the different corresponding bytes of 4-byte (or 8-byte) words line up, leading to relatively modest switching activity within the transfer of an atom; though there is still significant potential for switching activity between successive atoms.

With 8 subchannels, the data transfer using the straightforward baseline ordering on the 32-bit internal datapath is shown in Figure 8b. Since adjacent 4-byte data words tend to be more highly correlated than words 16-bytes away, the internal switching rate tends to decrease for benchmarks dominated by 32-bit datatypes. The increase in internal switching seen in the double-precision focused Exascale workloads arises because uncorrelated portions of a 64-bit value are transmitted back-to-back. This effect is exacerbated for increased switching behavior on the narrower 16-bit wide bus (Figure 8c).

Reordering to Reduce Toggle Energy: To reduce switching activity, we remap the data within a burst as it is written/read from the DRAM in a manner designed to cause highly correlated bits to be transmitted on the same wires in successive cycles. Based on the fact that commonly 32-bit and 64-bit data values from an array are sent within a DRAM atom, we developed a simple static ordering, shown in Figures 8d and 8e, that is applied to all DRAM atoms. Offsets of 8 bytes (well suited to double-precision values) are favored in successive cycles, then the remaining bytes at a 4-byte offset are transferred. The net result is that data with highly correlated 8-byte or 4-byte values tend to have reduced switching activity. Since there is more toggling energy spent in the narrow 16-bit interface and I/Os, we

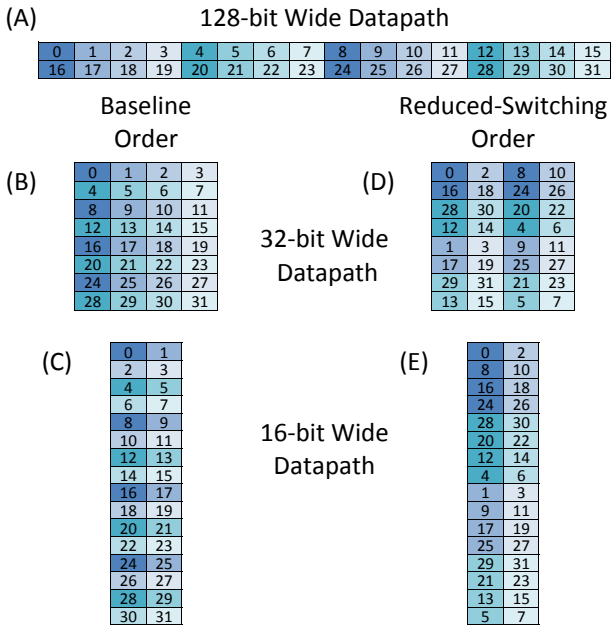


Figure 8. Data Ordering. Data from adjacent 32-bit or 64-bit words tends to be highly correlated, requiring reordering on narrower datapaths to avoid increased switching activity.

developed the mapping optimized for transfers across a 16-bit bus (Figure 8e). This ordering of the data is statically applied within the memory controller to every DRAM atom when it is being stored in DRAM and requires no additional hardware or meta-data storage. During a store operation to DRAM, the data to be sent is loaded into the transmit buffer on the memory-controller in the order specified by Figure 8(e), and then sent to DRAM. On a read, the data is returned from the DRAM in the toggle-efficient order and reordered to form the original ordering. The reordering scheme is built into the wiring the connects the transmit and receive buffers with the buffers in the memory controller.

As earlier work (CLR [19]) has noted, dynamically reordering the data within a cache-line or DRAM burst can significantly reduce switching energy on the memory interface. However, such techniques require evaluating dozens of potential data permutations on each cache-line write, and several meta-data bits to store the selected permutation. We evaluated dynamic schemes which supported up to four alternative data orderings (*e.g.,* favoring 2, 4, 8, or 16-byte interleaving) and which could have the permutation encoded within two meta-data bits we could appropriate from the ECC bits. Our simpler static scheme achieved 98% of the benefit and it is much easier to deploy within a 32-channel, high-bandwidth GPU.

VI. DRAM Controller Architecture

Each subchannel is controlled by an independent memory controller, and requests are appropriately directed into the queues of the subchannel controllers based on their address. Each subchannel controller performs request reordering and picks a request to schedule every cycle if allowed by timing constraints (similar to the baseline memory-controller [20]). However, since the subchannels in a channel share the address/command bus, only one subchannel can issue a request in a cycle. HBM has separate row and column command buses, so in a cycle one row and one column command can be issued, potentially by different subchannels. A command arbiter picks a command to issue from among the scheduled commands in the different subchannels. When command coalescing is enabled, the arbiter can merge several ready commands from different subchannels into one command if they are to the same row (activate) and bank or the same column and bank (read/write).

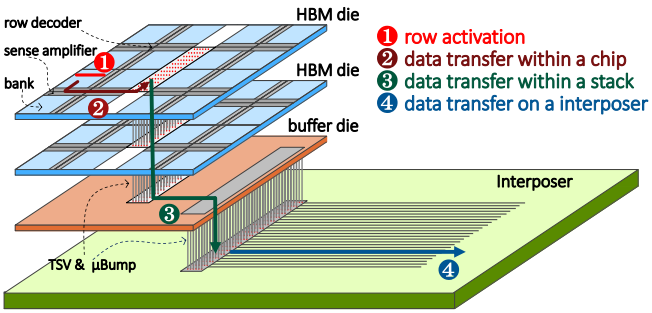


Figure 9. HBM model for energy estimation.

Overhead: We modeled the area overhead of splitting a memory controller into semi-independent subchannel controllers by using the latch, flip-flop, and logic cells from the NaNGate 45nm Open Cell library scaled to a 16nm process [21]. Implementing 8 subchannels for a 16-bank DRAM channel requires 0.07mm^2 additional controller area in 16nm (assuming typical 80% post-layout local cell-area utilization). This additional area consists of the subchannel/bank state and tracking logic (21%), arbitration and coalescing logic among subchannel-requests (55%), per-subchannel queue tracking (15%), and SERDES flops (9%). Much of the area of a GPU memory-controller comes from the transaction buffers and the associative search logic that enables pending requests for the same DRAM row to be grouped together. The peak bandwidth of a memory-controller is not changed with subchannels, but the latency is increased slightly due to the extra burst latency and the additional memory controller arbitration cycle. These few cycles of additional latency have negligible effect on average latency. Thus, we did not increase the queue depths over the baseline, and each subchannel has 1/8th the queue resources of the parent channel. Overall, for a 4-stack HBM system like in the NVIDIA P100, the total area increase across all 32 memory channels would be 2.3 mm^2, only 0.4% of the total 610mm^2 GPU die [2]. The incremental power of the extra logic is insignificant as the contribution of memory-controller power to overall system power is small; conservatively assuming power increases proportionally to the overall increase in die area, the additional memory controller power would be less than 1W on this large 200+ W GPU die. DRAM energy reductions with subchannels and GPU energy savings from higher performance far outweigh these overheads.

VII. Methodology

A. Energy Model

Our energy model uses the bottom-up methodology of the Rambus power model [22], but has been adapted to accurately model stacked memories in a modern process. We model the physical layout of the HBM die to derive detailed area estimates of the internal structures including the cell array, datapath components, address decoders, and the peripheral logic. We also model the 3D-stacked structure in its entirety including the TSVs, the logic base layer, and the silicon interposer as shown in Figure 9 using the HBM floorplan from [23] and die shots of modern DRAMs. For this model, we scale the DRAM process technology parameters presented in [22] from 55nm to 28nm, use TSV capacitance values reported in [24], and use the maximum input capacitance for a given I/O frequency (1GHz) from the HBM JEDEC specification [25] to model the energy of the I/O channel on the silicon interposer.

We estimate the row energy by estimating the capacitances switched on an activate per bitline and the voltage swing during the sensing, restoration and precharge phases. This energy is proportional to the number of bitlines on a row and is estimated to be 112 fJ/bit or 1.8nJ for a 2KB row. Previous work [13], [16] and CACTI-3DD [26] report even higher values for row-energy (5-6 nJ per 2KB).

The datapath energy from the row-buffer to the GPU's pins can be broken down into the following components: *i)* the energy for the column access on a HBM die, which includes the energy to decode the column address and drive the column-select lines (`CSL`s), data transfer from the row buffer over the `LDL`s and `MDL`s to the global sense amps (`GSA`s), and the data transfer in periphery to the I/O buffer, *ii)* the energy for moving data within the stack from the HBM die over the TSVs and then over base layer its I/O drivers, and *iii)* the data transfer energy on the silicon interposer. The first two items constitute the column energy and the third is the I/O energy. The energy consumption on the data path depends on the length of the wires and the rate at which these elements are toggled, *i.e.,* the actual bit values transferred in successive cycles. Since the `MDL`s and `LDL`s are precharged to a mid-value voltage before every transfer, their energy consumption is toggle independent, unlike the rest of the datapath. Considering the capacitance values on the datapath, we find that the toggle-independent column energy is 1.48 pJ/b, while the rest of the datapath consumes 2.85pJ/b at 50% toggle rate (2.31 pJ/b for column, 0.54 pJ/b for I/O) and 5.7pJ/b at 100%.

We use DBI-AC [25], [27], as used in HBM, to reduce the effect of toggling, and we evaluate the benefits of our reordering scheme on top of this optimization. We also account for the minor overheads for segment-select line charging (negligible), latch power of the address decoupling registers (tens of micro-Watts), and the static energy dissipated in keeping multiple rows open (0.5mW) in subchannel energy.

B. Simulation Details

We simulate a modern GPU-system based on the NVIDIA P100 chip [2] (configuration in Table I) using a detailed, in-house GPU and memory simulator. The memory controller model is optimized to harvest maximum bandwidth and thus employs optimized address mapping, deep request buffers, aggressive request reordering, and batched write-drains to minimize write-to-read turnarounds (similar to the baseline in [20]). The caches use a 32B sector size for higher performance and energy-efficiency [28].

Table I
System Configuration.

#SMs, Warps/SM, Threads/Warp	60, 64, 32
L2 cache (size, assoc., block, sector)	4MB, 16-way, 128B, 32B
DRAM (capacity, bandwidth)	16GB, 1TB/s (4stacks)

Table II
HBM Stack Configuration.

Capacity, #I/Os, Bandwidth	4GB, 1024, 2Gbps/pin
#Channels, #Banks, Row-size	8, 16/channel, 2KB
HBM Timing Parameters (in *ns* unless specified)	tRC=47, tRCD=14, tRP=14 tCL=14, tWL=2cyc, tRAS=33 tRRDl=6, tRRDs=4, tFAW=16 tRTP=3.5, tBURST=1cyc tCCDl=2, tCCDs=1, tWTRl=8 tWTRs=3, tRTPl=4, tRTPs=3

We evaluate memory intensive regions of 25 CUDA applications from the Rodinia [29] and Lonestar [30] suites, Exascale workloads [1], [31]–[35] (*CoMD, HPGMG, lulesh, MCB, MiniAMR, Nekbone*), as well as two well-known memory-bound applications with disparate access patterns, STREAM [36] and GUPS [37], to show the effect of our proposals on the spectrum of applications executed by a GPU. We also investigate 74 graphics workloads including desktop and mobile games, rendering engines, and professional graphics. To estimate the column energy of *STREAM, GUPS* and *MCB* we used the average toggle-rate across the applications since we do not have the actual data value traces used by these applications.

VIII. Results

A. Energy Improvement

Figure 10 shows the total DRAM access energy broken down into the row, column and I/O components for three different configurations — `Baseline`, eight subchannels without burst reordering (`SC-8_No_Reordering`), and eight subchannels *with* reordering to reduce toggle energy (`SC-8`) from left to right on the graph. By effectively reducing the row activation granularity to 1/8th of the `Baseline`, `SC-8` (and `SC-8_No_Reordering`) achieves an average 74% reduction across the applications. The largest benefits are obtained by applications that have low locality, either due to irregular access patterns (*bfs, bh, dmr, sp, sssp, MCB, GUPS*) or due to interference between memory accesses from different threads (*heartwall, kmeans, nw, MiniAMR*).

However, the toggle-rate increases with the default narrow subchannel design (`SC-8_No_Reordering`) leading to significant increase in average column (1.12×) and I/O energies (2×) across the evaluated benchmarks, with several benchmarks (*srad_v1, mst, sssp, MiniAMR, lulesh*) losing the benefits of row energy reduction. However, our toggle-aware burst reordering mechanism mitigates this issue making the column and I/O energies closer to the baseline (1.02× and 1.001× respectively). Consequently, the total DRAM energy consumption with `SC-8` is 35% lower than the `Baseline`.

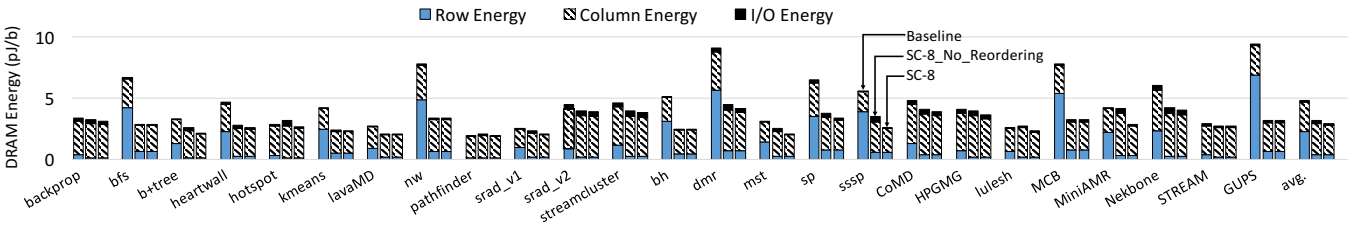


Figure 10. DRAM access energy per bit (lower is better).

Graphics applications, which generally have higher row locality than the compute applications, consume 10% lower energy per access with `SC-8` compared to `Baseline`. Regardless of row locality, all applications now incur roughly similar energy costs for accessing a bit from DRAM. By precisely activating only the necessary part of a row, `SC-8` decouples the total memory system power from row locality.

B. Performance

Figure 11 compares the performance of subchannels without parallelism (`SC-8_No_Parallelism`), with parallelism but no command coalescing (`SC-8_No_Coalescing`), and subchannels with both performance optimizations (`SC-8`) against `Baseline`.

For applications that utilize only a small fraction of the total DRAM bandwidth (*b+tree, heartwall, hotspot, lavamd, srad_v1, bh, dmr, mst,* and *sp*), there is little difference in performance between the baseline and the subchannel configurations. For the memory-intensive benchmarks (*backprop, bfs, kmeans, nw, pathfinder, srad_v2, streamcluster, sssp, CoMD, HPGMG, lulesh, MCB, MiniAMR, Nekbone, STREAM and GUPS*) the effect of our proposals depends on the memory access patterns. However, regardless of individual characteristics, all memory intensive benchmarks see significant drop in performance with `SC-8_No_Parallelism` (average 74% degradation). This result is expected, because reducing the row size in an area efficient manner (*i.e.,* not increasing the bandwidth of each mat) reduces the bandwidth of the bank to 1/8th of the baseline.

Enabling parallel operation across subchannels (`SC-8_No_Coalescing`) can recover this performance loss, and even lead to improvement over `Baseline` depending on the application behavior. Applications that had high row-buffer locality in the baseline, and were thus able to use the bandwidth adequately (*pathfinder, srad_v2, streamcluster, sssp, CoMD, HPGMG and Nekbone*), have their bandwidth requirements met by `SC-8_No_Coalescing`. Similar to the baseline, the entire bank datapath is now engaged in effective data transfer, although unlike the baseline, each subchannel is servicing a different request. On the other hand, applications which had low row-buffer locality but high bandwidth demands, were bottlenecked by bank conflicts in the baseline. Such applications (*bfs, kmeans, nw, sp, MiniAMR, GUPS*) are boosted by two beneficial features of the subchannel architecture. First, parallel subchannels offer additional bank-level parallelism that makes it possible to overlap activations of different rows in the same bank in different subchannels (and subarrays). Second, smaller rows reduce row-activation current draw that then allows a higher rate of activates. `SC-8_No_Coalescing` thus outperforms the baseline significantly in several bandwidth intensive applications: *bfs* (19.9%), *kmeans* (13.6%), *nw* (41%), *sp* (26.3%) *MiniAMR* (24.3%), *GUPS* (112%). A secondary benefit of `SC-8_No_Coalescing` is that it reduces the effect of write-drains on read performance by allowing reads and writes to proceed in parallel in different subchannels which benefits even high-locality, but bandwidth-bound, write-intensive applications like *lulesh* (6%). Overall, `SC-8_No_Coalescing` improves the `Baseline` performance by 8.1% across the evaluated benchmarks.

However, even with parallel subchannels, the command bus that served one data channel is now shared by 8 subchannels. This bus becomes a bottleneck when multiple activates must be sent to activate parts of the same row in different subchannels (which required one activate in `Baseline`). Consequently, the performance of our most bandwidth-intensive application, *STREAM*, drops by 2.7% with `SC-8_No_Coalescing`. Activate coalescing is able to eliminate this bottleneck by issuing the activates to the same row in different subchannels as a single command, eliminating the performance drop in *STREAM* (the `SC-8` bar). While activate coalescing is a preventive measure against activate/precharge command bandwidth inflation in subchannels, read/write coalescing is an optimization that can reduce the read/write command bandwidth over the baseline by merging several reads (or writes) to different subchannels in a bank into a single command if they have the same column address. With only 8 distinct columns per subchannel (256B row), the memory controller can often find such coalescing opportunities. Consequently, the `SC-8` configuration appreciably boosts the performance of bandwidth-intensive benchmarks (*bfs* (66%), *kmeans* (35%), *streamcluster* (9.1%), *sp* (39%),*lulesh* (10%), *MiniAMR* (35%), *GUPS* (152%)) leading to a 13% average improvement across all workloads over the baseline. Graphics applications see no significant change in performance with subchannels owing to their regular access patterns.

IX. Impact on ECC

ECC in the array: To support ECC in non-DIMM DRAMs, extra storage is provided in the DRAM row to store the ECC bits (*e.g.,* ×9 RLDRAM or ECC-supporting ×144 HBM). This approach creates slightly longer non-power-of-2 row sizes. To support a subchannel architecture, each mat needs to be wider, *e.g.,* 32 576×512 mats per subarray, each supplying 9 bits. The ECC bits for a burst can be fetched from the mats in a subchannel.

ECC on the I/Os: In the baseline HBM design, an 8-bit

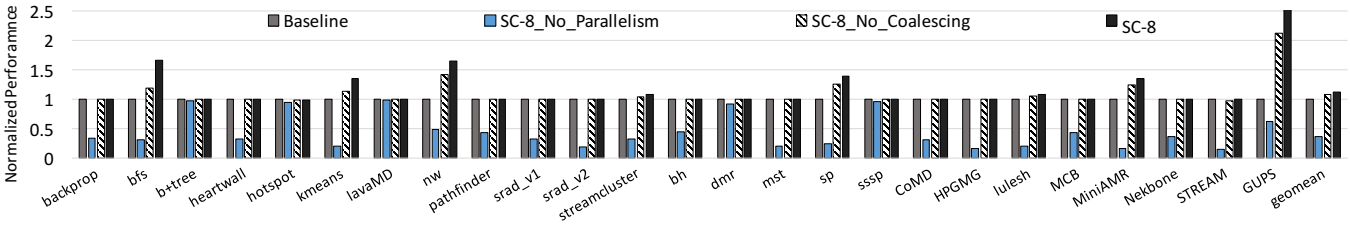


Figure 11. Performance normalized to the baseline (higher is better).

Table III
SC−8 COMPARED TO PREVIOUS WORK.

Technique	Avg. Energy Savings	Avg. Speedup	Area Overhead
128-banks	36.8%	18%	34.5%
SC−8	35%	13%	2.6%
Half-DRAM	14%	2.1%	3%
SALP	11%	7.8%	$< 1\%$

ECC word provides SECDED protection for 64 bits of data, with each 32B atom having four such 64-bit regions. Each 128-bit baseline HBM data channel has a 16-bit wide ECC lane to transfer these bits. When the number of subchannels increases, the number of signals used to transmit a single DRAM atom is reduced. In the limit of 16 subchannels, the I/O interface is 8 bits wide, with an additional 9th ECC bit. All 32 ECC bits are still transmitted with a 32-byte DRAM atom, but a sustained error on a given I/O signal will now potentially affect up to 32 bits within a single DRAM atom, and can produce errors that the four SECDED ECC words cannot protect against. To solve this issue, we propose using a robust CRC-based error-detection scheme along with a retry mechanism in the event that an error is detected (similar to DDR4 and GDDR5 systems [38]). The CRC can be chosen such that it reliably detects many types of sustained faults on the I/O signals. DRAMs with on-die ECC can rely on this scheme to protect the I/Os, with the ECC protecting the array.

X. Related Work

In this section we present a quantitative comparison of SC−8 against relevant related work (Table III).

Banks with smaller rows: Dividing existing banks into numerous smaller banks, each with 1/8th the row size, and the same bandwidth as a baseline bank is obviously the best choice for energy efficiency and high performance. However, this requires 8× more LDLs and MDLs, increasing the height and width of the mat respectively due to limited wiring tracks. It also requires additional area for 8× GSAs and the new address decoders, which makes the area overhead a prohibitive 34.5%. Through the SC−8 architecture, we can harvest most of these benefits with a much lower area overhead of 2.6%. Several previous partial row-activation techniques proposed fetching a DRAM atom from a subset of the mats on a subarray and using an activation decoder to pick the subset without creating new banks to avoid replicating the address-decoders. However, some of these techniques (*e.g.,* FGRA [11] and SBA [10]) ignored the practical layout of the mat datapath and did not account for the high area overhead that would be required for their implementation. Microbanks [39] acknowledges the need to increase the datapath width, but reports little area overhead as it underestimates the cost of the resultant taller LDL-stripe (8-way microbanking requires 64 differential pairs of LDLs, increasing the LDL-stripe height by 8×), and makes the optimistic assumption that the microbank-select signals can be routed in fine-pitch polysilcon. In essence, these techniques assumed the advantages of narrower banks but did not approriately estimate the area overheads.

Half-DRAM: To activate half of a row, Half-DRAM splits each LWL into two halves, and activates the left-half in odd mats and the right half in the even mats, using all mat datapaths for transferring a DRAM atom, thus keeping the baseline bandwidth unchanged. Clearly, Half-DRAM is a single step design that cannot be scaled to smaller activation granularities. Also, Figure 5a shows that in existing DRAMs LWDs are staggered on either side of a mat with alternate LWLs being driven from alternate sides as LWD pitch is 2X the LWL pitch. Thus there is no space to add two LWDs per LWL, one on each side, as required by Half-DRAM. Doing so will require doubling the number of LWD stripes leading to more area overhead than SC−8, but with lower energy and performance benefits.

More Row-Buffers: The row-conflict rate and hence the row energy can be reduced by keeping multiple rows active in a bank. Unlike proposals to build separate row-buffer caches on the base layer of a DRAM stack [40], [41] or near the I/Os [42], [43] which incur bandwidth and area overheads, Subarray-Level-Parallelism (SALP) [15] allows access to the already existing row buffer of each subarray in a bank. However, we found that speculatively keeping a few extra rows open per bank (15 in SALP) to reduce the number of row conflicts is less energy efficient than reducing the energy cost of each row conflict (as done by SC−8) as the number of threads accessing a bank is very high in GPUs. Also, since SALP keeps the row size and thus the activate current unchanged, only four activates can be performed in a tFAW period, restricting the performance benefit of SALP for irregular applications which are activate-rate limited. However, SALP is very unintrusive to layout and has very low area overhead.

Other work: The D-BANK architecture [9] is related to our proposals because it activates a subset of mats connected to a MWL to simplify the wiring in the periphery. However, the total number of mats that are activated are the same as the baseline, and thus there is no energy benefit or bandwidth implication. Subranked memory systems [44]–[48] use a subset of DRAM chips on a DIMM to fetch a single 64-byte cache line and thus reduce DRAM energy. These

techniques share the same philosophy as the subchannel design, but are not directly applicable to HBM or single-die solutions. In fact, our data reordering proposal can improve I/O and column energies in subranked memory systems. Orthogonal to memory architecture improvements are efforts in software and data placement to increase DRAM row-buffer hit rates [49]–[51]. The baseline GPU memory controller already aggressively reorders requests by using deep buffers to harvest most of the row-locality [20], [52], and is thus unlikely to benefit from these techniques. Some of these mechanisms also expend precious DRAM bandwidth for data migration, which renders them cumbersome and inefficient to implement in the presence of many thousands of threads in GPUs.

XI. Conclusion

Throughput oriented processors, such as GPUs, have a history of using specialized DRAMs optimized for bandwidth. As stacked, on-package DRAM technologies enable increasing bandwidth, the intrinsic DRAM array access energy is becoming significant. This row energy is particularly problematic in workloads that exhibit poor row locality, requiring frequent bank activates and precharges. The growing emergence of these low-locality workloads, coupled with a high degree of DRAM bank contention due to increasing parallelism in the processor, further motivate the need for a DRAM architecture having high internal parallelism and a small effective row size.

The proposed subchannel architecture creates smaller effective rows via master-wordline segmentation, while simultaneously providing additional internal parallelism by forming a number of semi-independent subchannels. This architecture is born out of practical considerations of DRAM layout, and is very area efficient, increasing DRAM area by 2.6%. Using 8× smaller rows and a well-crafted data layout pattern, we achieve a 35% energy improvement over HBM. The additional internal parallelism in this subchannel architecture, along with the command coalescing optimizations we describe, provide a 13% performance improvement. While evaluated in the context of GPUs, we believe that this subchannel design is a promising approach for the next generation of throughput-optimized DRAMs targeting a range of throughput processing architectures.

XII. Acknowledgments

This research was developed, in part, with funding from the United States Department of Energy and, in part, with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] 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, "Scaling the power wall: A path to exascale," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, November 2014.

[2] NVIDIA, "Inside Pascal: NVIDIA's Newest Computing Platform," 2016, https://devblogs.nvidia.com/parallelforall/inside-pascal/.

[3] A. Sodani, "Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor," in *HotChips 27*, 2015.

[4] AMD, "High Bandwidth Memory," http://www.amd.com/en-us/innovations/software-technologies/hbm.

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

[6] 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, "A $6f^2$ Buried Wordline DRAM Cell for 40nm and Beyond," in *Proceedings of the International Electron Devices Meeting (IEDM)*, December 2008, pp. 1–4.

[7] D. James, "Recent Advances in DRAM Manufacturing," in *Proceedings of the SEMI Advanced Semiconductor Manufacturing Conference*, July 2010, pp. 264–269.

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

[9] N. Sakashita, Y. Nitta, K. Shimomura, F. Okuda, H. Shimano, S. Yamakawa, M. Tsukude, K. Arimoto, S. Baba, S. Komori, K. Kyuma, A. Yasuoka, and H. Abe, "A 1.6-GB/s Data-Rate 1-Gb Synchronous DRAM with Hierarchical Square-Shaped Memory Block and Distributed Bank Architecture," in *IEEE Journal of Solid State Circuits*, vol. 31, no. 11, Nov. 1996, pp. 1645–1655.

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

[11] E. Cooper-Balis and B. Jacob, "Fine-Grained Activation for Power Reduction in DRAM," *IEEE Micro*, vol. 30, no. 3, pp. 34–47, May/June 2010.

[12] J. T. Pawlowski, "Multi-bank Memory Accesses Using Posted Writes," 2005, United States Patent 6,938,142.

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

[14] Hynix, "Hynix GDDR5 SGRAM Part H5GQ1H24AFR Datasheet Revision 1.0," 2009.

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

[16] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "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)*, June 2014, pp. 349–360.

[17] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, "Page Placement Strategies for GPUs within Heterogeneous Memory Systems," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, March 2015, pp. 607–618.

[18] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[19] B. Childers and T. Nakra, "Reordering Memory Bus Transactions for Reduced Power Consumption," in *Proceedings of the*

Workshop on Languages, Compilers, and Tools for Embedded Systems, June 2000, pp. 148–161.

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

[21] NanGate, "NanGate 45nm Open Cell Library," 2008, http://www.nangate.com/page_id=2325.

[22] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2010, pp. 363–374.

[23] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and H. S, "A 1.2V 8Gb 8-channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods Using 29nm Process and TSV," in *Proceedings of the International Solid State Circuits Conference (ISSCC)*, 2014, pp. 432–433.

[24] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens, "System and Circuit Level Power Modeling of Energy-Efficient 3D-Stacked Wide I/O DRAMs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2013.

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

[26] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2012, pp. 33–38.

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

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

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

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

[31] "Coral benchmarks," https://asc.llnl.gov/CORAL-benchmarks/.

[32] J. Mohd-Yusof and N. Sakharnykh, "Optimizing comd: A molecular dynamics proxy application study," in *GPU Technology Conference (GTC)*, March 2014.

[33] "Mantevo benchmark suite," https://mantevo.org/packages.

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

[35] S. Layton, N. Sakharnykh, and K. Clark, "Gpu implementation of hpgmg-fv," in *HPGMG BoF, Supercomputing*, November 2015.

[36] "STREAM - Sustainable Memory Bandwidth in High Performance Computers," http://www.cs.virginia.edu/stream/.

[37] "GUPS (Giga Updates Per Second)," http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/.

[38] JEDEC, *JESD79-4: JEDEC Standard DDR4 SDRAM*, 2012.

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

[40] G. Loh, "A Register-file Approach for Row Buffer Caches in Die-stacked DRAMs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2011, pp. 351–361.

[41] D. Woo, N. Seong, and H. Lee, "Pragmatic Integration of an SRAM Row Cache in Heterogeneous 3-D DRAM Architecture using TSV," *IEEE Transactions on VLSI Systems*, vol. 21, no. 1, pp. 1–13, December 2012.

[42] S. Rixner, "Memory Controller Optimizations for Web Servers," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2004, pp. 355–366.

[43] E. Herrero, J. Gonzalez, R. Canal, and D. Tullsen, "Thread Row Buffers: Improving Memory Performance Isolation and Throughput in Multiprogrammed Environments," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1879–1892, September 2013.

[44] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, November 2008, pp. 210–221.

[45] J. Ahn, N. Jouppi, and R. S. Schreiber, "Future Scaling of Processor-Memory Interfaces," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, November 2009.

[46] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Improving System Energy Efficiency with Memory Rank Subsetting," *ACM Transcations on Architecture and Code Optimization*, vol. 9, no. 1, pp. 4:1–4:28, March 2012.

[47] J. Ahn, J. Leverich, R. S. Schreiber, and N. Jouppi, "Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 5–8, 2008.

[48] Rambus, "Rambus Module Threading," http://www.rambus.com/module-threading/.

[49] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, "Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, March 2010, pp. 219–230.

[50] H. Park, S. Baek, J. Choi, D. Lee, and S. H. Noh, "Regularities Considered Harmful: Forcing Randomness to Memory Accesses to Reduce Row Buffer Conflicts for Multi-core, Multi-bank Systems," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, March 2013, pp. 181–192.

[51] S. P. Muralidhara, L. Subramaniam, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2011, pp. 374–385.

[52] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. Loh, and O. Mutlu, "Staged Memory Scheduling: Achieving High Performance and Scalability in Hetergenous Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 416–427.