Fine-Granularity Tile-Level Parallelism in Non-volatile Memory Architecture with Two-Dimensional Bank Subdivision

Matthew Poremba
Pennsylvania State University, AMD Research
matthew.poremba@amd.com

Tao Zhang
Pennsylvania State University, Nvidia Corporation
tao.zhang.0924@gmail.com

Yuan Xie
University of California, Santa Barbara
yuanxie@ece.ucsb.edu

ABSTRACT

Emerging memory technologies such as phase-change memory (PCM) and resistive RAMs (RRAM) have been proposed as promising candidates for future DRAM replacements. Due to the nature of how these memories operate, unique properties (such as non-destructive read and current-sensing) can be exploited to further subdivide memory and provide increasing parallelism with negligible overhead. In this work, we leverage these properties to design a fine-grained non-volatile memory (FgNVM), featuring ***two-dimensional bank subdivision for tile-level parallelism (TLP) in a NVM memory bank***, with much finer-granularity and increased parallelism than the *one-dimensional bank subdivision for subarray-level parallelism (SALP) in a DRAM memory bank*. With such new tile-level parallelism, three new memory access modes are proposed for further performance improvement and energy reduction: *Partial-Activation, Multi-Activation, and Background Writes.* Our experimental results show that the new architecture is highly effective in boosting non-volatile memory performance with significant energy reduction. To the best of our knowledge, this is the first work to study fine-granularity memory access in emerging non-volatile memory architectures.

1. INTRODUCTION

DRAM has been the *de facto* technology of main memory for decades. To improve DRAM performance, prior work proposed methods for parallel memory accesses by subdividing a DRAM bank into multiple subarrays, which *slices the bank in one dimension into multiple sub-units* that can be accessed in parallel to exploit *subarray-level parallelism* [1, 2, 3, 4]. Examples include selective bitline activation (SBA) [1], subarray-level parallelism (SALP) [2] and half-DRAM [5]. Many of these designs are very specific to DRAM design and command sequences, which limit the design space of highly parallel memories. This prevents DRAM banks from being further subdivided into two dimensions without a significant power/area overheads.

Emerging non-volatile memories (NVM) have been explored recently as an alternative main memory for replacing or augmenting DRAM. The emerging memory technologies such as phase-change memory (PCM), resistive RAM (RRAM), and spin-transfer torque RAM (STT-RAM), can potentially provide significant capacity, energy-efficiency, and performance benefits than conventional DRAM technologies at high capacity [6, 7, 8, 9].

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06. . . $15.00
DOI: http://dx.doi.org/10.1145/2897937.2898024

For emerging NVM, due to its non-volatility, there are unique properties that can be exploited to overcome this design space limitation and further subdivide memory to provide even more parallelism with negligible overheads. For example, read operations in DRAM is destructive and write-back operations must be performed for every read operation. Refresh must also occur periodically, while NVM read operations are non-destructive and there is no need for refresh.

Leveraging such unique properties in NVM, we propose a fine-grained non-volatile memory (FgNVM), which features *two-dimensional bank subdivision for tile-level parallelism (TLP) in a NVM memory bank*, with much finer-granularity and increased parallelism over the *one-dimensional bank subdivision for subarray-level parallelism (SALP) in a DRAM memory bank*. FgNVM allows only a portion of a memory row (a.k.a. segment) to be activated. Furthermore, it is able to activate multiple segments from *different* rows in parallel for high performance. Thus, FgNVM eliminates the restriction that only one row in each bank can be opened at any given time, and that *all* the data in this row must be sensed.

With such novel tile-level parallelism, three new memory access modes (*Partial-Activation, Multi-Activation, and Background Writes*) are proposed for further performance improvement and energy reduction. First, the ability to enable tile-level memory access, rather than subarray-level access, allows for lower energy consumption. Second, it allows for simultaneous accesses to distinct rows to boost performance. Third, the write latency overheads in non-volatile memories can be mitigated by allowing for reads during write to hide write latency and increase performance. Our evaluation shows that the new architecture is highly effective in boosting non-volatile memory performance with significant energy reduction. Analysis of our proposed design shows such techniques can be realized with low area overhead and minor modifications to existing design layouts. *To the best of our knowledge, this is the first work to study tiled-based memory access specifically tailored towards emerging non-volatile memories*.

2. MEMORY BANK SUBDIVISION

Hierarchical Memory Structure. Main memory architecture is designed in a hierarchical manner, consisting of multiple independent *memory channels*. Each channel contains one or more *ranks* which share the same bus but are otherwise fully independent. Similarly, each rank contains one or more *banks* which share the same bus but are independent otherwise. Each bank consists of a matrix of *rows and columns* which hold the individual bits of memory data. For state-of-the-art memories, *banks are the smallest accessi-*

ble unit of memory, after which memory requests destined for these units must be serialized. Any request which must wait for a request in a particular bank to complete before being serviced is called a *bank conflict*.

Bank Subdivision. The performance of the memory system can greatly benefit from further bank subdivision. Prior work proposed to share peripheral circuitry and adding extra logic to provide ability to access subdivisions of memory banks [2, 3, 4]. These works leverage the natural subdivision of banks into *memory tiles*. One tile of rows and columns within each bank is designed to trade-off area, performance, and energy.

SALP and SBA in DRAM. Prior work has considered bank subdivision for DRAM memory. In particular, research has been conducted on selective bitline activation (SBA) [1] and subarray-level parallelism (SALP) [2]. The implementation of SBA requires that wordlines be segmented in order to fetch data only applicable to a cache line request. However, segmenting wordlines requires that more bits be fetched from each tile to provide a full memory word of output [5]. This requires either an increase in the DRAM burst length or a larger internal bus and more so-called *Helper Flip-Flops*, which prefetch bits for the next data burst. It was shown that increasing the DRAM burst length can have a 30-84% reducing in performance on memory intensive benchmarks [5]. Similarly, increasing the internal data bus can tremendously increase area overhead [10]. SALP successfully decomposes a DRAM bank into one-dimension.

Finer-Granularity Bank Subdivision in NVM. In NVMs, none of these major constraints is necessarily applicable, and therefore design flexibility that would not be possible in DRAM are allowed in a non-volatile memory bank design. For example, sense amplification in NVM can be done using *current-mode sensing* made possible by resistive memory storage, whereby a known current value flows across a memory cell and the resulting voltage is compared against a known reference voltage. Prior work has shown that current-mode sense amplification time scales sub-linearly with bitline length [11], meaning cells can be sensed from outside of the array. This known voltage is typically part of the memory periphery meaning a single cell can be sensed independent of other memory bitlines. NVM reads are not destructive due to the non-volatility property and therefore multiplexers can be placed in the memory path before sense amplifiers. Due to the added resistance of multiplexing (which are typically designed as a tree of pass transistors), this is mostly applicable to NVM technologies with large difference in on/off state, such as PCM and RRAM [6, 7, 8, 9].

Our Contribution. We propose a novel NVM memory bank design to provide architectural benefits for a fine-grained memory system, to enable **two-dimensional bank subdivision** that allows access to small amounts of data from any tile without large area overhead. By achieving this goal, several techniques are proposed to allow memory accesses to tiles in different subarrays simultaneously [2, 3, 4, 5, 1, 12]. Additionally, we can allow different request types (e.g., reads and writes) simultaneously to tackle the problem of NVM write latency.

3. TILE-LEVEL PARALLELISM WITH TWO-DIMENSIONAL BANK SUBDIVISION

Our motivation for allowing two-dimensional subdivisions of memory to enable tile-level memory access parallelism is to improve performance and reduce energy. Performance can be improved by increasing parallel memory accesses and hiding write requests. Energy benefits can also be achieved by reducing the number of bits being sensed, which is a direct benefit of tile-based access.

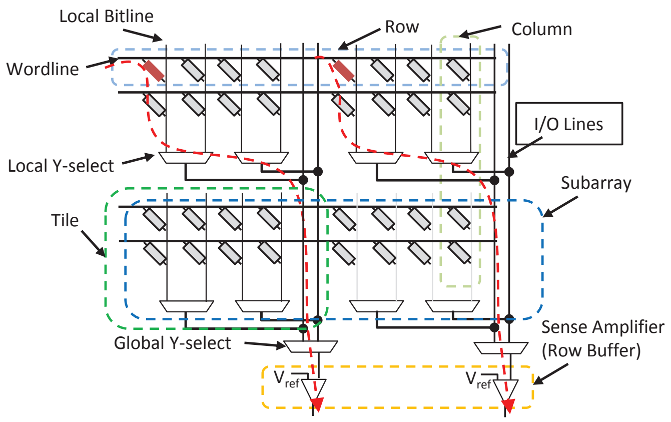


Figure 1: NVM circuit design showing small 2x4 tiles in 4x8 bank [13]. Labels point to respective memory units, red arrows show a read of 2-bits of data, and red cell are sensed.

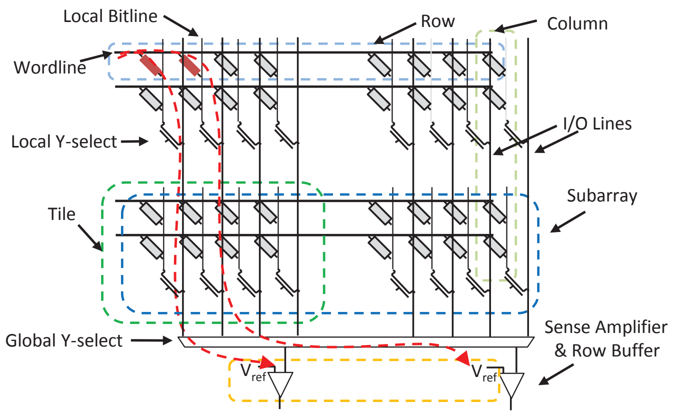


Figure 2: Our proposed FgNVM circuit design showing small 2x4 tiles in 4x8 bank. Labels point to respective memory units, red arrows show a read of 2-bits of data, and red cell are sensed.

In this section, we first describe the baseline NVM bank structure, and then present the proposed FgNVM bank design that leverages the unique non-volatility properties to allow tile-level two-dimensional subdivision.

3.1 State-of-the-art NVM Bank Structure

Our baseline non-volatile memory bank structure is based on a recent prototype [13], which is shown in Figure 1. The smallest unit of NVM memory is referred to as a *tile* and consists of 8 cells in 2 rows and 4 columns for illustration purposes. Real tiles typically range from 512 rows by 512 columns to 4K rows by 4K columns. Each tile is paired with dedicated peripheral circuitry to control which bits are selected from the tile. *Multiplexers* (MUX) down-select multiple *bitlines (BL)* to a single *I/O line*, which is the output from one tile. Recent prototypes refer to these multiplexers as *Y-select* and utilize a *Local Y-select* (LY-SEL) for tiles. These Y-select require hundreds of signals called column select logic (CSL) to individually control which columns are connected to the I/O lines. *Wordlines* (WL) are used to select the row in a tile and in this particular prototype are designed as pass transistors where the transistor gate is used to "select" the row.

Full banks are created from a large matrix of these tiles (e.g., 32 x 32) and share peripheral circuits which control the WL selection, perform further multiplexing, and sense/drive the final data while storing their values. An address decoder is used to select a single wordline for an entire bank, and is connected to global WLs which controls the pass transistor gates of each tile's WL. A *Global Y-select* (GY-SEL) is used to select the final output for the entire bank. *Sense amplifiers* (S/A) are placed at the output of GY-SEL to compare the NVM cell resistance against a reference to determine the stored data value.

Cost is the most important factor in commodity memory design, and as such there are some implications of the design described above. To outline key points, the number of I/O lines is typically kept to the minimum. In both our baseline NVM and state-of-the-art DRAM designs, I/O lines are routed adjacent to the tiles. Since there are so many CSL lines, they are routed above tiles on another metal layer to prevent increasing area overheads significantly.

3.2 Fine-granularity NVM

We start with an example to lead into improvements that can be made on our baseline design. Consider reading a row from memory in Figure 1. NVMs density causes the number of bits in a row to be very large, requiring several layers of multiplexing to select data. This means multiple NVM pages (e.g., the largest amount of data that can be read internally by the memory) may reside in the same physical row. Here A, B, C, and D represent one cache line of data and there are two NVM pages: AC and BD where AC contains A and C while BD contains B and D. When a row operation in the bank occurs, all of the data is read and sensed at once. Assuming we want to read cache line A, we need to first sense AC. Now, if C is never used, sensing the data is unnecessary. However, because the LY-SEL is designed to select all page bits, we can not control what is being read. The CSLs only allow us to select AC or BD.

Our first key observation is that we can redesign the CSLs to be able to select individual cache lines. However, this will dramatically increase the number of CSLs that are needed to control each column of tiles. This is more clear using a more realistic example. Consider a bank which contains 8k bits of data in each physical row. This corresponds to 16 cache lines of data in row and there are 16 tiles horizontally. If the row size is 4 cache lines, this means we only need 4 CSLs for each column of tiles to select an NVM page. On the other hand, selecting an individual cache line requires 16 CSLs for each column of tiles – a 4x increase.

In order to alleviate this jump in signaling requirements, we propose to group bits of the same cache line into a single tile, rather than interleaving across an entire row. This may raise concern for increased soft error rates due to high correlation of errors in nearby cells. We assume NVM resistive based storage has significantly increased resilience against radiation-induced soft errors to make such bit organization possible. By grouping bits together, we can reduce the number of CSLs based on the group size. Only 1 CSL is needed if each tile contains one cache line. In general, we need a CSL signal equal to the number of cache lines in any given tile. The problem with this approach is that the number of I/O increases in parity with the number of bits of a cache line in a tile. To overcome this, we simply swap the locations of CSLs and I/O lines.

Figure 2 shows an illustrative example of such a design where each tile contains two cache lines and therefore two CSLs are needed. The key difference between the baseline NVM design in Figure 1 and our proposed FgNVM design in Figure 2 are the placement of data, degree of multiplexing at each Y-select level, and the control of the Y-selects. In the baseline NVM in Figure 1, we obtain two bits of memory which are interleaved across all tiles in a subarray while FgNVM places data bits progressively.

In our proposed design, the LY-SEL is replaced with a single pass transistor, which is equivalent to a 1:1 MUX in Y-select designs. This Y-select can be controlled by a single CSL per tile. Note that the effective multiplexing degree is the same, however we offload most of the work to the GY-SEL. This can potential reduce chip area, as the LY-SEL must be duplicated for each tile and only one GY-SEL exists. In our design we use standard one-hot CSLs for each tile's LY-SEL, i.e., one CSL per tile per subarray. We must also introduce new register to hold CSL values for each column of tiles (see Section 5.1 for area analysis).

With the ability to control individual cache lines using CSLs, we can move one step further and control the WLs in individual tiles as well. To accomplish this we propose the same techniques used for providing subarray-level parallelism [2] and subarray-level refresh [3, 4] in DRAM. To summarize this concept, we replace the global wordline decoder with small decoders on a per-subarray basis. This does not have a large impact on area since the decoder grows linearly with input size. However, we do need to add row address registers for each subarray and logic to select registers based on the address being activated. Area overhead analysis for these constructs is confirmed in this work in Section 5.1.

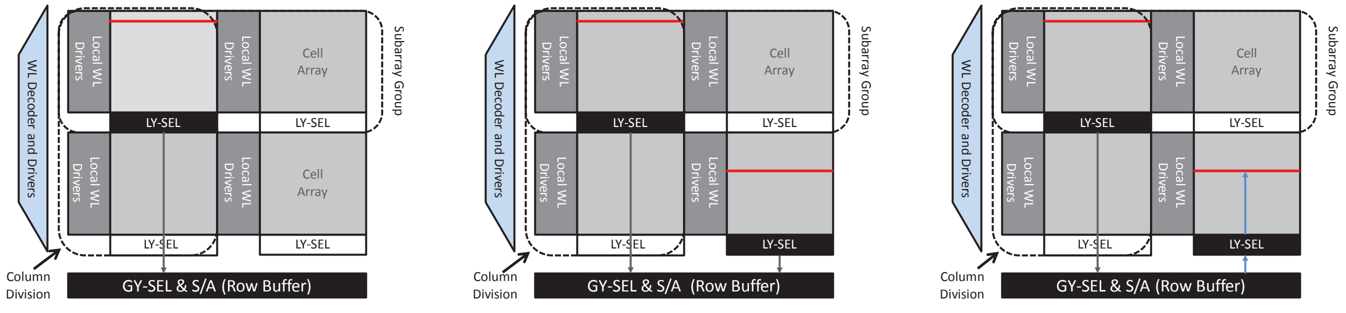
4. NEW MEMORY ACCESS TYPES

With the proposed fine-granularity two-dimensional bank subdivision, three new type of access modes are enabled for further performance improvement and energy reduction[1]:

Partial-Activation. A *Partial-Activation* is a row activation in which only a subset of bits in the row are sensed. This approach allows for reduced energy consumption by only sensing a subset of data based on requests from the memory controller. A simple example of this operation is shown in Figure 3(a). In the example, only the multiplexer control signals for the upper-left tile are enabled, as indicated by a black background on the column mux. Only data bits from the upper left array are sensed in this case, even though the wordline is selected across the entire upper row of tiles. Similarly, the bottom row has a wordline selected, but none of the multiplexer control signals are enabled. Therefore no data is sensed from the bottom tiles and they do *not* conflict with the sensing of the upper-left tile. In this case, we can reduce the sensing energy of reading this particular row by up to 50%.

Multi-Activation. *Multi-Activation* can fully make use of the free bitlines to allow multiple rows to be selected and sensed in parallel. Figure 3(b) illustrates how Multi-Activation works. Compared to Figure 3(a), the multiplexer control signals for the upper-left tile and the lower-right tile are enabled, as indicated by the column multiplexer blocks shaded black. In this way, the memory can sense data from different rows *simultaneously*. Again, the rest of tiles (upper-right and lower-left tile in the figure) are still disconnected. Therefore, no data is sensed in those disconnected tiles and the sensing operation in the active tiles is not disturbed. In the baseline NVM design, accesses to two tiles must be serialized due to lack of parallelism. By leveraging Multi-Activation, two tiles can be accessed in parallel to improve memory bandwidth and latency. The time to perform two reads is reduced by a factor of two accordingly. The Multi-Activation, however, still has two constraints. First, it cannot activate any other tile in the same CD as a tile being sensed. For instance, the upper-left and lower-left

[1]In our explanation, groups of memory tiles in the same row are denoted as subarray groups (SAGs). Groups of memory tiles in the column column are denoted as column divisions (CDs). Note that these groups are *logical* and that a subarray group or column division may contain multiple rows or columns of *physical* tiles.



(a) Partial-Activation (b) Multi-Activation (c) Backgrounded Write

Figure 3: The access schemes proposed in FgNVM. (a) Partial-Activation: Only one of the top two tiles is read from the bank, energy saved is in the other tile; (b) Multi-Activation: Data read from tiles in different rows, same back; twice the bandwidth potential; (c) Backgrounded Write: Upper-left tile is read, lower-right tile is written. Reduces read/write interference.

tiles in the figure cannot be sensed in parallel (and similarly for upper-right and lower-right tiles). Additionally, if two rows are to be sensed when the rows reside in the same tile or SAG, the data cannot be sensed in parallel as only one wordline can be selected within a tile or SAG at a time. These constraints limit the maximum subarray-level parallelism that can be achieved.

Backgrounded Writes. *Backgrounded Writes* operate similarly to Multi-Activation except sensing is replaced with write driving. An example of a backgrounded write is shown in Figure 3(c). Different from Multi-Activation, the lower-right tile is now being written instead of read. In the example, the upper-left and lower-right tiles are both selected, as indicated by the column mux shaded black. While a time-consuming write operation to the lower-right tile takes place, the upper-left tile is still available to be accessed. As shown, a read operation is ongoing while the write operation is still incomplete. Similar to Multi-Activation, it is prohibited to read from a tile in the same CD that is being written. Furthermore, the subarray group is also unavailable until the write completes. In this simple bank (2×2 tiles), only one tile is available for read. Nonetheless, for more realistically sized banks such as a 32×32 tile bank, the remaining 31×31 tiles are still available for up to 31 reads. In that case, approximately 93.8% of data in the bank is still able to be accessed during a backgrounded write operation.

5. DESIGN IMPLEMENTATION

The hardware implementation of FgNVM requires modifications to row decoders and the control of the Y-select for CDs. Typical row decoder designs use two-stage decoding. The row address must be held in a row address latch and the row decoder persistently selects the wordline. In other words, the contemporary row decoder is unable to open multiple wordlines simultaneously.

Similar to Y-select logic, an NVM design must persistently enable the local Y-select within a tile in order to be sensed. The NVM baseline enables the local Y-select corresponding to the selected columns. We assume that the column select is broadcast to all tiles in a row at the same time the wordline is selected. We need to subdivide this broadcast into individual CDs to provide fine granularity access.

Row Decoder Designs. Similar to the changes needed for subarray-level parallelism [2] and concurrent refresh [3, 4], the row decoder must be split to have dedicated row decoders for each subarray group. To support Multi-Activation, we pair each row decoder with a local latch. A multiplexer is used to select which latch to write when an activate command arrives by looking at the address bits.

Table 1: Summary of Area Overheads in FgNVM design.

Component	Avg Overhead	Max Overhead
Row Decoder	N/A	N/A
Row Latches	$2,325\mu m^2$	$9,333\mu m^2$
CSL Latches	$636.3\mu m^2$	$4242\mu m^2$
LY-SEL Lines	$0\mu m^2$	$0.1mm^2$
Total	$2,961\mu m^2$ (<0.1%)	$0.11mm^2$ (0.36%)

Y-select Designs. FgNVM also requires additional logic for CDs. For LY-SEL, we propose to add an additional enable signal per SAG in each CD. The enable signals are assumed to be routed in parallel to the I/O lines in Figure 2. These signals enable LY-SEL in only a single SAG in any given CD. The Y-select enables are one-hot signals routed to the LY-SEL of each SAG from the edge of the bank. In this way, we can still broadcast the column select signals across a CD without modification to the LY-SEL logic.

5.1 Overhead Analysis

Area overhead must be carefully considered as memory is sensitive to the cost. Area overhead may potentially arise from additional latch overhead for each subarray, modifications to the row decoder, and additional bitlines for local Y-select enables. We summarize the area overhead in Table 1. The average overhead refers to an 8×8 FgNVM, while the maximum overhead refers to a a 32×32 FgNVM.

Row Decoder For comparing row decoder designs, we assume a two-stage decoder with predecoder and second stage decoder. Based on this, we approximate number of transistors following an equation from [14]. The size of this row decoder grows by $\Omega(NlogN)$ for N rows in a bank. Doubling the number of row decoders with $N/2$ size results in a negligible change in area.

Row Latch Overhead Additional row latches are required to hold the address being used to select a row in each SAG. We measure latch overhead by implementing the additional latches in VerilogHDL and synthesizing the design with TSMC 45nm low-power technology. Area overhead is similar to that of [2, 3, 4].

CSL Latch Overhead CSL Latches are required to continuously drive our LY-SEL. The process used to calculate the row latch overhead above is repeated to obtain the value in Table 1.

Y-select Enables We estimate the area of additional enable wires by moderately sized metal3 wire pitches and spacing. In the best case, there is enough space to route Y-select enables above tiles with the global I/O lines, resulting in no extra area overhead. With a wire and spacing of 3F at 45nm. The result is a 6F wire and spacing of 270nm. Using the maximum number of 32 subarray groups and 32 column divisions results in an enable signal bus width of $246\mu m$. Based on the prototyping chip, these wires must stretch over the entire bank that has a length of $4mm$ [13], resulting in a total overhead of $0.1mm^2$ (0.15%).

6. EVALUATION

CPU Simulation. We evaluate our design using the gem5 [15] simulator. Our experimental setup models a Nehalem-like CPU using recent parameters [16].We evaluated single-threaded benchmarks using the SPEC2006 benchmark suite [17]. Simpoint [18] is used to identify a single region of interest within these benchmarks one quarter billion instructions in size. All 30 SPEC2006 benchmarks were profiled using simpoint and we selected benchmarks with 10 misses per kilo-instruction (MPKI) at the last-level cache. We use gem5 in system emulation (SE) mode and restore from a checkpoint to run a single Simpoint slice.

Table 2: Memory System Setup

Main Memory	512-byte row buffer, FRFCFS 64 write drivers, 32 queue entries 4 column divisions, 4 subarray groups

PCM Timings	tRCD=25ns, tCAS=95ns, tRAS=0ns
	tRP=0ns, tCCD=4cy, tBURST=4cy
	tCWD=7.5ns, tWP=150ns, tWR=7.5ns

Memory Simulation. We use the NVMain simulator [19] as the baseline to simulate our FgNVM design. Since the FgNVM design does not limit scheduling algorithms, we utilize a first-ready first-come first-serve (FRFCFS) memory controller scheduler [20] as well as an augmented FRFCFS algorithm. Our simulated memory is an FgNVM design based on the PCM baseline protoype in [13]. We consider this PCM design as it is one of the more mature prototypes available, includes timing parameters, and has density comparable to contemporary DRAM devices. The overall memory system design is similar to the standard DDR memory hierarchy. We assume memory is an off-chip dual in-line memory module (DIMM) divided into channels, ranks, and then banks. Each rank is assumed to have 8 PCM devices, each of which supplies a global 512-byte row buffer and provides 8 bits of output over 8 DDR cycles to provide an aggregate 64B cache line per column command. Table 2 outlines our memory parameters for what is referred to as "FgNVM" throughout our discussion. We choose a reasonable FgNVM with 4 SAGs and 4 CDs as a starting point so that we may reasonably compare against multiple independent banks. Figures and text referring to *NxM* design imply *N* SAGs and *M* CDs, respectively.

Performance Improvment. Results of our specified design are shown in Figure 4. These results show IPC improvements normalized to the NVM baseline. Since our FgNVM design mimics having multiple independent banks, we compare the results of a 4x4 FgNVM against a memory system with 128 banks per rank. Each bank is sized to be the same as any (SAG,CD) pair. Based on our configuration in Table 2, this equates to the same amount of accessible memory units. Multi-Issue bars refer to simulations where

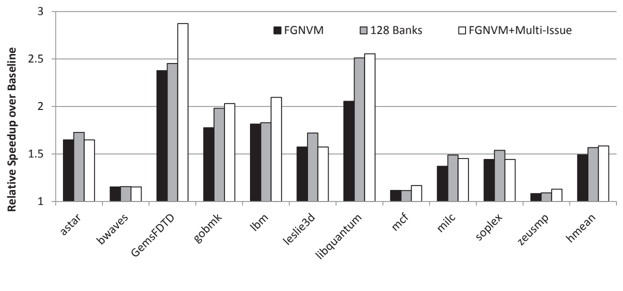


Figure 4: IPC improvement over baseline PCM design compared to FgNVM, multi-issue FgNVM, and an Ideal Scenario. All results show 8×2 FgNVM designs.

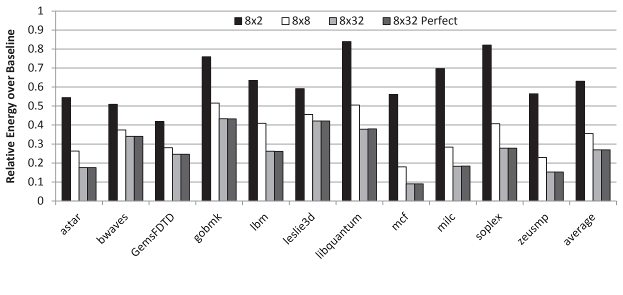


Figure 5: Energy consumption normalized to baseline NVM prototype. Shows a significant decrease in most FgNVM configurations.

multiple memory commands may be issued during the same cycle and multiple data may be returned via larger data bus.

Recall that an SAG being paired with a CD prevents all tiles in that CD from being accessed until the previous access is complete in Multi-Activation. Other columns may access that SAG assuming the same row is being accessed. Due to column conflicts (where a cache line to be served can not be issued because I/O lines are being used) and underfetch (where a portion of a row is not "prefetched" due to partial-activation), the 128 bank design preforms better than FgNVM. However the results are close to optimal for several benchmarks.

Energy Improvement. We perform an energy comparison of the FgNVM designs by sweeping over a varying number of CDs. We assume a read operation consumes 2pJ per bit, a write consumes 16pJ per bit, and background power averages to be 0.08pJ per bit of memory. Results are shown in Figure 5 and normalized to the NVM baseline, where we assume the entire row buffer is sensed during an activation. Therefore, 1KB of data must be sensed compared to 512B for 8×2, 128B for 8×8, and 32B for 8×32. For write requests, we still assume that only 64 bits of data can be written in parallel and is independent from the dimensions of the FgNVM array.

From the figure we can see that in all cases the energy of any FgNVM design is significantly lower than the baseline NVM design. Ideally, the energy consumption should decrease by a factor of two when the number of column divisions is doubled. However, due to the background energy and inability to decrease the energy of writes, the energy saving is not ideal. On average, however, the energy is reduced by 37%, 65%, and 73% in 8×2, 8×8, and 8×32 respectively. The 8×32 configuration is able to come close to ideal since this configuration reads no more than one cache line at a single time.

7. CONCLUSION

In this work, we propose to leverage unique properties of NVM to build a *fine-granularity* NVM — FgNVM, which can enable tile-level parallelism with two-dimensional bank subdivision. Our design allows for *low energy* intra-bank parallelism via *Partial-Activations, Multi-Activations, and Backgrounded Writes*. The area overhead of our design ranges from 0.1% – 0.36%, similar to sub-array-level bank subdivision in DRAM. Our techniques combined provide an average performance improvement of 56.5% over a base-line with up to 73% reduced energy. To the best of our knowledge, this is the first work to study fine-granularity memory access in emerging non-volatile memory architectures.

8. ACKNOWLEDGEMENTS

This work was supported in part by DOE award DE-SC0013553 and NSF awards 1461698, 1500848, and 1533933.

9. REFERENCES

[1] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking DRAM Design and Organization for Energy-constrained Multi-cores," in *ISCA'37*, pp. 175–186, Jun. 2010.

[2] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA'39*, pp. 368–379, Jun. 2012.

[3] T. Zhang, M. Poremba, C. Xu, G. Sun, and Y. Xie, "Cream: A concurrent-refresh-aware dram memory system," in *Proceedings of the 20th Annual IEEE International Symposium on High Performance Computer Architecture*, HPCA '14, 2014.

[4] K. K. Chang, D. Lee, Z. Chishti, C. Wilkerson, A. Alameldeen, Y. Kim, and O. Multu, "Improving dram performance by parallelizing refreshes with accesses," in *Proceedings of the 20th Annual IEEE International Symposium on High Performance Computer Architecture*, HPCA '14, 2014.

[5] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-dram: A high-bandwidth and low-power dram architecture from the rethinking of fine-grained activation," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, June 2014.

[6] Y. Xie, "Modeling, architecture, and applications for emerging memory technologies," *Design Test of Computers, IEEE*, vol. 28, no. 1, pp. 44–51, 2011.

[7] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), pp. 2–13, ACM, 2009.

[8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), pp. 24–33, ACM, 2009.

[9] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. L. Lung, and C. Lam, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 465–479, 2008.

[10] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, Dec 2010.

[11] X. Dong, C. Xu, Y. Xie, and N. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 7, pp. 994–1007, 2012.

[12] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Throughput enhancement for phase change memories," *Computers, IEEE Transactions on*, vol. 63, pp. 2080–2093, Aug 2014.

[13] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M.-G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8v 8gb pram with 40mb/s program bandwidth," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 46–48, 2012.

[14] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[15] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, *et al.*, "The gem5 Simulator," *Computer Architecture News*, vol. 39, pp. 1–7, Aug. 2011.

[16] E. Gunadi and M. Lipasti, "Crib: Consolidated rename, issue, and bypass," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 23–32, 2011.

[17] Standard Performance Evaluation Corporation, "SPEC2006 CPU." http://www.spec.org/cpu2006.

[18] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, (New York, NY, USA), pp. 45–57, ACM, 2002.

[19] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: Architectural simulator to model (non-)volatile memory systems," *Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2015.

[20] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pp. 128–138, 2000.