HAMLeT: Hardware Accelerated Memory Layout Transform within 3D-stacked DRAM

Berkin Akın, James C. Hoe, and Franz Franchetti
Electrical and Computer Engineering Department
Carnegie Mellon University, Pittsburgh, PA, USA
{bakin, jhoe, franzf}@ece.cmu.edu

***Abstract*—Memory layout transformations via data reorganization are very common operations, which occur as a part of the computation or as a performance optimization in data-intensive applications. These operations require inefficient memory access patterns and roundtrip data movement through the memory hierarchy, failing to utilize the performance and energy-efficiency potentials of the memory subsystem. This paper proposes a high-bandwidth and energy-efficient hardware accelerated memory layout transform (HAMLeT) system integrated within a 3D-stacked DRAM. HAMLeT uses a low-overhead hardware that exploits the existing infrastructure in the logic layer of 3D-stacked DRAMs, and does not require any changes to the DRAM layers, yet it can fully exploit the locality and parallelism within the stack by implementing efficient layout transform algorithms. We analyze matrix layout transform operations (such as matrix transpose, matrix blocking and 3D matrix rotation) and demonstrate that HAMLeT can achieve close to peak system utilization, offering up to an order of magnitude performance improvement compared to the CPU and GPU memory subsystems which does not employ HAMLeT.**

I. Introduction

Main memory has been a major bottleneck in achieving high performance and energy efficiency for various computing systems. This problem, also known as the *memory wall*, is exacerbated by multiple cores and on-chip accelerators sharing the main memory and demanding more memory bandwidth. 3D die stacking is an emerging technology that addresses the memory wall problem by coupling multiple layers of DRAM with the processing elements via high-bandwidth, low-latency and very dense vertical interconnects, i.e. TSVs (through silicon via). However, in practice, the offered high performance and energy efficiency potentials is only achievable via the efficient use of the main memory.

Exploiting the data locality and the abundant parallelism provided by multiple banks, ranks (and layers) is the key for efficiently utilizing the DRAM based main memories. However, several data-intensive applications fail to utilize the available locality and parallelism due to the inefficient memory access patterns and the disorganized data placement in the DRAM. This leads to excessive DRAM row buffer misses and uneven distribution of the requests to the banks, ranks or layers which yield very low bandwidth utilization and incur significant energy overhead. Existing solutions such as memory access scheduling [1] or compiler optimizations [2], [3] provide limited improvements. Memory layout transformation via data reorganization in the memory aims the inefficient memory access pattern and the disorganized data placement issues at their origin. Yet, transforming the memory layout suffers from the high latency of the roundtrip data movement from the memory hierarchy, to the processor and back to the main memory. Also the memory layout transformation is associated with a bookkeeping cost of updating the address mapping tables [4].

In this paper, we present HAMLeT, a hardware accelerated memory layout transform framework that efficiently reorganizes the data within the memory by exploiting the 3D-stacked DRAM technology. HAMLeT uses high-bandwidth, low-latency and dense TSVs, and the customized logic layer underneath the DRAM to reorganize the data in the memory by avoiding the latency and the energy overhead of the roundtrip data movement through the memory hierarchy and the processor. HAMLeT proposes a lightweight and low-power hardware in the logic layer to address the thermal issues. It introduces very simple modifications to the existing 3D-stacked DRAM systems such as the hybrid memory cube (HMC) [5]–it mostly uses the already existing infrastructure in the logic layer and does not require any changes to the DRAM layers. By using the SRAM based scratchpad memory blocks and the existing interconnection in the logic layer, it implements efficient algorithms to perform otherwise costly data reorganization schemes. For the reorganization schemes that we consider, HAMLeT handles the address remapping in the hardware, transparent to the processor. Our work makes the following contributions:

- To our knowledge, HAMLeT is the first work that proposes a high-performance and energy-efficient memory layout transformation accelerator integrated within a 3D-stacked DRAM.
- HAMLeT uses a lightweight hardware implementation that exploits existing infrastructure in the logic layer, and does not require any changes to the DRAM layers, yet it can fully exploit the locality and parallelism within the 3D-stacked DRAM via efficient layout transform algorithms.
- We evaluate the performance and energy/power consumption of HAMLeT for the data reorganizations such as: Matrix transpose, matrix blocking, and data cube rotation.
- For the analyzed reorganization schemes, HAMLeT handles the address remapping in the hardware, transparent to the software stack, and does not incur any bookkeeping overhead of the page table updates.
- We compare HAMLeT with CPU and GPU memory subsystems, which do not employ hardware accelerated data reorganization, and demonstrate up to 14x performance and 4x bandwidth utilization difference.

This work was sponsored by DARPA under agreement HR0011-13-2-0007. The content, views and conclusions presented in this document do not necessarily reflect the position or the policy of DARPA.

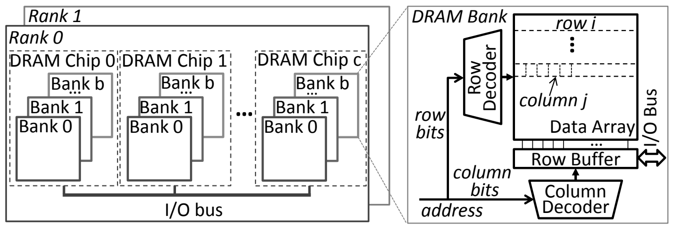

Fig. 1. Organization of an off-chip DRAM module.

II. DRAM Operation and 3D-stacking

A. DRAM Operation

As shown in Figure 1, modern DRAM modules are divided hierarchically into ranks, chips, banks, rows and columns. Set of DRAM chips which are accessed in parallel to form the whole DRAM word constitute a rank. Each DRAM chip has multiple internal banks that share the I/O pins. A bank within a DRAM chip has a *row buffer* which is a fast buffer holding the lastly accessed row (page) in the bank. If the accessed bank and page pair are already active, i.e. the referenced page is already in the row buffer, then a *row buffer hit* occurs reducing the access latency considerably. On the other hand, when a different page in the active bank is accessed, a *row buffer miss* occurs. In this case, the DRAM array is *precharged* and the newly referenced page is *activated* in the row buffer, increasing the access latency and energy consumption. Exploiting the spatial locality in the row buffer is the key to achieve high bandwidth utilization and energy efficiency.

In addition to the row buffer locality, bank/rank level parallelism has a significant impact on the DRAM bandwidth and energy utilization. Given that different banks can operate independently, one can overlap the latencies of the row precharge and activate operations with data transfer on different banks/ranks. However, frequently precharging and activating pages in different banks increases the power and total energy consumption.

B. 3D-stacked DRAM

3D-stacked DRAM is an emerging technology where multiple DRAM dies and a logic layer are stacked on top and connected by TSVs (through silicon via) [5], [6], [7], [8]. TSVs allow low latency and high bandwidth communication within the stack, eliminating I/O pin count concerns. Fine-grain rank-level stacking, which allows individual memory banks to be stacked in 3D, enables fully utilizing the internal TSV bandwidth [9], [5]. As shown in Figure 2(a), fine-grain rank-level stacked 3D-DRAM consists of multiple DRAM layers where each layer has multiple DRAM banks, and each bank has its own TSV bus. Vertically stacked banks share a TSV bus and form a vertical rank (or sometimes referred as vault [5]). Each vault can operate independently.

The internal operation and the structure of the 3D-stacked DRAM banks are very similar to the regular DRAMs (see Figure 1) except some of the peripheral circuitry is moved down to the logic layer which enables achieving much better timings [9]. As shown in Figure 2(b), the logic layer also includes a memory controller, a crossbar switch, vault and link controllers. The memory controller schedules the DRAM commands and aims to maximize the DRAM utilization while obeying the timing constraints. The crossbar switch routes the data between different vaults and I/O links. Finally, the vault and the link controllers simply transfer data to the DRAM layers and to the off-chip I/O pins respectively. Typically, these native control units do not fully occupy the logic layer and leave a real estate for a custom logic implementation [10]. However, the thermal issues limit the complexity of the custom logic.

III. HAMLeT Architecture

We propose the HAMLeT architecture shown in Figure 2(c) for efficient reorganization of the data layout in the memory. The processor offloads the memory layout transform to the HAMLeT, and the data reorganization is handled in the memory by avoiding the latency and energy overhead of the roundtrip data movement through the memory hierarchy. Overall, memory layout transform operations can be summarized as: reading chunks of data to the logic layer, local reorganization of the data in the logic layer and finally writing chunks of data back to the DRAM layers. This operation is repeated to transform the memory layout of big datasets.

The HAMLeT architecture shown in Figure 2(c) features three main components to handle the overall layout transform operation: (*i*) local fast buffers (*ii*) interconnection between these buffers and (*iii*) a control unit. In fact, existing 3D-stacked DRAM systems, such as HMC [5], [10], already provide the interconnection infrastructure in the logic layer. HAMLeT makes use of the crossbar switch in the logic layer–it only introduces SRAM blocks as the fast local buffer and a control unit that orchestrates the data movement between the DRAM layers and the SRAM blocks.

HAMLeT features dual-bank SRAM blocks per vault (i.e. per independent TSV bus) which are used for double-buffering to ensure continuous flow of data. It transfers data chunks between the DRAM layers and SRAM blocks in parallel by using independent vaults and also time-multiplex each TSV bus shared by multiple layers to maximize the parallelism. It always transfers consecutive row buffer size chunks of data to/from DRAM layers to minimize the number of row buffer misses. Hence, the SRAM blocks are sized according to the configuration of the 3D-stacked DRAM such that they can hold multiple DRAM rows. SRAM blocks do not incur any penalties depending on the memory access patterns, which enables efficient local data reorganization. Furthermore, the crossbar switch provides the substrate for exchanging data chunks between the SRAM blocks. Finally, the control unit accepts the layout transform commands with a few parameters (e.g. size, source, destination etc.)—it does not require a full-fledged processor. Interestingly, such an architecture which consists of SRAM blocks connected via a switch network can execute wide variety of data reorganization schemes [11].

IV. Layout Transform Operations

Several data-intensive applications such as linear algebra computations (GEMM, GEMV), spectral methods (single/multi-dimensional FFTs), signal processing (SAR imaging) require data shuffle operations as part of their computation (e.g. matrix transpose, 3D-matrix rotation [12], [13]) or can benefit from memory layout transformations (e.g. row-major to column-major layout transform, tiled data layout [12],

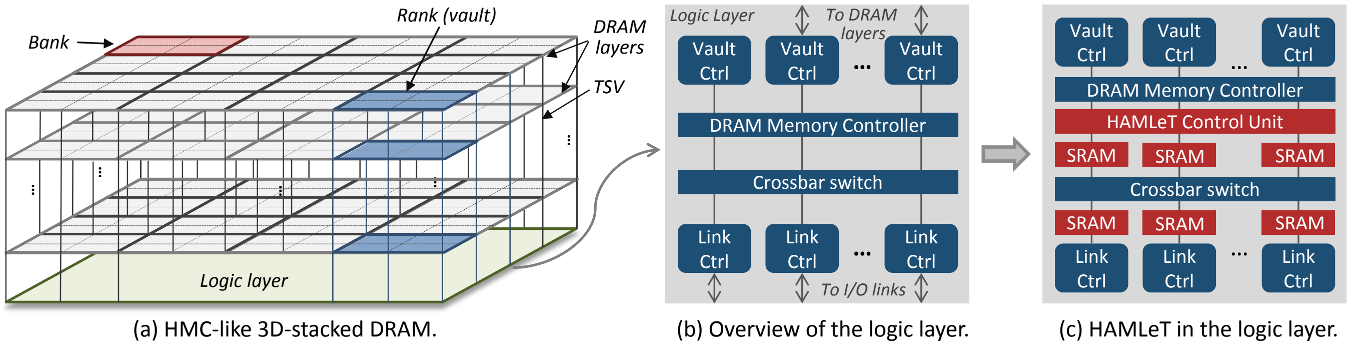

Fig. 2. 3D-stacked DRAM architectural overview and the HAMLeT architecture in the logic layer.

[14], [15]). Such operations are costly to perform and generally avoided since: (i) inefficient memory access patterns fail to utilize the locality and parallelism in the memory hierarchy which yields low bandwidth and high energy utilization, and (ii) roundtrip data movement through the memory hierarchy suffers from a high latency and energy overhead. In this work we analyze HAMLeT for the following layout transform operations: matrix transpose, matrix blocking and 3D-matrix rotation.

A. *Matrix Transpose*

Matrix transposition can be a required data shuffle operation as a part of the computation (e.g. 2D-FFT) or it can be used to transform the memory layout of a matrix for higher performance. Matrix transposition is generally avoided due to its cost, yet HAMLeT uses an efficient matrix transposition algorithm which exploits the locality and parallelism within the 3D-stacked DRAM.

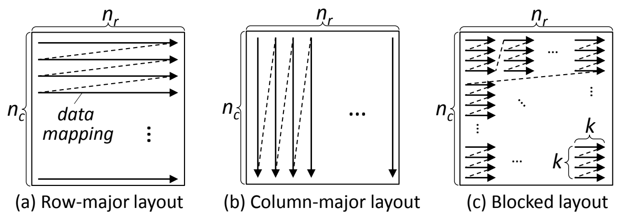

Fig. 3. Logical 2D array view of the matrices ($n_r \times n_c$). Arrows represent the linear data placement in the memory.

Matrix transpose simply takes a row-major matrix (Figure 3(a)) and transforms it into a column-major matrix (Figure 3(b)), or vice versa. Conventionally this operation is performed by reading rows from the source and writing them as columns to the destination. There exists optimized algorithms that exploit blocking and parallelism [16], [14]. HAMLeT uses a blocked transpose algorithm as follows; (i) assuming the DRAM row buffer holds r element pages where $r < n_r$, it transfers an $r \times r$ element tile by reading r DRAM pages to the logic layer and writes them into the SRAM blocks. (ii) The $r \times r$ element tile is locally transposed by exchanging data chunks among SRAM blocks via the crossbar switch. (iii) It can write the locally transformed data back to DRAM layers as whole DRAM pages (i.e. r elements) since it holds r columns. This scheme requires the SRAM buffers to hold r^2 elements. This operation is repeated until the whole dataset is transposed. HAMLeT efficiently utilizes the DRAM via three main components: (i) It always transfers complete DRAM pages which minimizes the row buffer misses and maximizes the data locality utilization. (ii) Writing/reading data chunks to/from SRAM blocks in parallel fully exploits the parallelism of independent per-vault TSV buses (inter-vault parallelism). (iii) It maximizes the bandwidth utilization of a TSV bus via fine-grain round-robin scheduling of the accesses to the layers within a vault (intra-vault parallelism).

B. *Matrix Blocking*

Often times blocked data layouts provide a middle ground in between the canonical row/column major layouts and offer significant improvements [17], [12], [15]. Transforming a row/column major layout to a blocked layout or re-sizing the tiles can be a crucial feature to utilize these improvements.

There exists various forms of blocked layouts, such as z-morton or various space filling curves, but Figure 3(c) demonstrates a case where elements within the blocks and the blocks themselves are ordered in row major. We select the block size to match the r-element DRAM page via $k \times k$ element tiles where $k = \sqrt{r}$. This ensures the transfer of the whole pages to minimize the number of DRAM row buffer misses. Further it minimizes the local SRAM requirement. In performing the matrix blocking, HAMLeT reads entire r-element pages. It needs to buffer at least k pages so that it can form $k \times k$ tiles locally. It writes the locally blocked data back to the DRAM layers as $k \times k$ tiles where each tile is mapped to a page. Transferring whole DRAM pages, i.e. tiles, throughout the transformation minimizes the row buffer misses. Similarly, multiple SRAM blocks utilize the inter/intra vault parallelism. This scheme requires SRAM buffers of size $k \times r = r^{3/2}$ elements.

C. *Cube Rotation*

Some scientific applications such as 3D-FFTs, molecular dynamics simulation, or tensor contraction have multidimensional datasets which require data shuffle and layout transform operations [12], [13].

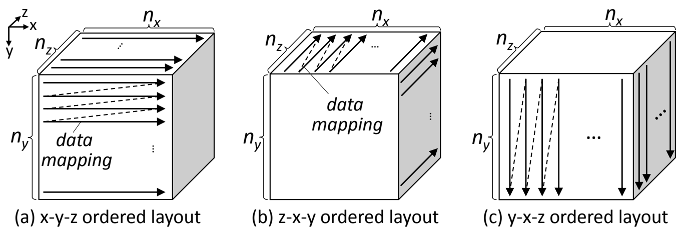

Fig. 4. Logical 3D view of the 3D matrices ($n_x \times n_y \times n_z$).

A multidimensional dataset, such as a 3D matrix (cube), can be mapped into the linear memory address space in various ways. x-y-z, z-x-y, and y-x-z ordered layouts are demonstrated in Figure 4. Assuming the x-y-z ordered layout in Figure 4(a), transferring elements in x-direction leads to sequential accesses, whereas y and z direction result in inefficient strided access patterns. For a given layout, we will call the dimension that generates sequential accesses as the *fast dimension*, and the other dimensions as *slow dimensions*.

In performing the cube rotation, for example from x-y-z order to z-x-y order, HAMLeT reads entire r-element DRAM pages in the fast dimension (x) to efficiently utilize the row buffer locality. It buffers r pages in the z direction, i.e. the fast dimension of the target layout, so that in the write-back stage it can write entire DRAM pages in the fast dimension of the target layout. Similar to the other transform examples, it uses the fast SRAM blocks and the crossbar switch for local reorganization and distribution of data chunks within the stack exploiting the parallelism and locality. It needs to buffer $r \times r$ elements from the data plane constructed by the fast dimensions of the source and the target layouts (x-z plane in this case), which requires r^2-element size SRAM buffers.

V. Address Remapping

A memory layout transformation changes the mapping between the logical addresses and the actual physical locations of the data in the memory. Hence the address mapping needs to be updated according to the performed memory layout transformation. Conventionally, for a general purpose system that employs virtual memory, changing the address mapping requires TLB invalidations and updating the page table entries. Changing the OS page size or keeping a small mapping table in the memory controller are proposed to alleviate this problem [4], [18]. However these solutions offer limited scalability and cannot completely eliminate the overheads.

HAMLeT employs a pure hardware based address remapping mechanism to overcome this problem such that no software/OS involvement is necessary after the memory layout transform to update the address mapping. We make the observation that a memory layout transform can be represented by a data permutation operation. For the certain family of the data permutations the required address remapping can be expressed via simple bit shuffle operations. HAMLeT uses a reconfigurable bit shuffle unit which can handle the address remappings for the memory layout transform examples presented in this work. Accesses are forwarded to their new locations via the address remapping unit in the hardware transparent to the OS or software stack. Address remapping examples for the matrix transpose and matrix blocking are shown in Figure 5. Although the address remapping through bit shuffle operations can capture the matrix layout transforms, it may not be applicable to an arbitrary layout transform. Formal analysis and the generalization of the address remapping via bit shuffles is beyond the scope of this paper and we defer it as a future work.

VI. Evaluation

In this section we present the performance and power/energy evaluation of the HAMLeT for the selected memory layout transformations. We also provide a performance and bandwidth utilization comparison of the 3D-stacked DRAM based HAMLeT with CPU and GPU based memory layout transformations.

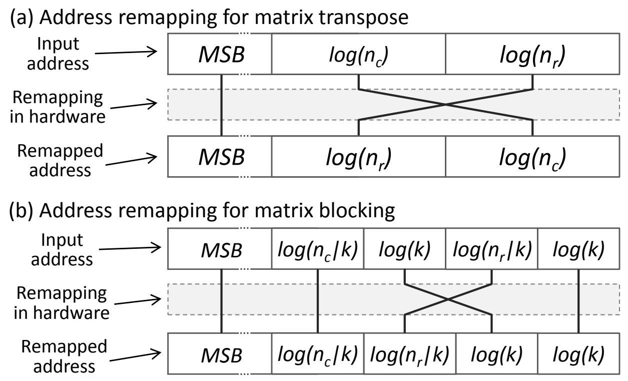

Fig. 5. Address remapping in hardware for the memory layout transforms matrix transpose and matrix blocking ($n_r \times n_c$ element matrix, $k \times k$ blocks).

TABLE I. 3D-stacked DRAM configurations. (32nm tech.)

Name	Configuration $N_{\text{stack}}/N_{\text{bank}}/N_{\text{TSV}}$/Page	tCL-tRCD-tRP-tRAS-tTSV (ns)	Peak BW (GB/s)
L4-B8-T512	4 / 8 / 512 / 8 kb	12.2-7.89-16.8-22.9-0.68	337.2
L6-B16-T512	6 / 16 / 512 / 8 kb	10.7-7.18-6.10-10.8-1.22	732.0
L4-B16-T512	4 / 16 / 512 / 8 kb	10.7-7.22-8.37-13.8-0.68	809.8
L4-B16-T256	4 / 16 / 256 / 8 kb	10.7-7.22-8.37-13.8-0.68	466.7
L4-B8-T256	4 / 8 / 256 / 8 kb	12.2-7.89-16.8-22.9-0.68	201.2

A. Layout Transform Using HAMLeT

We use a modified version of the CACTI-3DD [9], that includes more detailed parameters from the published work [7], [10]. We generate the low level timing and energy parameters, then input them to our custom power/performance simulator to get the estimations for the memory access pattern of a particular layout transform. In our analysis, we target a variety of 3D-stacked DRAM configurations which are shown in Table I. These configurations are within the range of typical 3D-stacked DRAM systems where the theoretical peak bandwidth within the stack is ranging from 200 GB/s to 800 GB/s [8], [5] and the area efficiency is larger than 40% [7].

Figure 6 presents the performance, bandwidth utilization, power/energy consumption results for the HAMLeT in performing matrix transpose, matrix blocking and cube rotation examples. HAMLeT efficiently exploits the 3D-stacked DRAM to reorganize the memory layout such that (i) it utilizes the DRAM row buffer by writing/reading entire pages to/from DRAM layers and minimizes the row buffer misses, (ii) it fully exploits the parallelism among the independent TSV buses by writing/reading data chunks to/from local per-vault SRAM blocks in parallel, (iii) it maximizes the bandwidth utilization of a TSV bus via round-robin scheduling of the accesses to the layers within a vault. Consequently, for all of the layout transform operations, HAMLeT can achieve close to the peak bandwidth utilization, high performance and low power/energy consumption for a variety of memory configurations. Next, to put the numbers in Figure 6 into perspective, we compare the results to CPU and GPU based layout transformations.

B. Comparison Against CPU and GPU

For the comparison against CPU and GPU, we use optimized reference implementations on each platform. We use a

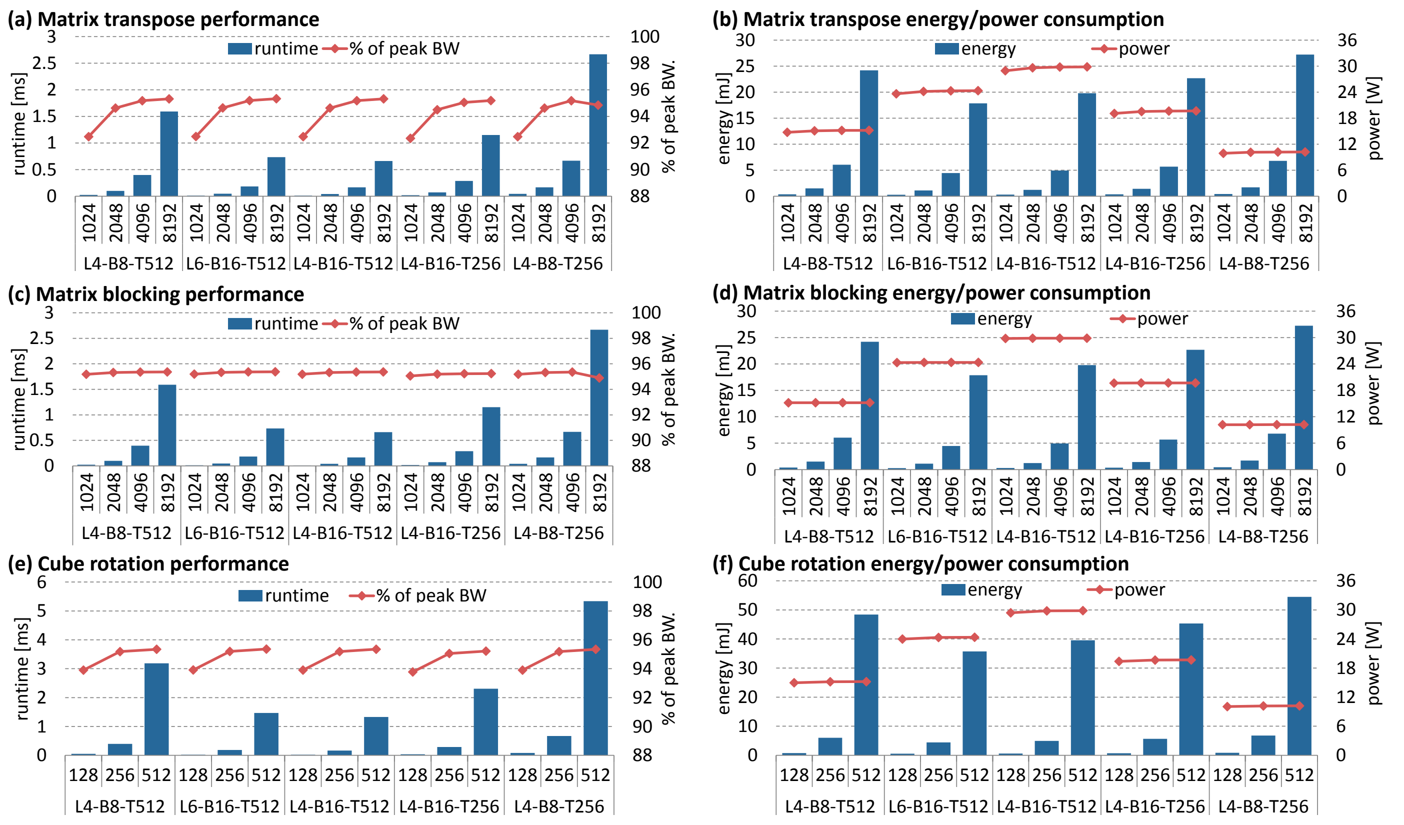

Fig. 6. Runtime, bandwidth utilization, energy and power consumption of layout transform operations for various matrix sizes and various DRAM configurations.

multi-threaded implementation from [19], which outperforms Intel MKL, running on an Intel Xeon E3-1230 (Sandy Bridge) processor. Xeon E3-1230 features a dual channel DDR3-1333 DRAM with peak bandwidth of 21 GB/s. For the GPU implementation, we use an optimized CUDA implementation from Nvidia [20] running on an Nvidia GTX 670 GPU. For the HAMLeT implementation we use a low-end configuration (L4-B8-T256) which has a peak bandwidth of 201 GB/s which is very close to the GTX-670's 192 GB/s peak bandwidth. The comparison is presented in Figure 7.

HAMLeT demonstrates up to an order of magnitude higher performance in terms of runtime as shown in Figure 7. Both CPU and GPU suffer from high latency overhead of the roundtrip movement of the data from DRAM to on-chip memory hierarchy and back to the DRAM. Further, inefficient strided access patterns of the layout transforms degrade the DRAM performance and make the overall operation memory bounded. HAMLeT implements efficient layout transform algorithms on its parallel architecture which enables much higher, very close to peak, bandwidth utilization compared to CPU and GPU (see Figure 7). Here we emphasize that HAMLeT is not proposed as an alternative to CPU and GPU, instead, it can be integrated into their memory subsystem as a hardware accelerator for data reorganization.

C. Hardware Area and Power Analysis

We also present power and area cost analysis for the hardware implementation of HAMLeT in 32nm technology node. We synthesize the HDL implementation targeting a commercial 32nm standard cell library using Synopsys Design Compiler following a standard ASIC synthesis flow.

TABLE II. AREA AND POWER CONSUMPTION OF THE SRAM BLOCKS.

DRAM conf. Page (kb) / Vaults	Required SRAM conf. Banks / Size (kB) / Width	Area (mm^2)	Power (mW)
8 / 8	8 / 64 / 32	1.28	232.1
8 / 16	16 / 32 / 32	1.28	253.8
16 / 8	8 / 256 / 32	4.68	792.2
16 / 16	16 / 128 / 32	4.62	835.8

In addition to the standard ASIC synthesis flow, for non-HDL SRAM memory blocks, we use CACTI 6.5 [21]. An example control unit implementation that handles the matrix blocking consumes 1970 μm^2 chip area and 1036 μW power at 1 GHz clock frequency. In fact, even the fully generic HAMLeT control unit (see Figure 2(c)) is a very simple state machine, hence the overall area and power is dominated by the SRAM blocks. Table II reports the area and the power consumption of the SRAM blocks at full utilization for various memory configurations. We observe that the typical area and power overhead of the HAMLeT is very low such that it consumes 2–7% of the logic layer area and it increases the power consumption by 3–5.5% when considered as a part of an HMC-like system [10]. Hence it can be safely integrated into the logic layer without thermal concerns.

VII. RELATED WORK

Memory access scheduling techniques for achieving high performance [1] are limited to a fixed window of requests and do not address the source of the problem. Furthermore, compiler based optimizations such as [2], [3] are limited by data dependencies, and they cannot capture the dynamic runtime information.

Transforming the memory layout at runtime has the latency

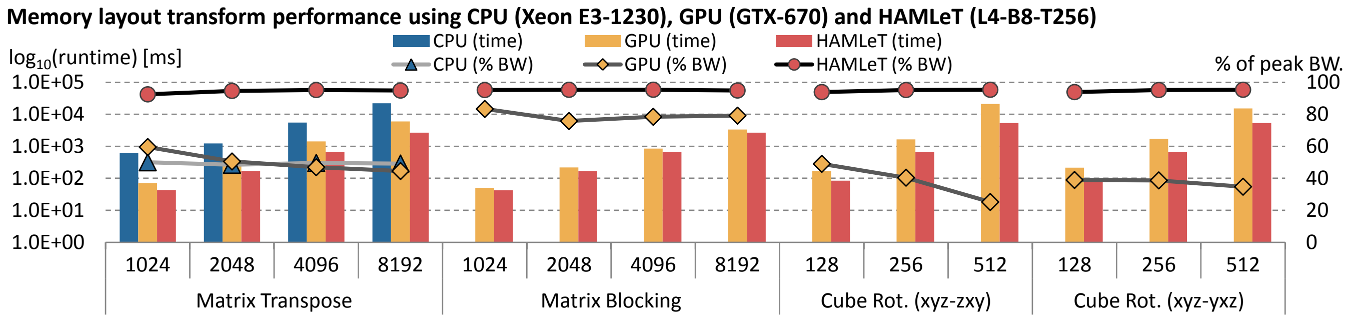

Fig. 7. Memory layout transform performance on CPU, GPU and 3D-DRAM based HAMLeT for various matrix sizes.

and energy overhead of the roundtrip data movement [22], [4], [18]. Further, it is associated with a bookkeeping cost for updating the page tables [4], [18]. Specialized copy engines [23], changing the structure of the DRAM for bulk data movement [24], or using GPU's high memory bandwidth to overlap the layout transforms with slow PCI data transfer [25] are some of the proposed techniques to mitigate the mentioned problems. Whereas, HAMLeT is a high-bandwidth and energy-efficient data reorganization accelerator integrated within 3D-stacked DRAM that implements efficient layout transform algorithms in the memory.

There are also several related work demonstrating 3D-stacked DRAM and processing elements integrated together for application acceleration [8], [26], [27], and for general purpose computing [6], [10], [5], [7]. Integrating a high-performance general purpose processor underneath the DRAM raises some thermal issues, on the other hand, energy-efficient accelerators are specific to a certain application.

VIII. Conclusion

In this paper, we presented a high-bandwidth and energy-efficient hardware accelerated memory layout transform (HAMLeT) framework integrated within 3D-stacked DRAMs. HAMLeT implements efficient memory layout transform algorithms on its parallel architecture which utilizes the locality and parallelism (bank/vault/layer) in a 3D-stacked system. Further, the hardware implementation of HAMLeT in the logic layer is very simple and does not require any changes to the DRAM layers. Our experiments demonstrate that it can transform multidimensional matrix layouts while achieving close to peak bandwidth utilization, offering an order of magnitude higher performance than commodity CPU and GPUs.

References

[1] S. Rixner *et al.*, "Memory access scheduling," in *Proc. of the 27th Int. Symposium on Computer Architecture (ISCA)*, 2000, pp. 128–138.

[2] K. S. McKinley *et al.*, "Improving data locality with loop transformations," *ACM Trns. Prog. Lang. Syst.*, vol. 18, no. 4, pp. 424–453, 1996.

[3] M. E. Wolf *et al.*, "A data locality optimizing algorithm," in *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, ser. PLDI '91, 1991, pp. 30–44.

[4] K. Sudan *et al.*, "Micro-pages: Increasing dram efficiency with locality-aware data placement," in *Proc. of Arch. Sup. for Prog. Lang. and OS*, ser. ASPLOS XV, 2010, pp. 219–230.

[5] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hotchips*, 2011.

[6] G. H. Loh, "3D-stacked memory architectures for multi-core processors," in *Proc. of the 35th Annual International Symposium on Computer Architecture, (ISCA)*, 2008, pp. 453–464.

[7] C. Weis *et al.*, "Exploration and optimization of 3-D integrated dram subsystems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 597–610, April 2013.

[8] Q. Zhu *et al.*, "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*, Oct 2013, pp. 1–7.

[9] K. Chen *et al.*, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Design, Automation Test in Europe (DATE)*, 2012, pp. 33–38.

[10] J. Jeddeloh *et al.*, "Hybrid memory cube new dram architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*, June 2012, pp. 87–88.

[11] M. Püschel *et al.*, "Permuting streaming data using RAMs," *Journal of the ACM*, vol. 56, no. 2, pp. 10:1–10:34, 2009.

[12] B. Akin *et al.*, "FFTs with near-optimal memory access through block data layouts," in *Proc. IEEE Intl. Conf. Acoustics Speech and Signal Processing (ICASSP)*, 2014.

[13] "Gromacs," http://www.gromacs.org, 2008.

[14] J. O. Eklundh, "A fast computer method for matrix transposing," *IEEE Transactions on Computers*, vol. C-21, no. 7, pp. 801–803, July 1972.

[15] N. Park *et al.*, "Tiling, block data layout, and memory hierarchy performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 640–654, July 2003.

[16] J. Choi *et al.*, "Parallel matrix transpose algorithms on distributed memory concurrent computers," in *Proceedings of the Scalable Parallel Libraries Conference*, Oct 1993, pp. 245–252.

[17] B. Akin *et al.*, "Memory bandwidth efficient two-dimensional fast Fourier transform algorithm and implementation for large problem sizes," in *Proc. of the IEEE Symp. on FCCM*, 2012, pp. 188–191.

[18] X. Dong *et al.*, "Simple but effective heterogeneous main memory with on-chip memory controller support," in *Intl. Conf. for High Perf. Comp., Networking, Storage and Analysis (SC)*, Nov 2010, pp. 1–11.

[19] A. Vladimirov, "Multithreaded transposition of square matrices with common code for Intel Xeon processors and Intel Xeon Phi coprocessors," http://research.colfaxinternational.com, Aug 2013.

[20] G. Ruetsch *et al.*, "Optimizing matrix transpose in cuda," Nvidia Tech. Report, Jan 2009.

[21] "CACTI 6.5, HP labs," http://www.hpl.hp.com/research/cacti/.

[22] N. Park *et al.*, "Dynamic data layouts for cache-conscious implementation of a class of signal transforms," *IEEE Transactions on Signal Processing*, vol. 52, no. 7, pp. 2120–2134, July 2004.

[23] L. Zhao *et al.*, "Hardware support for bulk data movement in server platforms," in *Proc. of IEEE Intl. Conf. on Computer Design, (ICCD)*, Oct 2005, pp. 53–60.

[24] V. Seshadri *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proc. of the IEEE/ACM Intl. Symp. on Microarchitecture*, ser. MICRO-46, 2013, pp. 185–197.

[25] S. Che *et al.*, "Dymaxion: Optimizing memory access patterns for heterogeneous systems," in *Proc. of Intl. Conf. for High Perf. Comp., Networking, Storage and Analysis (SC)*, 2011, pp. 13:1–13:11.

[26] S. Pugsley *et al.*, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on mapreduce workloads," in *Proc. of IEEE Intl. Symp. on Perf. Analysis of Sys. and Soft. (ISPASS)*, 2014.

[27] B. Akin *et al.*, "Understanding the design space of DRAM-optimized FFT hardware accelerators," in *Proc. of IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors (ASAP)*, 2014.