Exploring Specialized Near-Memory Processing for Data Intensive Operations

Salessawi Ferede Yitbarek*, Tao Yang†, Reetuparna Das*, Todd Austin*
*University of Michigan, Ann Arbor
{salessaf,reetudas,austin}@umich.edu
†University of California, San Diego
tay016@eng.ucsd.edu

***Abstract*—Emerging 3D stacked memory systems provide significantly more bandwidth than current DDR modules. However, general purpose processors do not take full advantage of these resources offered by the memory modules. Taking advantage of the increased bandwidth requires the use of specialized processing units. In this paper, we evaluate the benefits of placing hardware accelerators at the bottom layer of a 3D stacked memory system compared to accelerators that are placed external to the memory stack. Our evaluation of the design using cycle-accurate simulation and RTL synthesis shows that, for important data intensive kernels, near-memory accelerators inside a single 3D memory package provide 3x-13x speedup over a Quad-core Xeon processor. Most of the benefits are from the application of accelerators, as the near-memory configurations provide marginal benefits compared to the same number of accelerators placed on a die external to the memory package. This comparable performance for external accelerators is due to the high bandwidth afforded by the high-speed off-chip links. On the other hand, near-memory accelerators consume 7%-39% less energy than the external accelerators.**

I. Introduction

Given the importance of workloads that process large volumes of data, numerous efforts are being channeled into designing systems that can process these big-data applications at higher speeds while consuming less power. For these applications, cost of data movement dominates system performance and energy efficiency. Processing In Memory (PIM) is a promising approach towards reducing the cost of data movement by integrating memory and accelerators in the same chip package.

The advent of commercially feasible 3D chip stacking technology, such as the Hybrid Memory Cube (HMC) [1], has renewed interest in PIM [2]–[5] architectures. A 3D chip can consist of multiple logic and DRAM layers stacked on top of each other and connected through high bandwidth through-silicon-vias (TSVs). Unlike previously studied PIM systems [6]–[8], the logic and DRAM layers can be manufactured using different process technologies. This makes it possible to efficiently integrate high-quality logic into the memory package.

In this paper, we implement and evaluate a PIM architecture with accelerators for common data-intensive kernels: sorting, string matching, memory copy, and hash-table lookup. We study the advantages, limits, and challenges of integrating them into a layer in a 3D stacked memory package, and compare this near-memory processing approach with a conventional approach of implementing the accelerators in a processor die. We also propose a data-mapping scheme which optimizes for contiguous memory access patterns prevalent in the near-memory workloads and facilitates load-balancing across accelerators.

Our studies show that while near-memory accelerators provide significant speedup (up to 13x) and better energy efficiency (up to 159x) compared to general purpose cores, they render modest advantages compared to accelerators implemented on the processor die and external to the memory system. This is because accelerators that are integrated inside the processor die still benefit from high bandwidth access to memory. A single Hybrid Memory Cube (HMC), for example, offers a bandwidth of 160GB/s-320GB/s – more than 10x the bandwidth offered by a single DDR3 module [1]. Furthermore, since these accelerators can generate multiple outstanding memory requests, they can achieve high throughput without suffering greatly from the added latency of off-chip communication. On the other hand, near-memory placement of the accelerators saves 7-39% energy compared to an accelerator architecture that is tightly coupled to the processor.

II. Background and Motivation

A. 3D Stacked Memory Systems

Memory system designers have been striving to increase the amount of bandwidth and memory density per pin without consuming disproportionate power. In the progression from DDR to DDR4, the bandwidth of a *single* memory package increased from 2.66GB/s to 21.3GB/s while energy consumption per bit accessed decreased 6 fold.

To maintain this trend, 3D stacked memory systems have been introduced in recent years. Recently, Micron Inc. has introduced the Hybrid Memory Cube (HMC), a 3D stacked package that integrates multiple DRAM layers and a single logic layer. The logic layer consists of memory control, testing, and interfacing logic. The different layers are connected using high bandwidth through-silicon-vias (TSVs), which can provide an aggregate internal bandwidth up to 320GB/s between the logic layer and DRAM layers.

In tandem with internal bandwidth improvements, DRAM vendors have spent significant effort to improve the external interface bandwidth as well. The package communicates with the processor die (or other external units) through high bandwidth serializer-deserializer (SerDes) links and can support a bandwidth of 160-320 GB/s depending on the number of links. The memory slices are connected to the external SerDes links through a crossbar.

Unfortunately, current processors are not capable of taking full advantage of these improvements in internal and external memory bandwidth. Figure 1 shows the memory resource utilization of a simulated Quad Core Xeon CPU executing data intensive benchmarks, while connected to a 16-vault HMC DRAM with a 160 GB/s external SerDes interface. For each execution, we measure the fraction of total bandwidth utilized throughout the execution of the program. We also observed that all banks are idle more than 80% of the time on average. We consider a bank to be idle if it is not servicing a read, write, or refresh operation during a particular cycle. Clearly, the processor cores cannot take advantage of the bandwidth improvements.

B. Data Intensive Kernels

To evaluate the benefits of near memory processing, we implement accelerators for important big-data kernels described below.

978-3-9815370-6-2/DATE16/

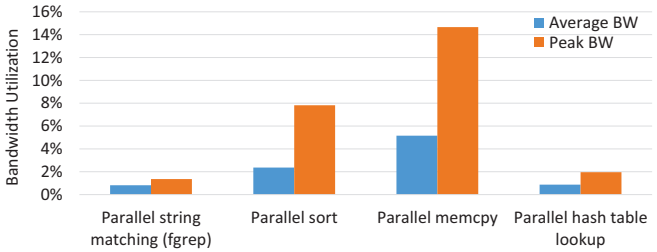

Fig. 1. **Bandwidth Utilization**. Bandwidth utilization of selected data intensive operations running on a simulated Quad Core Xeon processor. All of the operations run in four concurrent threads. Values are normalized against 160GB/s - the maximum bandwidth from a 16-vault HMC.

Sorting: Database and data analytics systems usually need to sort intermediate and final query results. Data processing pipelines such as *Hadoop* and *Spark* rely on sort-based algorithms to shuffle data among mapper and reducer processes. Sort operations have limited parallelism and significant data movement across different memory locations.

String Matching: String matching is a fundamental operation when searching through any text. Example applications include database queries, searching through unstructured text and network intrusion detection.

Memcopy: A recent study that profiled warehouse scale computers in a Google data center has reported that at least 4-5 % of data center cycles are spent on *memcpy* and *memmove* operations[9]. This study also found that approximately two-thirds of the time spent on remote-procedure calls is consumed by data movement of the payloads.

Hash Table Lookup: Hash table lookup is representative of operations with poor data locality. In-memory key-value stores, database join algorithms, and text indexing programs are a few examples of applications that need to maintain large hash tables in main memory. Furthermore, linked-list based hash-table implementations have to perform pointer chasing operations which require multiple expensive trips to memory followed by short computations.

III. An Architecture for Accelerated Near-Memory Computation

We base the design of our near-memory architecture on Micron's Hybrid Memory Cube. Figure 2 illustrates the high-level architecture of the design. The DRAM layers are composed of multiple independent channels called vaults. Each of these vaults are organized as independent vertical slices constructed from multiple DRAM layers. Each of the vaults can be accessed in parallel, and thus have independent accelerators and memory controllers associated with them.

A. Application-Specific Accelerators

When operating on data residing in their *local* vault, accelerators will have direct high-bandwidth access to the DRAM layers via the TSVs. Some of the operations we will discuss below require data to be gathered from multiple vaults. In this case, memory has to be accessed through the on-chip crossbar interconnect at a lower throughput. Whenever possible, operations are scheduled on accelerators that are directly connected to the banks containing the accessed data. The DRAM controllers associated with each vault handle requests from accelerators co-located with them as well as read and write requests that come from the processor and other accelerators through the interconnects. The accelerator controllers also accept commands from the processor cores

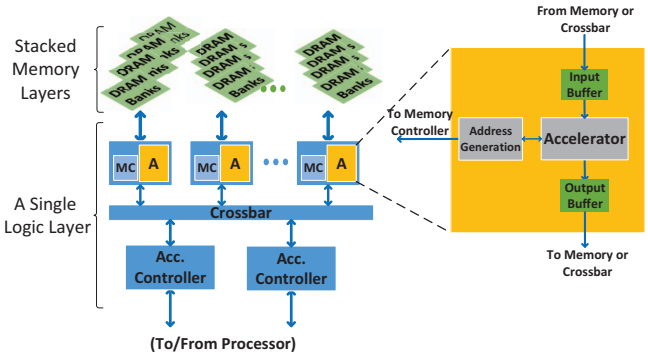

Fig. 2. **Near Memory Accelerator Organization.** Accelerators have high bandwidth access to *local* vaults stacked directly on top of them. Accelerators can also access banks in *non-local* vaults by sending data, address, and command through the crossbar connecting all vaults together. For increasing throughput, operations are scheduled on accelerators that are directly connected to the banks containing the data, whenever possible.

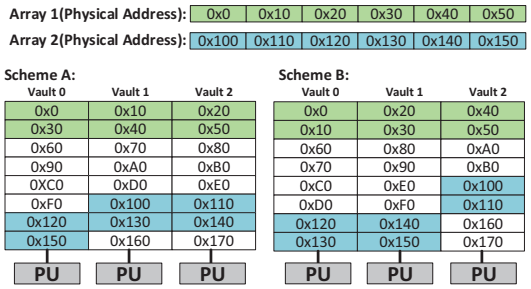

Fig. 3. **Memory Bank Mapping Schemes**: Scheme A is the default address mapping scheme under the HMC specification. Scheme B is ideal for near-memory processing but requires extra operations to lay out data.

and coordinate the operation of accelerators. The design of individual accelerators is discussed in Section IV.

B. Optimizing Memory Mapping

The default memory mapping scheme (Scheme A in Figure 3) interleaves contiguous addresses across vaults. Near-memory workloads typically process data stored in contiguous addresses. Thus, it is preferable to have contiguous addresses mapped to the same vault (Scheme B in Figure 3). This allows the accelerators to access data directly from their local vaults.

Scheme B mapping can be achieved by allocating data in the granularity of operating system pages, which ensures that contiguous addresses map to the same vault. Minor modifications to the address decoding scheme can ensure that an entire 4KB page will be stored in a single vault. In the default vault interleaved scheme (Scheme A), for a 4GB device, physical address bits 6-9 are used as the vault index. For proposed scheme (Scheme B), the physical address bits 12-15 are used as the vault index. The rest of the bits can also be re-mapped in a manner that optimizes for row hits and bank-level parallelism. This re-mapping does not require any extra hardware in the Hybrid Memory Cube as it allows user-defined memory mappings to be specified by writing to mode registers[1]. Finally, when the OS allocates free pages for near-memory processing, it has to ensure balanced mapping of pages across vaults.

One disadvantage of mapping contiguous memory blocks to the same vault is that it increases bank conflicts. In Section V we explore the impact of the increased bank conflicts on regular programs.

IV. Accelerator Designs

In this work, we evaluate accelerators for four different data-intensive operations. We chose these particular operations because they represent a non-trivial subset of important memory-intensive operations that are commonly part of big-data workloads. In this section, we describe the design of the accelerators used in this study.

A. Sorting Unit

The sorting accelerators proceed in two phases. The first phase sorts multiple small subsets of the data. The second phase progressively merges these sorted subsets into a fully sorted list.

During the first phase, multiple sorting networks operate on the data in parallel. We use Batcher's bitonic sort network [10] for this initial phase. Our area estimates based on a 45nm cell library show that a sorting network with 128 input elements will have an area of $1.72mm^2$ while a network with 64 input elements will have an area $0.88mm^2$. Since we place one sorting unit per vault, it is not feasible to have sorting networks that process large data sets in one pass. In this study, we model a sorting network with an input size of 64 elements (64-bits each) and runs at 800MHz.

After the sorting networks finish operating on the input data, the merge units sequentially load the sorted subsets from memory and progressively merge these sorted subsets into a fully sorted list. We also implemented additional logic for in-place merging, which reduced the amount of extra memory that is needed.

B. String Matching Unit

To evaluate string matching workloads, we implemented the hardware described in [11]. It is an implementation of the well-known Aho-Corasick algorithm. This algorithm is also used by fgrep - which is a part of the GNU grep utility.

Before the search process begins, the processor core compiles the state machine from the query strings and the constructed state machine representation is written to SRAM tiles within the accelerator. When the search process begins, the accelerator starts loading memory blocks into a buffer. Each of the characters in a memory block are serially processed to update the state of the string matching module. It takes 64 cycles to process a 64 byte memory block.

C. Hash Table Lookup Unit

This unit is designed to search through open-chained hash-tables, i.e, buckets store pointers to linked lists that store the key-value data. When a key lookup request is received, the bucket for the key is computed using hash units at the memory interface. The request is then sent to the unit integrated with the vault that is storing the bucket for the specific key. Open-chained hash tables have poor locality. Hence, the units integrated within each of the vaults may need to send packets to remote vaults to fulfill a key lookup request.

The major component of this unit is the linked-list traversal module. It is essentially a content addressable memory (CAM) that keeps track of the state of multiple outstanding requests. The table stores the keys being searched for, the address of the current outstanding read requests for those specific keys, and the state of the active look ups (reading pointer to next element, fetching key). When a read response comes from memory, the CAM is searched to find the key and state associated with a specific memory load.

The pointer chasing process requires virtual-to-physical address translation when traversing linked lists. All the addresses stored as part of the hash table are virtual addresses while the hardware requires the corresponding physical addresses. Introducing an extra Translation Look aside Buffer (TLB) in the memory cube and frequently updating it can have high overheads. Fortunately, TLB lookups can be avoided by mapping part of a process's contiguous virtual address space to a contiguous physical address range [12], either manually within the OS or via large virtual pages. This eliminates the need for TLB lookups.

D. MemCopy Unit

The memory copy unit is straightforward. The start address and the size of the source and destination memory areas are written to the accelerator before the copy begins. All reads and writes occur in block-size chunks. As read requests can arrive out-of-order, the accelerator checks the address of a memory read reply and computes the write offset.

V. Evaluation Methodology

We based our memory system design on the HMC specification version 1.1. The memory structure is a 3D stacked DRAM, with 16 vaults. The vaults are connected through a crossbar to four serialize-deserializer (SerDes) channels. We set the memory block size to 64 bytes, with a burst length(BL) of 4 at a memory controller clock frequency of 800MHz. With these settings, we were within an error of 1% of the HMC peak read bandwidth. However, we observed that small changes to these parameters have minimal effect on simulation results. All the accelerators were also run at a frequency of 800MHz.

For performance evaluation, we use the MARSSx86 cycle-accurate full-system many-core simulator integrated with the DRAMSim2 memory simulator. The DRAMSim2 simulator was modified to model the crossbar and links. The 16 vaults are simulated as 16 independent channels. The accelerators were modeled using custom cycle-accurate SystemC models.

Power consumption estimates for the Xeon cores were obtained using McPat 1.3. We only consider the dynamic power of the cores as the leakage component will still be present when computations are completely offloaded to accelerators. For the Atom cores, we assume an average power consumption of 3W[13]. We created an approximate power and area model for the accelerators by synthesizing the different specialized units to a 45nm standard cell library using Synopsis Design Compiler. Since the power consumed by the string matching engine is mostly dominated by the SRAM arrays, we estimate its the power consumption using CACTI 6.0.

We assume the DRAM read energy to be 3.76 pJ/bit, while, the logic layer has an additional transfer energy of 6.78 pJ/bit[13]. Furthermore, for conservative estimation of the accelerators' energy efficiency, we assume a total DRAM background power of 1.4 W based on the values in [2].

We analyze the following four systems in our experiments:

- **Xeon+HMC:** A single Quad Core Xeon processor (2.6GHz, 15MB L3, 256KB L2 and 32KB I-cache and D-cache) connected with an HMC 3D DRAM stack.
- **NM Atom:** 16 Intel Atom cores (1GHz, 2-issue, 1MB L2 cache, 32KB I-cache and D-Cache) integrated into the logic layer of HMC 3D DRAM stack.
- **External Acc:** 16 external accelerators integrated in the processor die. Each of the accelerators are connected to HMC via the SerDes links. Since there are only four links, the accelerators are connected to the SerDes links through arbiters, with four accelerators per SerDes link.
- **NM Acc:** Near Memory (NM) architecture with 16 accelerators.

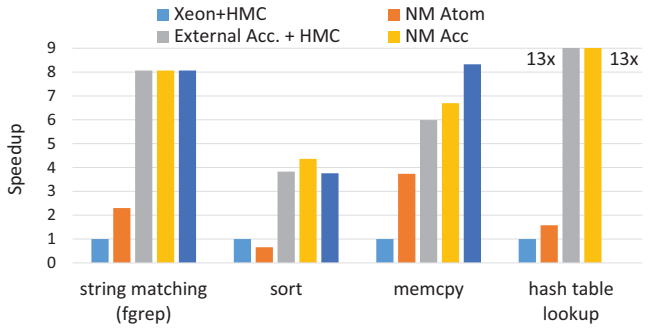

Fig. 4. **Performance Comparison**

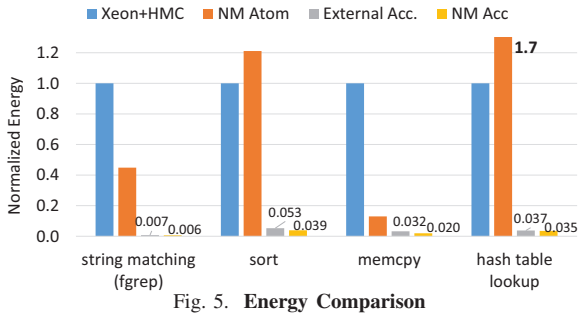

Fig. 5. **Energy Comparison**

We use representative input sizes for the four workloads studied. The sorting benchmark sorts 256 MB of 8 byte integers (i.e, 33 million keys). The software implementation of sorting utilizes a multi-threaded quick-sort followed a series of merge-sort operations. The string matching benchmark uses 256MB of input data. The software implementation of string matching is a modified version of fgrep that takes advantage of multiple cores by searching through different subsets of the input string in parallel. The memory copy benchmark copies 128 MB of data and the software implementation is multi-threaded with each thread operating on a subset of the data. The hash table lookup benchmark processes 1 million keys for a open-chain hash table with 2 million buckets.

VI. Experimental Results

Performance Comparison: The results in Figure 4 show that near-memory accelerators provide up to 13x speedup on data intensive operations compared to an external quad-core Xeon CPU, and up to 8.5x speedup over near-memory Atom cores. Clearly, the accelerators provide significant performance value, but the value of near-memory configurations is less, as these configurations have only slightly increased performance over external accelerators. This is due to the insensitivity of the applications to the modest latency increase experienced by external accelerators.

Optimized Memory Mapping: The proposed mapping scheme enables a 20% speedup over vault interleaved mapping for near-memory memcopy operation. On the other hand, the proposed mapping scheme degrades near-memory sort performance by 16%. Our measurements show that the last merge step (which accounts for 70% of the execution time) benefits from the increased bank-level parallelism provided by the vault interleaved mapping while the balanced mapping scheme helps speed up only the first 30% of the execution period.

We measured that the multi-threaded string matching and sorting kernels running on the Xeon core experience less than 1% slowdown under the modified physical address to vault/bank mapping. However, when subjecting the memory system to continuous external reads and writes by having all 4 cores perform memcpy operation, a 4% slowdown is incurred.

Energy Comparison: The near-memory accelerators are 19x-159x more energy efficient compared to the Xeon core coupled with an HMC (Figure 5). On the other hand, near-memory accelerators consume 7%-39% less energy than externally placed accelerators. So if energy is a critical constraint, near memory computing becomes a more attractive option.

VII. Conclusion

While emerging stacked memory systems provide significantly more bandwidth and bank-level parallelism compared to current DDR3 and DDR4 modules, general purpose processors have limited capabilities in leveraging these resources. In this work we show that specialized accelerators integrated into a 3D stacked memory system can provide 3x-13x speedup over a Quad Core Xeon processor. However, the near-memory accelerators provide marginal benefits over accelerators placed on an external die, since the off-chip links are capable of providing a large amount of bandwidth. Furthermore, the applications are not sensitive to the modest latency increase due to off-chip communication. On the other hand, near memory-accelerators can save 7%-39% energy compared to off-chip accelerators.

VIII. Acknowledgments

The authors would like to thank the reviewers, whose insights improved this work. This work was supported in part by C-FAR, one of the six STARnet Centers, sponsored by MARCO and DARPA.

References

[1] H. M. C. Consortium, *Hybrid Memory Cube Specification 1.0*, 2013.
[2] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," *2014 IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2014.
[3] B. Akin, F. Franchetti, and J. C. Hoe, "Data reorganization in memory using 3d-stacked dram," in *Proceedings of the 42nd Annual Int'l Symp. on Computer Architecture (ISCA)*, 2015.
[4] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," *2013 IEEE International 3D Systems Integration Conference (3DIC)*, 2013.
[5] D. Ping, N. Zhang, Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: throughput-oriented programmable processing in memory," *International Symp. on High-Performance Parallel and Distributed Computing (HPDC14)*, 2014.
[6] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. Lacoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, G. Daglikoca, and M. Rey, "The Architecture of the DIVA Processing-In-Memory Chip," *Int'l Conf. on Supercomputing (ICS)*, 2012.
[7] J. Gebis, S. Williams, and C. Kozyrakis, "VIRAM1 : A Media-Oriented Vector Processor with Embedded DRAM," *Design Automation Conference (DATE)*, 2004.
[8] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "FlexRAM: Toward an advanced intelligent memory system," in *IEEE 30th Int'l Conf. on Computer Design (ICCD)*, 2012.
[9] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42Nd Annual Int'l Symp. on Computer Architecture (ISCA)*, 2015.
[10] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of spring joint computer conference*, 1968.
[11] L. Tan, B. Brotherton, and T. Sherwood, "Bit-split string-matching engines for intrusion detection and prevention," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 3, 2006.
[12] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," Proceedings of the 40th Annual Int'l Symp. on Computer Architecture (ISCA), 2013.
[13] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *2012 Symp. on VLSI Tech. (VLSIT)*, 2012.