Improving Node-level MapReduce Performance using Processing-in-Memory Technologies

Mahzabeen Islam[1], Marko Scrbak[1], Krishna M. Kavi[1],
Mike Ignatowski[2], and Nuwan Jayasena[2]

[1] University of North Texas, USA
{mahzabeenislam,markoscrbak}@my.unt.edu,krishna.kavi@unt.edu
[2] AMD Research - Advanced Micro Devices, Inc., USA
{mike.ignatowski,nuwan.jayasena}@amd.com

Abstract. Processing-in-Memory (PIM) is the concept of moving computation as close as possible to memory. This decreases the need for the movement of data between central processor and memory system, hence improves energy efficiency from the reduced memory traffic. In this paper we present our approach on how to embed processing cores in 3D-stacked memories, and evaluate the use of such a system for Big Data analytics. We present a simple server architecture, which employs several energy efficient PIM cores in multiple 3D-DRAM units where the server acts as a node of a cluster for Big Data analyses utilizing MapReduce programming framework. Our preliminary analyses show that on a single node up to 23% energy savings on the processing units can be achieved while reducing execution time by up to 8.8%. Additional energy savings can result from simplifying the system memory buses. We believe such energy efficient systems with PIM capability will become viable in the near future because of the potential to scale the memory wall.

Keywords: Processing-in-Memory, 3D-DRAM, Big Data, MapReduce

1 Introduction

While the idea of moving processing to memory (i.e., Processing-in-Memory, PIM) is not new [13, 19, 6, 10] the advent of 3D-stacked DRAMs [2, 4, 9] which include dedicated logic dies within a DRAM package, have generated renewed interest in PIMs [19, 20, 12, 15]. Current research shows that enough free silicon area is available within the logic layer to permit the inclusion of computational units. PIM architectures are particularly beneficial for data intensive and memory bounded applications that do not necessarily benefit from the conventional cache hierarchy [26]. PIM cores can access memory using faster, high bandwidth TSVs (Through Silicon Via) [15, 20, 24] instead of conventional or specialized high bandwidth memory buses that consume significant energy for transferring data between DRAM and off-chip processing cores [4]. Because of this observation we favor PIMs over a heterogeneous multicore system where a number of small cores (or GPUs) are integrated with powerful main CPUs since they

require excessive amounts of data transferred from/to off-chip DRAM units. Nonetheless, several challenges remain. Among the key issues to investigate include the types of computing cores and how many of them to include in the logic layer to fully utilize available (3D) DRAM bandwidth while not exceeding power budgets.

In this paper, we propose a new server architecture with a number of simple in-order single-issue cores as PIM cores. We describe the roles and responsibilities of the main processor and PIM cores and our assumptions about the memory. We also propose modifications to MapReduce framework in order to optimize this unconventional architecture specifically for Big Data processing. In Big Data analysis, generally clusters of large number of commodity machines are used in conjunction with a standard MapReduce framework [5, 1]. A cluster of small number of our proposed servers and the modified MapReduce framework will be able to provide better performance with lower energy consumption than existing large commodity cluster systems.

Phoenix++ [18] is a highly optimized MapReduce framework for large-scale shared memory CMP (Chip Multiprocessor)/SMP (Symmetric Multiprocessing) systems. We find that single node performance of Phoenix++ is better than that of Hadoop. This is also true for similar shared memory MapReduce library [11]. We propose a two level MapReduce framework: inter-node and intra-node level. The inter-node level execution flow will be similar to a standard MapReduce framework (e.g., Hadoop). In this paper we focus on intra-node level MapReduce execution flow. We start with Phoenix++ and optimize it for our PIM architecture. The intra-node level reduce phase performs local (node-level) optimization before sending the results for inter-node level reduce phase i.e. global reduction. Our preliminary analyses show that a single node with proposed model can obtain up to 23% energy savings on the processing units and 8.8% reduced execution time as compared to Phoenix++ running on an SMP system for several Big Data workloads.

The rest of the paper is organized as follows. Section 2 includes the background and related work. Section 3 describes the new PIM architecture and system organization. Section 4 describes MapReduce as a use case for the PIM architecture and shows the modifications in the MapReduce workflow. In Sect. 5 we discuss experimental results. Section 6 concludes and discusses future steps.

2 Background and Related Work

3D-stacked DRAM is an emerging memory organization providing larger capacity with lower latency, higher bandwidth and lower energy than existing 2D-DRAM technologies [2, 23]. 3D-DRAM package is composed of several layers of DRAM cells stacked on top of a logic layer containing the necessary peripheral circuitry for the DRAM. There are several prototypes, including the Hybrid Memory Cube (HMC) [4] and the High Bandwidth Memory (HBM) DRAM [9]. The logic layer can accommodate additional processing capabilities [19, 20, 12, 15]. Processing-in-Memory is the concept of moving computation closer to mem-

ory that was investigated a decade ago [13, 19, 6, 10]. Researchers explored how to integrate logic with memory for various applications.

MapReduce framework for large-scale data processing on clusters of commodity machines was first developed by Google [5]. It involves three major phases: map, reduce and merge. The user supplies the map() and reduce() functions and the MapReduce runtime manages parallelization. Several different types of frameworks are available for MapReduce, including Google MapReduce [5], Apache Hadoop [1], MRMPI [14] for commodity clusters, Phoenix++ [18], Metis [11], Ostrich [3] for shared memory systems, Mars [8], GPMR [17] for systems using GPUs. We model our PIM architecture as a shared memory system, albeit with Non-Uniform Memory Access (NUMA). Thus we rely on shared memory MapReduce frameworks.

The Phoenix system [18, 16] and others [11, 3] provide MapReduce framework for conventional large-scale shared memory CMP and SMP systems. We use Phoenix++ [18], the most recent and highly optimized MapReduce framework with NUMA-awareness for our study and propose changes to adapt it to our PIM architecture. The architecture and MapReduce framework we propose differ from the Phoenix++ system presented by Talbot et al. [18]. We also diverge from Google MapReduce [5] and Hadoop [1] in node-level task execution.

Recently proposed Near Data Computing (NDC) architecture [15] provides a similar idea to our study and assumes 3D-DRAMs embedded with processing cores. However the NDC study works with in-memory MapReduce workloads where the entire input for computation is assumed to reside in the system memory. We do not make such assumptions but consider conventional storage systems (e.g. Hard Disk Drive-HDD, Solid State Drive-SSD) as the source of input. This difference significantly changes how we approach the Map and Reduce functions using PIM cores.

3 Proposed PIM Architecture

PIM architectures could prove beneficial for data intensive and memory bounded applications that may not necessarily benefit from a cache hierarchy [26]. PIM cores can access memory using faster, high bandwidth, lower power TSVs [15, 20, 24]. Therefore moving the computation from the main processor closer to the memory is a better choice for such applications. We base our server architecture on the model proposed by Zhang et al. [20]. The server consists of a host multi-core processor. The host is connected to four 3D-stacked DRAM Memory Units (3DMUs). The host views the entire memory as a single physical memory distributed among the 3DMUs. Figure 1 depicts the proposed PIM architecture.

Each 3DMU has several dedicated Processing-in-Memory cores (PIM cores) embedded in its logic layer. Each PIM core is a simple in-order, single-issue, energy efficient processing unit operating at a lower clock frequency than that of the host cores. The system memory for the host and PIM cores is comprised of the DRAM layers in the 3DMUs. We also assume that each PIM core has its own small instruction and data caches. The execution of the threads running

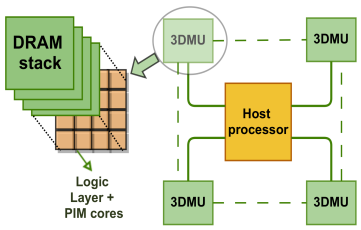

Fig. 1: Proposed Hardware Architecture

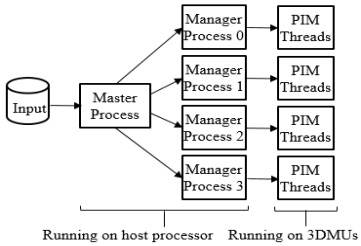

Fig. 2: Proposed Programming Model

on PIM cores is controlled by a manager process running on a host core. The threads can access any physical address residing in any 3DMUs which are part of its manager process address space. However, accesses to data in other 3DMUs should be limited to avoid performance losses due to NUMA.

For our initial analyses we assume 8GB memory and 16 PIM cores in each of the four 3DMUs. In section 5 we justify the number of PIM cores per 3DMU and argue about a good number depending on the system usage. Note that the number of PIM cores in the logic layer should be small enough not to exceed the 10W TDP of the logic layer [24]. Different architectural choices for the PIM cores also play a big role, in terms of both performance and energy consumption. At this time we assume ARM-like processing cores [21] as PIM cores. The proposed server architecture can be used as a node in a cluster configuration for dealing with very large amounts of data.

4 MapReduce using PIM

MapReduce workloads are memory intensive and do not benefit much from conventional deep cache hierarchies [26]. Our goal is to optimize node level performance of a MapReduce cluster by parallelizing the different phases with the help of PIM cores. Our proposed MapReduce framework consists of two levels, one is inter-node level (using processing nodes of a cluster) and the other is intra-node level. The inter-node level execution flow can be similar to a standard MapReduce framework like Hadoop [1]. The intra-node level reduce phase performs local (node-level) optimization before sending the results for inter-node level reduce phase i.e. global reduction.

Since the server architecture proposed here is a special case of hierarchical multi-core system with NUMA shared-memory (from the PIM cores point of view), we have used Phoenix++ [18], a MapReduce framework designed for large-scale SMP systems that exhibit NUMA behavior, as our base. We pay particular interest to the structures for the intermediate <key, value> stores. Because of these structures we can use PIM cores to efficiently parallelize map, reduce as well as part of the merge phase. Additionally, we propose changes in the actual Phoenix++ MapReduce flow, where we overlap the reading of the input from storage with the actual map phase. The key issues related to MapReduce

applications, when executed on shared memory systems, are to ensure the locality of map phase, selection of efficient intermediate data structures, decrease remote memory access during the reduce phase and to use an efficient memory allocator [18, 11].

4.1 Intra-Node MapReduce using PIM

Execution Flow of the Intra-Node MapReduce Framework. We assume that any process running on a host core can request the runtime system to allocate physical memory in any specific 3DMU, and thus aware of the location of the data for the purpose of spawning PIM tasks on that memory unit. This is a valid assumption because HSA (Heterogeneous System Architecture) Foundation [7] is advocating such an organization. We next describe the MapReduce runtime on a node level with respect to the architecture shown in Fig. 1. There is one master process, which creates 4 manager processes (corresponding to 4 3DMUs) and each manager process creates 16 worker threads on the 16 PIM cores of a certain 3DMU. Inter-process communication is achieved through shared memory. We label the 3DMUs and the manager processes from 0-3 so that each manager corresponds to a 3DMU respectively. Figure 2 depicts the model.

Map and Combine Phase. In Phoenix++, the library reads the input data from disk and keeps it in a single memory buffer prior to starting the map phase. However, in order to obtain maximum parallelism we will overlap these two phases. We next describe the overlapping process using the aforementioned numbered labels for better understanding. The master process reads 16 input splits at a time from the disk and places them in a shared memory buffer residing in 3DMU-0. The master process then starts reading the next 16 input splits into 3DMU-1. In parallel, the manager process-0, which manages the 16 PIM core threads of 3DMU-0, allocates necessary memory in 3DMU-0 for the PIM threads to generate the intermediate output and then hands the execution over to them. Each thread will start processing one input split with the provided map function. This process is repeated until all of the input is processed.

Reduce and Merge Phase. The reduce phase across the 3DMUs is assumed to be completed by the manger processes running on the host processor either independently or with the help of PIM threads. The reduce phase can potentially benefit from the parallel reduction on sets of unique keys. Initial stages of the merge phase can be performed by the PIM cores in parallel as well.

5 Experiments and Results

5.1 Experiments

In order to evaluate the proposed architecture, we use a conventional server as our baseline system. The configuration is provided in Table 1. This baseline

Table 1: Baseline and New System Configuration

	Baseline System Configuration	New System Configuration	
		Host Processor	PIM Cores
Processing Units	2 × Xeon E5-2640 6 cores/processor, 2 HT/core Out-of-order 4-wide issue	1 × Xeon E5-2640 6 cores, 2 HT/core Out-of-order 4-wide issue	64 (4 × 16) ARM Cortex-A5 In-order Single-issue
Clock Speed	2.5 GHz	2.5 GHz	1 GHz
LL Cache	15 MB/processor	15 MB	32 KB I, D/core
Memory BW	42.6 GB/s per processor	42.6 GB/s	1.33 GB/s per core
Power	TDP = 95 W/processor Low-power = 15 W/processor	TDP = 95 W	80 mW/core (5.12 W for 64)
Memory	32 GB (8 × 4GB DIMM DDR3)	32 GB (4 × 8GB 3DMU)	
Storage	1 TB HDD, SATA3, PERC H710	1 TB HDD, SATA3, PERC H710	
MapReduce	Phoenix++ Framework	Proposed Framework (Sect. 4)	

Table 2: MapReduce workload execution time (in seconds) for baseline system

Workload	IP Size	t_{baseline} (s)	t_{read} (s)	t_{map} (s)	t_{reduce} (s)	t_{merge} (s)
word_count	16 GB	176.67	162	14.6	0.05	0.02
histogram	1.3 GB	13.254	12.9	0.35	0.002	0.002
string_match	16 GB	186.61	181	5.6	0.01	0.0
linear_regression	16 GB	185.61	181	4.6	0.01	0.0

Table 3: $t_{\text{transfer\_unit}}$

Storage Technology	$t_{\text{transfer\_unit}}$
HDD	10.42 ms
SSD	2.17 ms

Table 4: $t_{\text{map\_unit\_host}}$

Workload	$t_{\text{map\_unit\_host}}$
word_count	25 ms
histogram	7 ms
string_match	12 ms
linear_regression	7 ms

system runs Phoenix++ library [16] with its standard setup. In Table 1, we summarize the new system configuration we envision, which will be running the modified MapReduce framework described here.

We use Phoenix++ to obtain the execution times for the baseline system and to estimate the execution times for the proposed MapReduce framework running on the PIM architecture. The total execution time of a MapReduce workload on the baseline system can be expressed as:

$$t_{\text{baseline}} = t_{\text{read}} + t_{\text{map}} + t_{\text{reduce}} + t_{\text{merge}} \ . \tag{1}$$

In the baseline configuration there are 24 threads and 16 map tasks per thread (total 384 map tasks). We ran different workloads on the baseline system for different input sizes from 100MB up to 16GB and Table 2 shows execution times for a specific input size. From the collected statistics we compute the following two parameters: $t_{\text{transfer\_unit}}$, time to read one input split (1MB) from

storage into memory (Table 3) and $t_{\text{map\_unit\_host}}$, time to process an input split (1MB) by one map task running on the host (Table 4). In the baseline system we also have used Samsung PM830 SSD as storage and run the benchmarks. In this case we observed around 4.8 times speedup in reading the input as compared to HDD storage as implied by Table 3 data. We also run them with input sizes larger than the physical memory (32GB), the results are discussed in Sect. 5.4.

5.2 Performance Analysis

The execution time benefit of the proposed MapReduce model lies in the overlapping of map tasks with the reading of input from storage to memory. As long as the PIM cores do not sit idle waiting for input buffers to get filled, we believe that this approach delivers performance improvements over a serialized process where all of the input is first read before starting map tasks.

For the baseline system, from (1) the total execution time is $t_{\text{baseline}} = t_{\text{read}} + t_{\text{map}} + t_{\text{reduce}} + t_{\text{merge}}$. For different workloads we find that when the input size is smaller than that of available physical memory then, $t_{\text{read}} > t_{\text{map}} + t_{\text{reduce}} + t_{\text{merge}}$. We discuss the case when the input is larger than available physical memory in Sect 5.4. To reduce the total execution time we overlap the read and map phases in our MapReduce framework. Hence the total execution time for the proposed PIM based system is:

$$t_{\text{new}} = t_{\text{read}} + t_{\text{reduce}} + t_{\text{merge}} \ . \tag{2}$$

In order to achieve (2), we must ensure:

$$t_{\text{map}} \leq t_{\text{read}} \ . \tag{3}$$

thereby t_{map} is completely overlapped with t_{read}.

Another important fact is that the processing speed of PIM cores will be slower than the host processor since PIM cores operate at lower clock rate and use in-order single-issue execution. On the other hand, PIM cores are sitting closer to memory so memory accesses are faster for them. We performed a simulation using gem5 [25], and compared the execution time of a map function running on an OoO X86 and an In-Order ARMv7 CPU model. The simulation parameters were picked to mimic the actual CPU specifications in Table 1. We find that the PIM cores would run approximately 4 times slower (i.e., slowdown factor, $s = 4$) than the host cores.

Initially we proposed a server with four 3DMUs each with 16 PIM cores. The following analysis will explain why we choose 16 PIM cores per 3DMU. We wanted to know the minimum number of PIM cores needed on each of the 3DMUs in order to satisfy (3). Each PIM core runs one thread and processes one input split at a time. In our case, following must hold for (3) to be true,

$$s \times t_{\text{map\_unit\_host}} \leq 4 \times n \times t_{\text{transfer\_unit}} \ . \tag{4}$$

In (4), s is the slowdown factor ≥ 1, $t_{\text{map\_unit\_host}}$ is the time to process an input split (1MB) by one map task running on the host core, 4 is the number

of 3DMUs in the server, n is the number of PIM cores in each 3DMU and $t_{\text{transfer\_unit}}$ is the time to read one input split (1 MB) from storage into memory. Here we want the time taken by a group of PIM cores to process the input splits to be smaller than, or equal, to the time taken by the host to fill in the buffers in each of the 3DMUs.

We observe two cases, depending on how fast the PIM cores can process the input in Fig. 3 (a) and (b). The host keeps reading input splits from storage as long as there is more input. As soon as the input is available in a 3DMU, the PIM cores in that 3DMU start the map tasks. In Fig. 3(a) the PIM cores are processing the input at a much higher rate than the host can fill in the buffers. In Fig. 3(b) the PIM cores in each 3DMU are busy processing the input almost up to the point of time when the next set of input splits becomes available.

To achieve full utilization of the PIM cores following must hold,

$$s \times t_{\text{map\_unit\_host}} = 4 \times n \times t_{\text{transfer\_unit}} \quad . \tag{5}$$

We solve (5) to find the minimum n (number of PIM cores per 3DMU) for each of the workload independently. We use data from Table 3 and 4, and compute n for a range of slowdown factors s. Figure 4 shows the required number of PIM cores per 3DMU for different slowdown factors for different workloads.

Analyzing the graphs in Fig. 4 for two different storage technologies and four different workloads one can conservatively estimate (choosing the closest greater or equal integer which is a power of two) the number of PIM cores needed per 3DMU as 16 when estimated 4 times slower execution (s=4) of the map tasks on a PIM core. Our study allows one to decide on the minimum number of PIM cores per 3DMU needed so that t_{map} is completely overlapped with t_{read}. One can use more PIM cores than the minimum, but their utilization will drop.

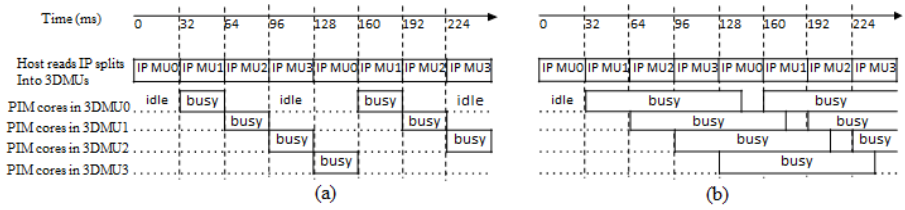

Fig. 3: (a) PIM core utilization is low (b) PIM core utilization is high

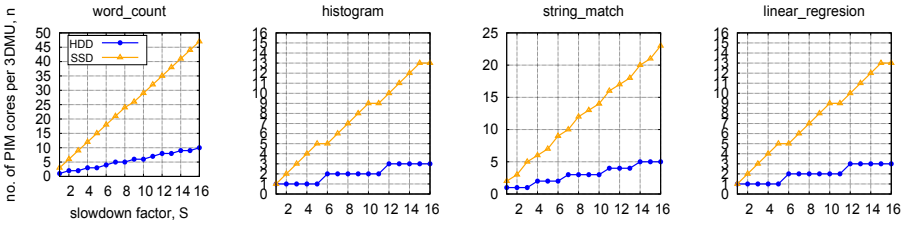

Fig. 4: Each graph shows the number of PIM cores required per 3DMU (Y axis) for different slowdown factors (X axis) for 2 different storage technologies, HDD and SSD.

We also analyze the area and power overhead of placing 16 PIM cores in the logic layer of each 3DMU with ARM Cortex-A5 core as PIM core. Each such core, with 32KB data and 32KB instruction cache, has an area of 0.80mm^2 in 40nm technology [21]. So 16 PIM cores in the logic layer have an area overhead of 11.9% [15] when HMC [4] is used as 3DMU. Furthermore, accumulated power consumption of the 16 PIM cores will be 1.28W [21] which is only 12.8% of allowable 10W TDP of logic layer per stack [24]. Therefore we claim that integrating 16 PIM cores in the logic layer of each 3DMU is feasible.

We conclude that the total execution time in the proposed model, t_{new} is faster than total execution time in the baseline system, t_{baseline} by t_{map}. Figure 5 shows t_{new} normalized to t_{baseline} for different workloads. We use data from Table 2 for t_{baseline}, for t_{new} we have used (2), and we take the estimated slowdown factor of 4. We observe that the overall execution time for the proposed model is reduced by 2.5% to 8.8% when compared to the baseline system for different Big Data workloads. This evaluation includes only the performance gain for the map phase; additional speedup may be achieved by parallelizing reduce and merge (partially) phases on the PIM cores.

5.3 Energy Consumption

The total energy consumption of running a Big Data workload in the proposed system is reduced by using lower power cores as well as decreasing the overall execution time. We define E_{baseline} and E_{new} as the total energy consumed by the processing elements of the baseline and the new system respectively. The baseline system consists of two Xeon processors (Table 1). To make our analyses fair, and even favor the baseline, while computing E_{baseline}, we assume that only one of the processors in the baseline is active while reading the input i.e. during the time t_{read} the second processor will be placed in low power state consuming 15W. For the other phases both processors are fully active. While computing E_{new}, we assume that the host processor and all the 64 PIM cores (Table 1) are active during the entire processing. We calculate the energy consumption of the processing units for the baseline and the new system as follows.

$$E_{\text{baseline}} = [(TDP + P_{\text{low\_power\_state}}) \times t_{\text{read}}] + [2 \times TDP \times (t_{\text{map}} + t_{\text{reduce}} + t_{\text{merge}})] \ . \quad (6)$$

$$E_{\text{new}} = [TDP + (64 \times P_{\text{PIM\_core}})] \times (t_{\text{read}} + t_{\text{reduce}} + t_{\text{merge}}) \ . \quad (7)$$

The power specifications for baseline and new system are listed in Table 1 and the execution times of the different phases are given in Table 2. Figure 6 shows that, for processing part, relative energy savings of one node range from 12% to 23% as compared to the baseline system. The absolute energy savings range from 80J to 2045J, depending on the workload.

5.4 Input Exceeding Physical Memory Capacity

If the input is larger than the available physical memory, we observe a non-linear increase in map phase execution time (t_{map}) for our baseline. This happens

because by the time map phase starts, all the starting pages containing the input are swapped out and there will be a large number of page faults. In some cases we even have $t_{\text{map}} > t_{\text{read}}$ (e.g. word_count in Table 5). This would not happen if the input splits, on which the map tasks will work, were in the memory. In our proposed model we handle such cases by bringing input splits into memory and performing map tasks on them in an incremental fashion.

Table 5 shows the execution times for workloads in the baseline system (Table 1) for input size larger than the physical memory. In such cases, with our proposed model, one can achieve up to 56% reduction in execution time and up to 71% energy savings on the processing units compared to the baseline system, as calculated by (2) and (7) respectively. Note that here we get these numbers for stand alone server performance. But in such cases, where the input is larger than physical memory, one may choose to use a cluster of such nodes and for each node we may get statistics as Table 2 and obtain gains as presented in Sects. 5.2 and 5.3.

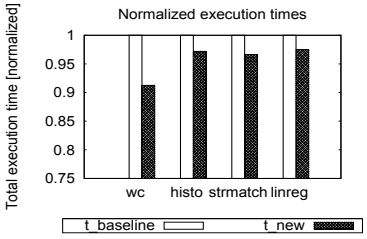

Fig. 5: Normalized execution times for 16 PIM cores per 3DMU with slowdown factor of 4, as compared to the baseline. The total execution time is reduced by 2.5% to 8.8%.

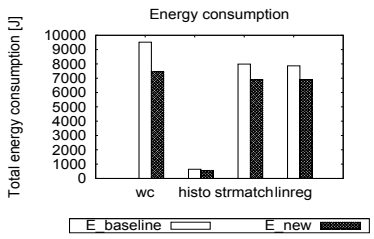

Fig. 6: Energy consumption of processing units of the PIM model compared to the baseline. Energy savings range from 80J to 2045J (12% to 23%).

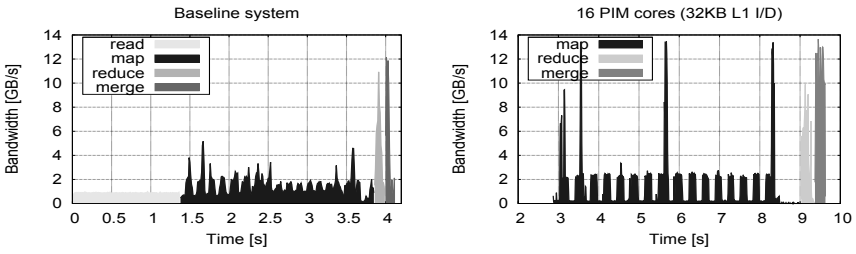

Fig. 7: Bandwidth consumption when running word_count on two different systems.

Table 5: Execution time for baseline system when input is larger than physical memory

Workload	IP Size	t_{baseline} (s)	t_{read} (s)	t_{map} (s)	t_{reduce} (s)	t_{merge} (s)
word_count	32.5 GB	801.358	349.981	449.731	1.605	0.041
string_match	32.4 GB	538.346	348.127	190.198	0.021	0.0
linear_regression	32.5 GB	466.538	365.959	100.559	0.02	0.0

5.5 Bandwidth Utilization and Link Power

Figure 7 shows the actual bandwidth consumption of word_count for different MapReduce phases when running on different systems. Interestingly, the bandwidth consumed by the baseline system does not exceed 15GB/s. PIM cores show higher bandwidth utilization at lower power consumption. For our proposed PIM server we can have low bandwidth links between the host processor and the 3DMUs and thereby reduce power consumption. Note that during the map phase the peak bandwidth required will depend on whether or not the intermediate data structures fit in the PIM core caches. 3DMUs provide memory bandwidth of up to 320GB/s within the memory stack [4, 23]. The same bandwidth is available to the host processor via 8 high speed SerDes links [4], each of which provides bandwidth of 40GB/s with average power consumption of 5W [23]. We believe that PIM architectures are more energy efficient than traditional heterogeneous multi/many core architectures because they utilize the bandwidth available within the memory stack and do not need the power hungry SerDes links. The bandwidth consumed by the 16 PIM cores in one 3DMU will not exceed 22GB/s [22] which is well below the 320GB/s available in the unit. However, 64 PIM cores in four 3DMUs will have an effective peak bandwidth consumption of 88GB/s. In order to support the same bandwidth for the system with off the 3D-DRAM chip heterogeneous cores, we would need at least 3 SerDes links, consuming three times more energy on the links.

The bandwidth utilized within each 3DMU can be further increased by increasing the number of PIM cores per 3DMU, however at the expense of higher power consumption and possibly lower utilization. To fully utilize the 320GB/s bandwidth, more than 200 PIM cores are needed, but then the power consumption will exceed the constraint of 10W TDP for the logic layer of a 3DMU [24].

6 Conclusion and Future Work

In this paper we outlined our ideas about using simple cores embedded within the logic layer of 3D-DRAMs for running MapReduce applications. We overlap input reading and map phases. We also propose to utilize locality of data for assigning tasks to PIM cores. Our preliminary results show gains in terms of reduced execution time and energy savings for several MapReduce applications.

We intend to extend our preliminary work in several directions. First we want to explore other possible architectures for PIM cores, including GPGPUs, simple RISC cores, FPGA and Dataflow. Second, we want to characterize which emerging workloads, and particularly which functionalities, benefit from a PIM architecture and how to exploit the possible benefits. This includes extensive simulation of memory intensive workloads in a PIM augmented system in order to show the benefits in terms of energy savings as well as performance gains.

Acknowledgments. This work is conducted in part with support from the NSF Net-centric IUCRC and AMD. We acknowledge David Struble's help in making this paper more readable.

References

1. Apache Hadoop, http://hadoop.apache.org/
2. Black, B., Annavaram, M., Brekelbaum, N., DeVale, et al.: Die stacking (3D) microarchitecture. In: Micro, pp. 469-479. IEEE, (2006)
3. Chen, R., Chen, H.: Tiled-MapReduce: Efficient and Flexible MapReduce Processing on Multicore with Tiling. In: Transactions on Architecture and Code Optimization 10(1), 3. ACM, (2013)
4. Hybrid Memory Cube Consortium, http://hybridmemorycube.org/
5. J. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of the conference on Symposium on OSDI, vol. 6. (2004)
6. Draper, J., Chame, J., Hall, M., et al.: The architecture of the DIVA processing-in-memory chip. In: Proceedings of the Supercomputing, pp. 14-25. ACM, (2002)
7. HSA Foundation, http://www.hsafoundation.com/
8. He, B., Fang, W., Luo, Q., et al.: Mars: a MapReduce framework on graphics processors. In: Proceedings of Parallel architectures and compilation techniques, pp. 260-269. ACM, (2008)
9. JEDEC, http://www.jedec.org/category/technology-focus-area/3d-ics-0
10. Rezaei, M., Kavi, K. M.: Intelligent memory manager: Reducing cache pollution due to memory management functions. In: Journal of Systems Architecture, 52(1), 41-55. (2006)
11. Mao, Y., Morris, R., Kaashoek, M. F.: Optimizing MapReduce for multicore architectures. In: CSAIL, Massachusetts Institute of Technology, Tech. Rep. (2010)
12. Loh, G., Jayasena, N., Oskin, M., et al.: A Processing in Memory Taxonomy and a Case for Studying Fixed-function PIM. In: Near-Data Processing workshop. (2013)
13. Patterson, D., Anderson, T., Cardwell, N., et al.: A case for intelligent RAM. In: Micro, 17(2), 34-44. IEEE, (1997)
14. Plimpton, S. J., Devine, K. D.: MapReduce in MPI for large-scale graph algorithms. In: Parallel Computing, 37(9), 610-632. (2011)
15. Pugsley, S. H., Jestes, J., Zhang, H.: NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads. In: International Symposium on Performance Analysis of Systems and Software. (2014)
16. Phoenix System for MapReduce Program, http://mapreduce.stanford.edu/
17. Stuart, J. A., Owens, J. D.: Multi-GPU MapReduce on GPU clusters. In: Parallel and Distributed Processing Symposium, pp. 1068-1079. IEEE (2011)
18. Talbot, J., Yoo, R. M., Kozyrakis, C.: Phoenix++: modular MapReduce for shared-memory systems. In: Proceedings of the international workshop on MapReduce and its applications, pp. 9-16. ACM, (2011)
19. Torrellas, J.: FlexRAM: Toward an advanced Intelligent Memory system: A retrospective paper. In: Intl. Conference on Computer Design, pp. 3-4. IEEE, (2012)
20. Zhang, D. P., Jayasena, N., Lyashevsky, A., et al.: A new perspective on processing-in-memory architecture design. In: Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, p. 7. ACM, (2013)
21. ARM, http://www.arm.com/products/processors/cortex-a/cortex-a5.php
22. Atmel SAMA5D3, http://www.atmel.com/microsite/sama5d3/highlights.aspx
23. Graham, S.: HMC Overview. In: memcon Proceedings. (2012)
24. Zhang, D., Jayasena, N., Lyashevsky, A., et al.: TOP-PIM: throughput-oriented programmable processing in memory. In: Proceedings of international symposium on High-performance parallel and distributed computing, pp. 85-98. ACM, (2014)
25. gem5 Simulator System, http://www.m5sim.org
26. Ferdman, M., Adileh, A., Kocberber, O., et al.: A Case for Specialized Processors for Scale-Out Workloads. In: Micro, pp. 31-42. IEEE, (2014)