LooPIM: A Loop-Oriented Acceleration Framework for Processing-in-Memory

Chuangshi Liu¹, Xianfeng Li²

¹School of Electronic and Computer Engineering, Peking University, Shenzhen, China
²Macau University of Science and Technology, Macau, China

Abstract. The latest development of 3D stacking technology provides a feasible solution for processing in memory (PIM) to solve the memory wall crisis. Since data can be processed locally in a logic layer adjacent to memory, large emerging big data applications may achieve significant performance and energy-efficiency benefits. Researchers continue to propose various PIM architectures to utilize the advantages of stacked memory efficiently. However, previous efforts relied on manually mapping specialized kernels to the logic layer, making it infeasible to perform more general workloads. We propose a loop-oriented acceleration framework that intelligently partitions loops and maps them onto appropriate execution units. The mapping mechanism can significantly reduce the communication overhead between the host processor and the PIM stack. Besides, we introduce a simple execution model that exploits data-level parallelism transparently. Experiments show that our framework achieves approximately 2.4x performance speedups comparing to the baseline 3D memory system.

Keywords: Processing-in-Memory, Data-Level Parallelism, 3D stacked memory, acceleration, Heterogeneous (hybrid) systems

1. Introduction

In modern computer systems, the vast cost of transferring data through the deep memory hierarchy has become a significant threat to performance and energy efficiency. This issue is expected to worsen due to two trends: (1) the gap between processor speed and memory access speed keeps growing, and (2) emerging data-intensive applications require more and more bandwidth. Processing In-Memory (PIM) is one of the most effective ways to avoid data movement problems. Interest in PIM has been revived with the development of 3D stacking technology. Researchers have proposed various outstanding solutions on PIM logic layer architecture to use stack memory advantages fully.

Although there are many viable PIM architecture implementations, we now need to address a new challenge of PIM. The PIM architecture is usually used as a coprocessor or accelerator in collaboration with the host processor. In this heterogeneous system, tasks can be performed on the CPU or in memory. Therefore, the proper schedule of the application to the available resources is critical for achieving high performance. However, previous researches have paid more attention to the architecture design of the PIM system. A small number of studies involving task assignments can be broadly divided into two categories. First, the PIM is designed as a dedicated accelerator to accelerate the specific kernel operations involved in applications [1-3]. This coarse-grained assignment of tasks often relies on artificial speculation, which has a high learning threshold to programmers, so that this division mechanism can be subjective and inaccurate. Second, the other type corresponding to the
above is a fine-grained allocation scheme based on some means, such as PEI [4]. One potential problem is that PIM architectures can suffer from a significant communication overhead between the host processor and the PIM stack. This overhead degrades the performance and energy efficiency of the PIM-based system.

To address these two limitations, we propose a loop-oriented acceleration framework. Applications are partitioned according to loops' bounds because there are substantial data dependencies inside the body and relatively few dependencies outside the body. Then, we introduce a quantifiable factor that determines where the loops are going to be executed. This factor synthetically considers both memory access density and parallelism in the loop by a weighted sum of the corresponding parameters. The loops with interdependent relationships are assigned to the same processing unit for processing, so the communication overhead between heterogeneous systems can be reduced from the source. Moreover, multiple processing units included in a PIM architecture make it possible to leverage DLP within a loop. Evaluation results show our acceleration framework significantly improves the system performance compared to CPU+HMC systems.

2. Background and motivation

2.1. 3D-Stacked Processing in Memory

The core idea of processing in memory, also known as near data computing, is to integrate the computing unit and storage module in the same chip so that the memory unit can process data locally. With the advent of big data, the performance bottlenecks of traditional architectures need to be solved more and more urgently. Explosive data growth and the emergence of new manufacturing processes provide a unique opportunity to develop PIM architecture. Recent advances in die stacking technologies make it possible to integrate logic and memory die in a single package at a low cost. Recently, Micron introduced the Hybrid Memory Cube (HMC) [5], a 3D-stacked memory that integrates multiple DRAM layers and a single logical layer. The logic layer consists of the memory controller and interfaces logic. The different layers are connected using high-bandwidth through-silicon-vias (TSVs) [6], which can provide aggregated internal bandwidth of up to 320Gb/s between the logical layer and the DRAM layer. In recent years, the research direction of PIM has become more and more diversified, and the research results have covered different levels.

2.2. Off-chip traffic between the host processor and the PIM stack

A PIM-based system is a heterogeneous system in which data processing may occur either in the host processor or in memory. Therefore, it is inevitable to face data consistency and synchronization in two areas under the heterogeneous system. The cache consistency protocol used in the traditional multi-core processor architecture cannot be directly used in the PIM architecture. The data synchronization problem in PIM architecture is closely related to the allocation mechanism. The data consistency protocol should be designed according to the granularity of data cache and the way of task division and assignment to solve the problem of data synchronization. Fine-grained data synchronization causes frequent on-chip and off-chip communication overhead. In contrast, coarse-grained data synchronization may increase data dependency, cause waiting for instruction execution, and reduce concurrent application execution.

However, prior researches on PIM mainly focus on the architecture and hardware design of PIM memory systems and assumes that there is only a limited amount of sharing between the PIM logics and the processor. However, this is not the case for many important applications, such as graph and database workloads [7]. To get the maximum benefit from PIM architecture execution, we generally assign memory-intensive programs that take up most of the execution time to the PIM core. On the other hand, compute-intensive parts that exhibit high locality must be kept in the processor core to maximize benefits. As data sharing occurs, the previous approaches to data-coherence eliminate some benefits of the PIM architecture due to its higher coherence overhead. They can sometimes lead to poor PIM execution performance. Therefore, the impact of the communication overhead between the
host processor and the PIM stack on system performance must be fully considered to retain PIM architecture’s benefits in a wide range of applications.

3. LoopPIM mechanism

3.1. Loop-oriented task partitioning and mapping strategy
Due to the architectural characteristics of processing-in-memory itself, PIM architecture has natural advantages in handling data-intensive applications. The inspiration for partitioning applications into loops comes from three aspects: 1) loops often take up most of an application's execution time, 2) there are strong data dependencies within the loop and weak data dependencies between the loops, 3) there is a lot of DLP potential inside the loop. These make it possible for us to solve the problem of communication overhead mentioned above.

First, we get two files by profiling the source code: 1) Loop Partition Configuration (LPC) and 2) Loop Dependency Flow Diagram (LDFD). The LPC records all loops’ boundary information and tags indicating memory-density and parallelism. It’s worth noting that the memory-density is determined by the ratio of memory-access instructions and computational instructions. The positive parallelism tag means that the loop body can be entirely executed in parallel. According to the observation, data-intensive and parallelizable tasks have a significant acceleration effect in the PIM stack (the tag is positive). In contrast, the computation-intensive and non-parallel tasks are suitable for the host processor (the tag is negative). The remaining tasks with no acceleration potential cause a high communication overhead between the PIM stack and the host processor (the tag is zero).

Second, all loops are divided into several loop-trees according to LDFD. A loop-tree containing one or more loops with data dependencies is the smallest unit of task mapping we can do. The sum of all the loops’ tags in the loop-tree represents the location where the loop-tree is executed. This sum is called the weight factor (WF). A positive WF indicates that the program has significant acceleration potential performed in the PIM stack. If the WF is negative, the program should execute on the host processor. If WF happens to be zero, the program is mapped to an idle processing unit for execution.

3.2. Execution model in parallel
Figure 1 provides an overview of our architecture. We choose the hybrid memory cube (HMC) as our baseline memory technology. In general, a PIM-based system is composed of a high-performance host processor and a PIM stack. The host processor communicates with the PIM stack using high-speed I/O links.

As shown in figure 1, we integrate an energy-efficient in-order core and a set of PE arrays into the memory control interface of the PIM logical layer. The processing logic can communicate directly with the host processor and easily access data distributed across multiple vaults. The in-order core is

![Figure 1. A baseline PIM system and overview of the proposed architecture.](image-url)
used to execute non-parallel loop tasks and is responsible for unrolling the parallelable loop tasks to the PE arrays. The in-order core initializes the PE arrays, and then the parallelizable loop tasks are processed by the PEs.

4. Experimental evaluation
Because the PIM-based architecture is a heterogeneous system, to save the cost and ensure the experiment's accuracy, the current PIM architecture is performed in an emulator environment. In this experiment, we use the full-system simulator GEM5 to simulate the PIM architecture. Gem5 is an open-source simulation platform that can be flexibly configured. It integrates a variety of instruction architectures, CPU models, and memory simulators. The detailed configuration parameters of this experiment are shown in the following table 1.

| Component    | Configuration                                      |
|--------------|----------------------------------------------------|
| Host Processor | 8 out-of-order cores, 2GHz, 4-issue                |
|              | L1 I/D-cache: Private, 32KB, 4-way, 64B blocks     |
|              | L2 cache: Shared, 2MB, 8-way, 64B blocks           |
| PIM Processor | 8 in-order cores, 1GHz, single-issue              |
|              | L1 cache: 32KB, 4-way, 64B blocks                  |
| HMC          | 32GB, 8layers, 16vaults, 8stacks                  |
|              | Off-chip links: 16Bytes/cycle, four links          |

4.1. Workloads
We evaluated seven emerging data-intensive workloads from three key areas that are often categorized as big data workloads, including large-scale graph computing applications, natural language processing, and machine learning algorithms. They are Latent Dirichlet allocation (LDA), K-means, Liblinear [8], Breadth-First Search (BFS), PageRank, Average Teenage Follower (ATF), and Single Source Shortest Path (SSSP) [9].

4.2. Experiment Result

![Figure 2. The performance evaluation result of target applications.](image)

We compared the applications' performance under the three mechanisms:
- **Host-only** runs the application on a conventional system with memory and CPU.
- **PIM-only** runs the application entirely on HMC.
- **LooPIM** runs the application on our LooPIM system.

The experimental results are shown in Figure 2. The results show that the proposed framework has a significant acceleration effect for parallel data-intensive applications, with an average acceleration of 2.4 times compared with host-only. The k-means application is friendly to the host processor due to its sequential memory access, so its acceleration effect is not as significant as other random-access
Due to the strong data dependencies between loops, BFS is executed entirely on the near memory side. So its acceleration effect is weaker than that of the other three graph processing applications. However, its acceleration performance is still superior to the PIM-only mechanism because LooPIM takes advantage of the program's parallelism.

5. Conclusion
This paper carefully considers the communication overhead between the host processor and PIM stack, which was ignored by previous research. We have designed and implemented a loop-oriented framework called LooPIM, which is a practical model for processing-in-memory. In this framework, programs are divided into some loops based on the data dependencies. These loops are then mapped to the appropriate core for execution based on the memory-density and parallelism to reduce the communication overhead caused by data dependence. By the way, we also modify the PIM stack's hardware implementation so that the parallelism in the loop is fully explored. The efficiency of the proposed framework comes from two sources: massive parallelism and reduction in data movement. The experimental results show that our framework achieves performance significantly improved compared to traditional methods.

6. References
[1] R. Balasubramonian et al., "Near-Data Processing: Insights from a MICRO-46 Workshop," in IEEE Micro, vol. 34, no. 4, pp. 36-42, July-Aug. 2014, doi: 10.1109/MM.2014.55.
[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, USA, 2015, pp. 105-117, doi: 10.1145/2749469.2750386.
[3] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar and H. Kim, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, USA, 2017, pp. 457-468, doi: 10.1109/HPCA.2017.54.
[4] J. Ahn, S. Yoo, O. Mutlu and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, USA, 2015, pp. 336-348, doi: 10.1145/2749469.2750385.
[5] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," 2012 Symposium on VLSI Technology (VLSIT), Honolulu, HI, USA, 2012, pp. 87-88, doi: 10.1109/VLSIT.2012.6242474.
[6] M. Motoyoshi, "Through-Silicon Via (TSV)," in Proceedings of the IEEE, vol. 97, no. 1, pp. 43-48, Jan. 2009, doi: 10.1109/JPROC.2008.2007462.
[7] A. Boroumand et al., "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," in IEEE Computer Architecture Letters, vol. 16, no. 1, pp. 46-50, 1 Jan.-June 2017, doi: 10.1109/LCA.2016.2577557.
[8] S. Thomas et al., "CortexSuite: A synthetic brain benchmark suite," 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, NC, USA, 2014, pp. 76-79, doi: 10.1109/IISWC.2014.6983043.
[9] S. Hong et al., "Simplifying scalable graph processing with a domainspecific language," in Proc. CGO, 2014, pp. 20S-21S.

Acknowledgments
The work is sponsored by Shenzhen Science and Technology Innovation Committee (SZSTI) under the grant for basic research (NO. JCYJ20170818090006804).