PIM Architecture Exploration for HMC

Sangwoo Han, Hyeokjun Seo, ByoungJin Kim, Eui-Young Chung
Dept. of Electrical and Electronic Engineering
Yonsei University
Seoul, Korea
swhan0330@dtl.yonsei.ac.kr, jjsky7@dtl.yonsei.ac.kr, bryankim@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr

***Abstract*— Modern processor and memory have significant performance gap that incurs memory-wall phenomenon causing overall system performance degradation. Recently, PIM came out as one of solutions to overcome memory-wall phenomenon as well as increasing system performance. This paper studies how to implement functions within PIM logic for some broadly used applications. Our experiments are based on gem5 simulator. The experimental results show that some of functions are more efficient with PIM and their efficiency depends on the data size.**

Keywords— Processing in Memory; Hybrid Memory Cube;

I. INTRODUCTION

Memory has been developed towards high density, while processor has been aimed to high performance. They made huge performance difference between processor and memory, causing memory-wall phenomenon. Eventually, modern memory cannot provide high-bandwidth what processor needs, results in overall system performance degradation. To overcome this problem, lots of research about processing in memory (PIM) has been studied recently [1][2].

Different from Von Neumann architecture where processor loads data from memory, executes operation, and then stores data to memory, PIM itself includes both memory and logic so that it can execute operation directly within itself. By utilizing wide-internal memory bandwidth and cutting down external memory bandwidth, PIM can mitigate memory-wall problem leads to system performance improvement, against limited-external memory bandwidth [1].

As PIM architecture and application are not generalized at the moment, it is important to explore PIM structure and find out appropriate applications for PIM. In this paper, we select some applications which are broadly used, and implement PIM logics for these applications. In experiment, we observe their performance within PIM, and figure out appropriate applications and PIM logic structure for these applications.

II. BACKGROUD & MOTICATION

There have been many studies about 3-Dimensional integrate circuit to mitigate memory-wall problem, such as through silicon via (TSV). With 3-D integration, it has become possible to integrate wafers manufactured in different process. For example, we can integrate DRAM and processor in a single chip. By combining DRAM and Processor together, memory-wall problem can be solved in addition to release limited off-chip pin count, and long wire latency.

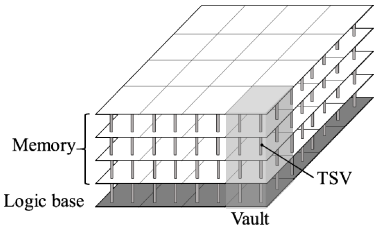
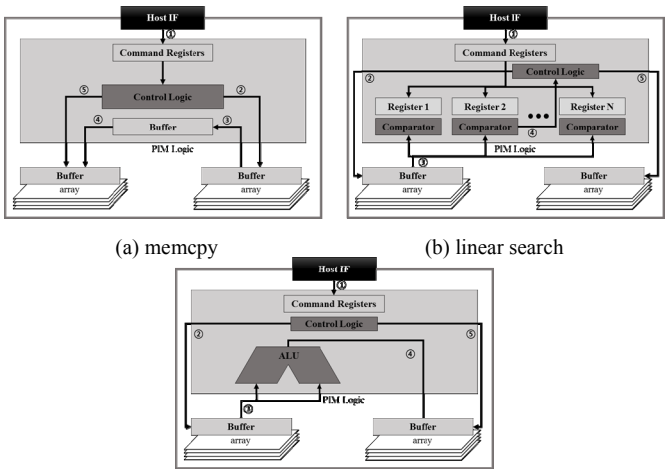


Fig. 1 Architecture of Hybrid Memory Cube (HMC)

High bandwidth memory (HBM) and hybrid memory cube (HMC) are representative examples using 3-D integration. [3] HMC consists of several memory dies and a logic die: memory is organized into vaults vertically. Each vault includes vault controller in logic die and operates independently. By exploiting redundant area in logic die, additional logic can be implemented without incurring area overhead

III. IMPLEMENT

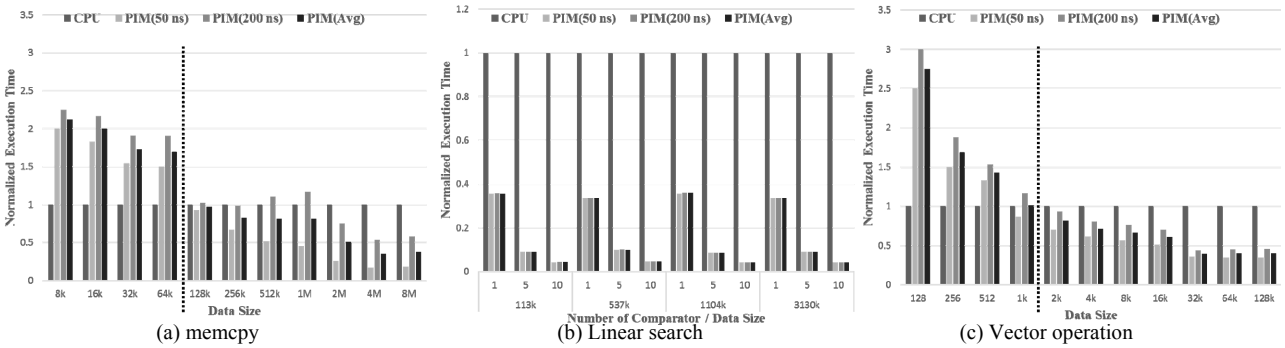
We select 3 broadly used applications: memcpy, linear search, and vector operations. We implement these 3 applications in logic layer.



(a) memcpy (b) linear search

(c) Vector operation

Fig. 2 Architecture of Hybrid Memory Cube (HMC)



(a) memcpy (b) Linear search (c) Vector operation

Fig. 3 Normalized PIM Function Result

A. *memcpy*

Figure 2(a) is PIM architecture of memcpy function logic that copies data of src to dst. With CPU, all of data passes through external bus, wasting memory bandwidth. With PIM, however, memcpy operation is executed using wide-internal bandwidth so that improves system performance.

Logic operates: 1) memcpy instruction is fetched from host(processor), 2) load all of data from src and 3) store in row buffer, 4) transfer data to memory array of dst, 5) save the data to dst memory.

B. *linear search*

Figure 2(b) is PIM architecture of linear search function logic which searches specific value from data set. With CPU, entire data set should be fetched to cache, wasting memory bandwidth. With PIM, fast memory access, and optimization of linear search architecture will improve operation speed.

Logic operates: 1) linear search instruction is fetched from host, 2) load data set into buffer. 3) Compare logic will compare data set and value which is stored in a register. 4) When compare logic find value, it will send value location of information to control logic. 5) finally control logic will store the address of found data.

C. *Vector operation*

Figure 2(c) is PIM architecture of vector operation function which execute arithmetic calculation with arrays. With placing logic close to memory array, this architecture can reduce read/write access latency and execute rapidly.

Logic operates: 1) host fetches vector operation instruction to this logic, 2) data of arrays are loaded to buffer. 3) data fetched through ALU serially, and 4) ALU calculates the proper operation, 5) calculation results are written to memory.

IV. EVALUATION

To verify functional operation and performance, we simulate a full-system equipped with ARM v7 2GHz, System clock 1GHz, 32kB L1, 8MB L2 cache using gem5 simulator. We add logic layer to enable directly access main memory through internal path. Our configuration assumes that PIM logic can access a whole memory row in a short time by utilizing large internal data bus. Also, we experiment with various memory latency as memory access latency vary depending on memory architecture and manufacturing process. Figure 3 shows the execution time for different memory access latency.

In case of memcpy instruction with small data, advantage of PIM is not much impressive because of CPU cache effect. If data size is larger than CPU cache size (128KB in this case), PIM performance start to outstrip CPU performance. With 8MB data, performance is improved 321%.

Linear search application is examined searching a specific word from various sizes of texts. Result shows that performance improvement is caused by low memory access latency and parallel execution. With the data set which size is 3130KB, PIM achieved 296%, 1117%, 2370% of speed up for searching for 1, 5, 10 words respectively.

In case of vector operation, we execute $\overline{\mathbf{F}} = (\overline{\mathbf{A}} + \overline{\mathbf{B}} - \overline{\mathbf{C}}) \div \overline{\mathbf{D}} \times \overline{\mathbf{E}}$. ALU operating cycle is assumed as 3 cycles for division, and 1 cycle for other arithmetic calculations. If Vector size is larger than 1KB, performance of PIM is better than that of CPU, resulting 226% performance improvement when data size is 128KB. In our experiment, we use only one ALU, a lot of ALUs can intensify performance improvement.

V. CONCLUSION

In this paper, some broadly used applications are implemented and verified by simulator. As a result, applications which have simple memory access pattern and negligible hardware logic overhead gain more performance benefit with PIM. PIM can contribute to performance improvement to overall system by mitigating memory-wall problem, through utilizing internal memory bandwidth.

ACKNOWLEDGMENT

This work was supported by the ICT R&D program of MSIP/IITP. [2016(R7177-16-0233), Development of Application Program Optimization Tools for High Performance Computing Systems] and by Samsung Electronics.

REFERENCES

[1] Ahn, Junwhan, et al. "A scalable processing-in-memory accelerator for parallel graph processing." Proceedings of the 42nd Annual International Symposium on Computer Architecture. ACM, 2015.

[2] Loh, Gabriel H., et al. "A processing in memory taxonomy and a case for studying fixed-function pim." Workshop on Near-Data Processing (WoNDP). 2013.

[3] "Hybrid memory cube specification 2.0," Hybrid Memory Cube Consortium.