A Study of Data Layout in Multi-channel Processing-In-Memory Architecture

Taeyang Jeong
Dept. of Electrical and Electronic Eng.,
Yonsei University
Seoul, Republic of Korea
+822-2123-7826
drthvbfg@yonsei.ac.kr

Duheon Choi
Dept. of Electrical and Electronic Eng.,
Yonsei University
Seoul, Republic of Korea
+822-2123-7826
cdh0527@yonsei.ac.kr

Sangwoo Han
Dept. of Electrical and Electronic Eng.,
Yonsei University
Seoul, Republic of Korea
+822-2123-7826
swhan0330@yonsei.ac.kr

Eui-Young Chung
Dept. of Electrical and Electronic Eng.,
Yonsei University
Seoul, Republic of Korea
+822-2123-7826
eychung@yonsei.ac.kr

ABSTRACT

In modern computing hardware, the performance gap between processor and memory is one of the most significant factors that limits overall performance improvement of computing system. Also, with the advent of multicore and manycore system, memory bandwidth per core is decreasing constantly. To solve this problem, recently, many researchers are interested in Processing-In-Memory (PIM). PIM is that processing elements are attached to memory-side, so near-memory-processing which is suitable for memory intensive application can be possible. Various researches studied PIM, but it was just single-channel memory system. In addition, PIM is a new architecture that is different with conventional computing system. Thus, common data layout cannot become optimal case for PIM. Optimal data layout is also needed to be studied.

In this paper, we propose the multi-channel PIM architecture with PIM-to-PIM communication, because data that is needed to operate can be distributed over several channels. To utilize multi-channel PIM architecture properly, we also introduce data layout that can minimize the number of PIM-to-PIM communications which are overheads of the system and maximize parallelism to reduce execution time. We evaluate it about vector arithmetic operation. The result is that execution time is improved about 393% and compared to the worst case, in the optimal data layout.

CCS Concepts

• **Hardware→Memory and dense storage**

Keywords

Processing-In-Memory; data layout; multi-channel memory architecture.

ICSCA 2018, February 8–10, 2018, Kuantan, Malaysia

ACM ISBN 978-1-4503-5414-1/18/02…$15.00
https://doi.org/10.1145/3185089.3185136

1. INTRODUCTION

Until the middle of 1990s, the most important research field of computer system was a processor. Because memory could offer sufficient memory bandwidth and latency, the performance of computer system was determined by clock frequency and cycle-per-instruction of processor. However, over the last several decades, the performance gap between processor and memory has gradually eliminated and even reversed. Due to this problem, the cases that memory cannot meet bandwidth and latency demands of host processor are increased. Also, the appearance of multicore and manycore system makes computing power better by utilizing parallelism, but allocated memory bandwidth per core is decreased. This problem is called memory wall [1] that memory becomes dominant part to decide overall system performance even if computing power of processor is enough to meet needs of applications.

In recent years, the demand of heavy workload applications, like neural networks and deep learning, that request a lot of memory accesses is grown quickly. Thus, memory wall problem becomes major issue on computer system. To solve it, researchers propose various methods to improve memory performance. PIM is one of them. PIM adds computing power to memory, so it can be a good solution to decrease the data movements between host processor and memory and reduce the memory latency by eliminating off-chip data transfers. PIM was studied in many ways [2] [3] [4] [5] a few decades ago, but it had technological limits, like power, area, and thermal issues.

Emerging 3D-stacked DRAM with through silicon via (TSV), like hybrid memory cube (HMC) and high bandwidth memory (HBM), gives PIM the potential that was proposed decades ago. 3D-stacked DRAM re-architects the DRAM to improve much better timing and energy efficiency and reduce area much smaller. The researches of new memory devices are also utilized to make PIM architecture. Thus, it becomes possible to implement PIM nowadays.

Recently, many researches discuss PIM with 3D-stacked DRAM [6] [7] [8] [9] or new memory devices, but they almost assume

single-channel PIM architecture. Today, processor for high performance computing server or even desktop support dual-channel or quad-channel memory. These researches cannot reflect the latest computing system. Thus, it is needed to study multi-channel PIM architecture. Furthermore, by attaching processing element to memory, the paths of data movement are added and the masters who request memory access are increased. Therefore, the study of optimized data layout to utilize properly multi-channel PIM architecture is required.

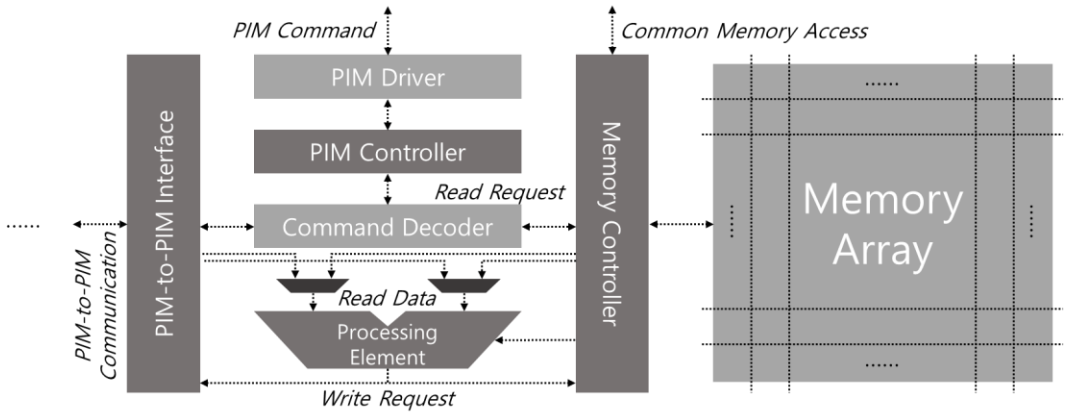

Figure 1. Multi-Channel PIM architecture

In this paper, first, we propose the multi-channel PIM architecture. We put processing elements on each channel and they are operated independently. By doing this, parallel computing can be done by each channel. Also, we implement PIM-to-PIM (channel-to-channel) communication which makes direct communication on each PIM without going through host processor or memory controller, because the data which is necessary to operate can be distributed by channel interleaving. Thus, to reduce the data transfer overhead of host-memory or memory controller-memory, direct communication method is required.

Second, we consider optimal data layout to minimize PIM-to-PIM communication and utilize parallelism on our proposed multi-channel PIM architecture about vector arithmetic operation. Operating system allocates data by page granularity and with interleaving policy. It can be allocated on only one channel or several channels. Thus, interleaving policy is important factor to determine data layout. In addition, optimal data layout can be different depending on application characteristics. With considering PIM-to-PIM communication, interleaving policy, and application characteristics, we analyze four cases of data layout.

2. ARCHITECTURE AND MODEL

In this chapter, we introduce our proposed multi-channel PIM architecture and analyze four cases of data layout with assuming quad-channel memory and with pipelining sequences.

2.1 Multi-Channel PIM Architecture

In Figure 1, it shows our proposed multi-channel PIM architecture. There is processing element to handle PIM command and PIM-to-PIM interface to exchange data with another channel directly. This architecture is implemented for each channel. We will explain each module with exploring data flow.

From the host processor, commands that are needed to operate on PIM are offloaded to memory-side. Offloaded commands are transferred to free channel. PIM driver which is in kernel is software stack of PIM architecture. Application calls PIM command then, PIM driver translates virtual address of source and destination data to physical address and sends command to PIM controller. If the data is common memory access, it is transferred to memory controller. PIM controller splits serialized command data to feasible command and schedules them to retain consistency and for optimal computation. Command decoder decodes PIM command to get opcode, source data address and destination data address and sends read request with source data address to memory controller. If memory controller finds source data in local memory, it transfers data to processing element. Otherwise, it sends read fail response to command decoder. Then, command decoder requests PIM-to-PIM communication to PIM-to-PIM interface to read data from other channel directly without interference of host processor. PIM-to-PIM interface broadcasts read request to another channel. When data is come, it transfers data to processing element. Processing element processes PIM operation with each input and sends write request to memory controller and memory controller handles it. If there is no destination data in local memory, memory controller sends write fail response to processing element. Then, processing element re-sends write request to PIM-to-PIM interface. PIM-to-PIM interface broadcasts it and transfers data to right channel.

PIM-to-PIM communication occurs in three cases, when reading data from and writing data to another channel and when PIM driver doing address translation. In case of address translation, page global directory (pgd) may not exist in local memory. Thus, if pgd does not exist on local channel, PIM in local channel should request pgd to another channel who has it.

We use single-instruction-multiple-data (SIMD) processor, because it can process many data which are aligned contiguously with low frequency, power and area by utilizing parallelism. It is also specialized to treat vector operation which does same operation on sequential data simultaneously. Therefore, optimal data layout to utilize computing power of SIMD processors is needed.

In this architecture, memory controller which is attached on processor-side conventionally is moved to memory-side, because it makes processing element access memory simple. Otherwise, data movements which are critical overheads are required to access memory from processing element. Such an architecture is implemented in HMC.

2.2 Case Study of Data Layout

Based on above multi-channel PIM architecture of chapter 2.1,

we assume quad-channel memory and analyze data layout for vector arithmetic operation to minimize PIM-to-PIM communication and utilize parallelism to improve performance.

When performing page allocation on the memory, the data layout is determined depending on the channel interleaving policy. If channel interleaving is not supported, one page is allocated to one channel, but if channel interleaving is supported, one page is allocated to multiple channels. Figure 2 shows four cases of data layout of vector operation depending on page allocation in quad-channel memory and shows pipelining sequences of each case. Figure 2-(a) shows the data layout where both source and destination data pages are allocated to one channel. Figure 2-(b) shows that source data pages are allocated to same channel, but destination data page is allocated to different channel. Figure 2-(c) shows the data layout in which all data pages are allocated to different channels. In Figure 2-(d), all the data pages are evenly distributed across all channels.

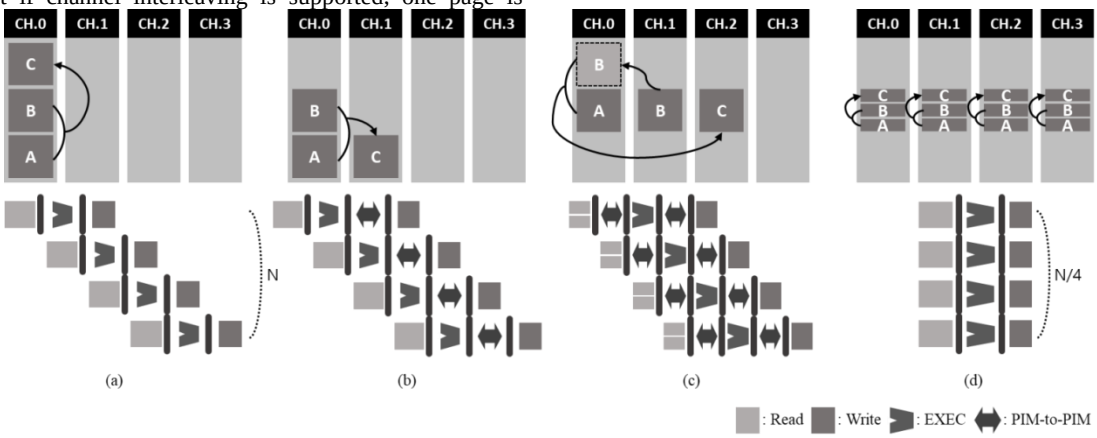

Figure 2. The Cases of Data Layout and Pipelining Sequences

First, in the case of Figure 2-(a), all data pages are allocated to the same channel. Thus, the pipelining sequence is that processing element reads two source data from the local memory, then performs the PIM operation, and writes results to the destination data of the local memory. This data layout is similar with single-channel PIM architecture. In this case, PIM-to-PIM communications are not needed, but address translation may be needed, Parallel computing is not performed either, because all processes are done in local memory.

In Figure 2-(b), two source data are allocated to the same channel, but destination data is allocated to another channel. Thus, PIM-to-PIM communication between source data channel and destination data channel is necessary. In pipelining sequence of Figure 2-(b), PIM-to-PIM communication is added between EXEC and Write in the pipelining sequence of Figure 2-(a).

In Figure 2-(c), the channel interleaving policy is not applied, and all data pages are allocated to different channels. Therefore, fetching the source data from the other channel through PIM-to-PIM communication is additionally needed in Figure 2-(b). We can see that this case is where the number of PIM-to-PIM communication is the greatest. In Figure 3-(c), the PIM-to-PIM communication overhead occupies a large part of the entire pipelining sequence. The interesting feature of this layout is that reading source data at the channel 0 and reading source data at the channel 1 to send them to channel 0 by PIM-to-PIM communication can be processed at the same time.

In Figure 2-(d), the channel interleaving policy is applied, and one page is split into four channels. If the indexes of the distributed source data and destination data is same, it is possible to perform operations in parallel on each channel without data read/write PIM-to-PIM communication. Therefore, it is expected that the number of pipelining sequence repetition is reduced to one-fourth as compared with Figure 2-(a). However, it needs address translation in every channel. Thus, a lot of address translation PIM-to-PIM communication overheads are expected.

3. SIMULATION ENVIRONMENT

We have experimented with gem5 full system simulator in which multi-channel PIM architecture is applied. The gem5 is cycle-accurate simulator, so it simulates micro-architecture cycle-by-cycle. To implement PIM architecture, we customize memory parts of gem5. We add processing element and implement PIM-to-PIM communication in the gem5.

Table 1 shows processor configuration that we use on gem5 simulator. ARM processor is used to host processor, because in case of x86 processor, when implementing PIM architecture on real system, customizing hardware architecture is difficult since -

Table 1. Processor Configuration

Parameter	Value
Core	ARM Cortex-A15
Frequency	1.0 GHz
# of Cores	1 – 4
L1 Caches	32KB Dcache, 32KB Icache, 4-set associatives.
L2 Caches	512KB, 8-set associatives
Etc	Out-of-Order Pipeline, Dynamic branch prediction, BTB, GHB

Table 2. Memory Configuration

Parameter	Value
Memory Type	HMC 1.0 [10]0
Memory Size	4GB
Bandwidth	160GB/s
Latency (tRAS)	21.6ns
# of Channels	4
# of Layers	4
# of Banks	2
Row Buffer Size	256B
Etc	32 TSVs, close page policy, 64 write buffers, 32 read buffers

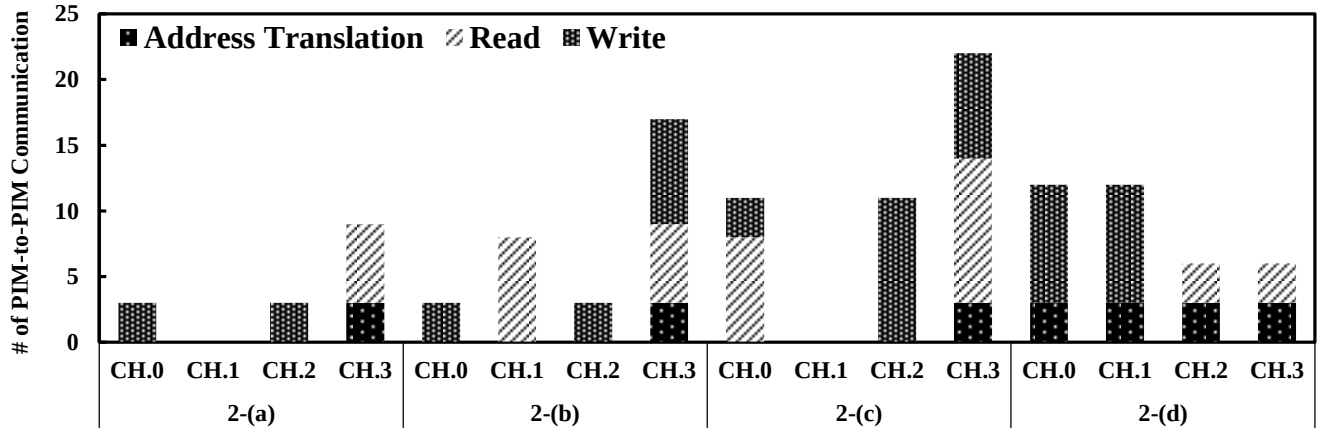

Figure 3. The Number of PIM-to-PIM Communication on Each Case

- already most of x86 processors have memory controller in on-chip, while ARM processor is fully customizable.

Table 2 shows memory configuration. We utilize HMC memory to apply PIM, because HMC has logic layer on bottom of 3D-stacked layers. This logic layer has user customizable area and its own atomic data operations. Thus, HMC is appropriate to apply PIM architecture. In addition, HMC has high memory bandwidth and low internal latency.

To support PIM hardware, we make PIM driver on kernel. we set some parts of kernel address ranges as PIM command transaction. Address translation of virtual addresses of source and destination data is done by PIM driver. By using PIM driver, host processor can offload operations to PIM through the kernel and PIM hardware can access memory by translated physical address.

4. EVALUATION

Based on simulation environment that sets on chapter 3, we compare our proposed multi-channel PIM architecture with single-channel PIM architecture. Also, we evaluate PIM-to-PIM communication overhead and execution time of vector operation on each case of data layout.

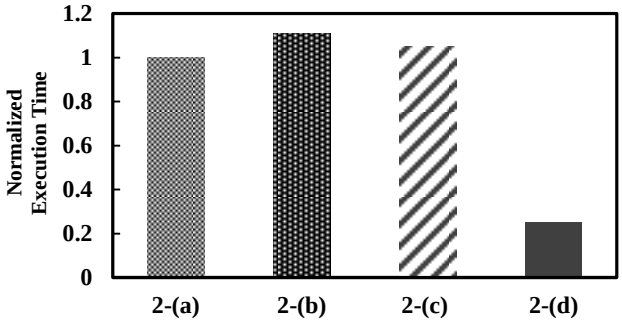

Figure 4. Normalized Execution Time on Each Case

Figure 3 shows the number of PIM-to-PIM communication case by case. The name of each case is taken from Figure 2. As we expected in chapter 2.2, the case, 2-(a), has the lowest PIM-to-PIM communication, totally 15 times, because it has essential data in its local memory and only address translation needs PIM-to-PIM communication to get pgd. 2-(b) and 2-(c) have several PIM-to-PIM communications. Except for address translation, because the requested data are in the different channel, data read and write operations are required through PIM-to-PIM communication. Each total number of communication is 31 and 44. In 2-(d), address translations are dominant factors of PIM-to-PIM communication, because it utilizes all processing elements of all channels. Thus, every channel performs address translations. it needs 4 times more than others. Total number of communication is 36.

Figure 4 represents normalized execution time on each case. We normalize execution time based on case 2-(a). As a result, we demonstrate that case 2-(d), which all data are distributed on each channel with channel interleaving policy, has the lowest execution time. It speedups about 393.5% faster than basis, 2-(a). Although 2-(d) has more number of PIM-to-PIM communication than 2-(a), because of parallelism, execution time is reduced. 2-(b) and 2-(c) are slower than 2-(a) due to PIM-to-PIM communication overheads.

But, it is primitive result, because cycle and delay model of PIM in gem5 is set heuristically. Also, it only compares the computation part of vector arithmetic operation code. Thus, our future works will be setting realistic parameters on gem5 and optimization in the entire application.

5. CONCLUSION

Performance gap between processor and memory is emerging as critical issues in modern computing systems. To solve this problem, several researches on PIM are performed. In this paper, we introduce PIM architecture in multi-channel memory and support PIM-to-PIM communication. This is to enable the data to be directly read without the host system and the memory controller when the data required for the operation exists in another channel. However, this is an overhead to overall system performance. Therefore, we have searched for data layout that minimizes PIM-to-PIM communication overhead and enhances performance through parallelism.

We observed a 393% faster data layout versus a bad case data layout for vector operations through high parallelism and shows for a data layout with minimal number of PIM-to-PIM communication.

6. ACKNOWLEDGEMENTS

This work was supported by Samsung Electronics and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2016R1A2B4011799).

7. REFERENCES

[1] Wm. A. Wulf, Sally A. McKee. 1995. Hitting the memory wall: implications of the obvious. *ACM SIGARCH Computer*

Architecture News. Volume 23 Issue 1. (March. 1995). pp 20-24.

[2] M. Gokhale, B. Holmes, and K. Iobst. 1995. Processing in memory: the terasys massively parallel pim array. *Computer, vol. 28. no 4*. pp 23–31.

[3] M. Oskin, F. T. Chong, and T. Sherwood. 1998. Active pages: A computation model for intelligent memory. *International Symposium of Computer Architecture*. pp. 192–203.

[4] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. 1997. A case for intelligent ram. *IEEE International Symposium on Micro Architecture, Volume 17, no 2*, pp 34–44, Mar 1997.

[5] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas. 2012. Flexram: Toward an advanced intelligent memory system. *IEEE International Conference on Computer Design*. pp. 5–14.

[6] Akin, B., Franchetti, F., & Hoe, J. C. 2015, June. Data reorganization in memory using 3D-stacked DRAM. *ACM SIGARCH Computer Architecture News. Vol. 43, No. 3*, pp. 131-143.

[7] Ahn, J., Hong, S., Yoo, S., Mutlu, O., & Choi, K. 2015, June. A scalable processing-in-memory accelerator for parallel graph processing., *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA* . pp. 105-117.

[8] Kim, D., Kung, J., Chai, S., Yalamanchili, S., & Mukhopadhyay, S. 2016, June. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (pp. 380-392).

[9] Nair, R., Antao, S. F., Bertolli, C., Bose, P., Brunheroto, J. R., Chen, T., ... & Fleischer, B. M. 2015. Active memory cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development, 59(2/3)*, 17-1.