Processing In Memory

Dual Degree Stage 1 Report

Submitted in partial fulfillment of the requirements

for the

Dual Degree Programme

by

Nayan Barhate

(Roll No. 180070037)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering
Indian Institute of Technology Bombay

October 2022

Acknowledgement

I express my gratitude to my guide Prof. Virendra Singh for providing me the opportunity to work on this topic.

Nayan Barhate
Electrical Engineering
IIT Bombay

Abstract

Processing-in-memory (PIM) is rapidly rising as a viable solution for the memory wall crisis, rebounding from its unsuccessful attempts in 1990s due to practicality concerns, which are alleviated with recent advances in 3D stacking technologies. However, it is still challenging to integrate the PIM architectures with existing systems in a seamless manner due to two common characteristics: unconventional programming models for in-memory computation units and lack of ability to utilize large on-chip caches.

Contents

List of Figures

Chapter 1

Introduction

All computing systems, including cloud and server platforms, desktop computers, mobile and embedded devices, and sensors, rely heavily on main memory. It is one of the primary pillars of any computing platform, along with 1) processing elements (or computational elements), which can include CPU cores, GPU cores, accelerators, or reconfigurable devices, and 2) communication elements, which can include interconnects, network interfaces, and network processing units.

Processing in Memory takes advantage of the ability to implement a wide range of general-purpose processing logic in the logic layer of 3D-stacked memory, as well as the high internal bandwidth and low latency available between the logic layer and the memory layers. This is a more general approach in which the logic implemented in the logic layer can be general purpose, benefiting a wide range of applications. Multiple layers of memory (typically DRAM in pre-existing systems) are stacked on top of each other in a 3D-stacked memory. Vertical through silicon vias connect these layers together (TSVs). Thousands of TSVs can be placed within a single 3D-stacked memory chip using current manufacturing process technologies.

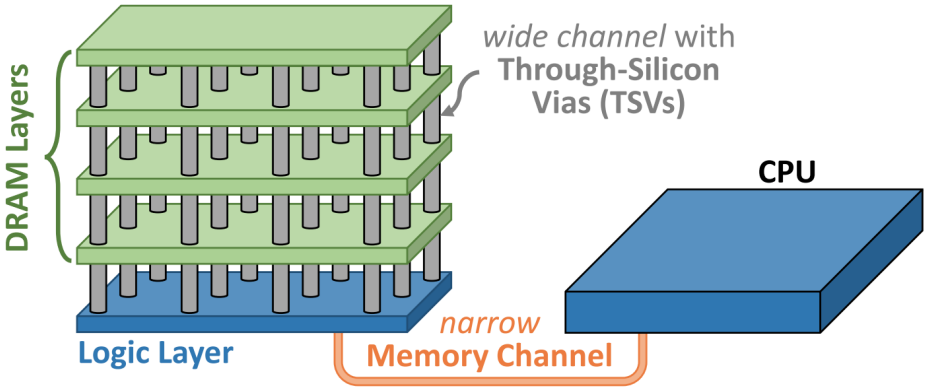

Figure 1.1: High-level overview of a 3D-stacked DRAM architecture[1]

Chapter 2

Literature Survey

Processing in memory (PIM) involves adding or integrating PIM logic (e.g., accelerators, simple processing cores, reconfigurable logic) close to or inside the memory. Many of these works place PIM logic inside the logic layer of 3D-stacked memories (eg. Hybrid Memory Cube or HMC) This PIM processing logic, which we also refer to as PIM cores or PIM engines, interchangeably, can execute portions of applications (from individual instructions to functions) or entire threads and applications, depending on the design of the architecture. Such PIM engines have high-bandwidth and low-latency access to the memory stacks that are on top of them, since the logic layer and the memory layers are connected via high-bandwidth vertical connections, e.g., through-silicon vias. Systems make use of relatively simple PIM engines within the logic layer to avoid data movement and thus obtain significant performance and energy improvements on a wide variety of application domains.

2.1 Coarse-Grained Application-Level PIM offload

In coarse grained application PIM offload an entire application is re-written to completely execute on the PIM substrate. In such a coarse-grained (i.e., application-granularity) approach, specialised programming model and specialised architecture/hardware are required. Large-scale graph processing is a popular application coarse grained PIM offload. Because of (1) large amounts of random memory accesses across large memory regions and (2) very small amounts of computation per data item fetched from memory. Thus graph analysis workloads are known to put significant strain on memory bandwidth.

Tesseract[2] is an example of a coarse-grained application level PIM accelerator. Tesseract consists of (1) a new hardware architecture that effectively uses the available memory bandwidth in 3D-stacked memory by placing simple in-order processing cores in the logic layer and allowing each core to manipulate data only on the memory partition it is assigned to control, and (2) an efficient method of communication between different in-order cores within a 3D-stacked memory that allows each core to request computation on data elements that reside in the memory partition it is assigned to control.

Rather than moving data elements across different memory partitions and cores, the Tesseract design moves functions (i.e., computations and temporary values) to data that is to be updated. It also includes two hardware prefetchers specialised for graph processing memory access patterns, which operate based on hints provided by programming model. Tesseract PIM architecture improves average system performance by 13.8x and achieves an 87% average energy reduction over conventional systems.

Application can also be offloaded by considering qualitative benefits of PIM such as parralelizability[3], reduced memory latency[4], bandwidth savings[5], energy efficiency[6],

thermal constraints[7], and workload characteristics[8].

2.2 Fine-Grained Instruction-Level PIM offload

With Fine grained approach, individual instructions can be offloaded to the PIM engine and accelerated. This fine-grained approach can have significant benefits in terms of potential adoption since existing processor-centric execution models already operate (i.e., perform computation) at the granularity of individual instructions and all such machinery can be reused to aid offloading to be as seamless as possible with existing programming models and system mechanisms.

PIM-Enabled Instructions (PEI)[9] aims to provide the minimal processing-in-memory support to take advantage of PIM using 3D-stacked memory, in a way that can achieve significant performance and energy benefits without changing the computing system significantly. PEI incorporated a locality-aware mechanism which tracked the locality of operand and dynamically decides where to execute the instruction. Examples of PEIs are integer increment, integer minimum, floating-point addition, hash table probing, histogram bin index, Euclidean distance, and dot product. The instructions were tailored to accelarated the application but were distinct from the host ISA. This affects significantly in the adoption of PIM for general purpose processing. Data locality is a great parameter for deciding the location of execution (host CPU or memory), but is not sufficient. In case the instruction was executed on the host CPU and one operand from memory, the feteched operand from memory may not be referenced again and pollutes the cache with unnecessary data movement.

2.2.1 Effects of Instruction-level offload

In case of PEIs the decision of fetching an operand from memory can result in pollution of cache. Thus we propose to take decide the location of execution by considering the affter-effects of fetching operand from memory / flushing the cacheline from host. Thus we predict the reuse distance of the cacheline in which the operand resides.

Chapter 3

Project Proposal

Instruction granularity does not affect the programmers model, and thus helps in the adoption of PIM for general purpose processing. We thus intend to accelarate X86 SIMD instructions which benefits from execution in PIM engines. For every operand that needs to be fetched from memory we predict its reuse distance. This prediction is very similar to mockingjay's[10] Reuse Distance Predictor with the difference that the prediction is done for operands of potential PIM instruction instead of incoming cacheline. If the predicted reuse distance is greater than any of the reuse distance stored in our predictor, we mark the instruction an offloadable instruction as fetching the operand from main memory results in pollution of cacheline and does not help the CPU for further operations.

References

[1] S. Ghose, A. Boroumand, J. Kim, J. Gómez-Luna, and O. Mutlu, "A workload and programming ease driven perspective of processing-in-memory," 07 2019.

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

[3] S. H. Pugsley, J. Jestes, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "Comparing implementations of near-data computing with in-memory mapreduce workloads," *IEEE Micro*, vol. 34, no. 4, pp. 44–52, 2014.

[4] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3d-stacked memory: Challenges, mechanisms, evaluation," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 25–32, 2016.

[5] K. Hsieh, E. Ebrahim, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 204–216, 2016.

[6] B.-H. Kim, E. C. Lim, and C. E. Rhee, "Exploration of a pim design configuration for energy-efficient task offloading," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2019.

[7] L. Nai, R. Hadidi, H. Xiao, H. Kim, J. Sim, and H. Kim, "Coolpim: Thermal-aware source throttling for efficient pim instruction offloading," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 680–689, 2018.

[8] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S.-K. Lim, and H. Kim, "Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 908–920, 2021.

[9] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," *SIGARCH Comput. Archit. News*, vol. 43, p. 336–348, jun 2015.

[10] I. Shah, A. Jain, and C. Lin, "Effective mimicry of belady's min policy," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 558–572, 2022.