Evaluating HPC Kernels for Processing in Memory

ABSTRACT

Memory sub-systems contribute significantly to HPC performance and energy efficiency. Traditional memory technologies with conventional organization (e.g., DRAM) are struggling to keep up with exceeding memory requirements of modern applications. Adopted techniques such as multi-layer cache hierarchy and out-of-order execution are still falling short to mitigate the penalty incurred by memory accesses. Processing-in-memory (PIM) is emerging as a promising technique, which suggests moving memory-intensive kernels to memory for execution, instead of bringing the data to the processing unit. This technique has recently received a lot of traction among computer architecture researchers and increasing research activity investigating this technique indicates its potential to alleviate main memory performance bottlenecks to a great extent. In this paper, we characterize and identify memory-intensive HPC kernels, perform a first-order evaluation of processing-in-memory technique for selected HPC kernels, quantify performance deviation and analyze the key factors that affect PIM efficiency.

CCS CONCEPTS

• **Computer systems organization** → *Processors and memory architectures*;

KEYWORDS

Processing in Memory, High Performance Computing

1 INTRODUCTION

Memory systems are dominant factors in large-scale High Performance Computing (HPC) clusters from performance and energy consumption perspective. It also significantly contributes to the setup and operational cost of such systems. Dynamic Random Access Memory or DRAM technology has been adopted and served as the primary building blocks of main memory sub-systems on the majority of computing domains including HPC. DRAM in its conventional organization (i.e., DIMMs) is struggling to accommodate the exceeding memory requirements of emerging applications from different domains. This is primarily because main memory systems and computing units are clearly separate devices and are usually located far apart and communicate via buses constrained by a limited number of pins. In addition, DRAM devices usually operate in much lower frequency than the compute units (CPU/GPU). As a result, accessing the main memory for reading and writing memory blocks are expensive operations in terms of latency and energy consumption. To give a perspective, accessing data from a DRAM device to the processing unit through the cache hierarchy takes about two orders of magnitude more energy than performing a floating point operation in a processor [1]. Over the years, several techniques have been adopted to complement the penalty occurring from memory operations. Several layers of cache hierarchy are introduced to reduce access to the main memory. Out-of-order cores are expected to continue execution while it's waiting for the data from the main memory. Despite the efforts, memory access overhead continues to sustain as a major bottleneck for efficient execution of applications in HPC and other domains.

Researchers are investigating alternative solutions to keep up with the increasing memory demands from modern applications. Part of that effort focuses on using novel memory technologies based on new materials such as Phase Change Memory (PCM), Spin-Transfer-Torque Magnetic RAM (STT-MRAM), Resistive RAM (ReRAM), etc. The idea is to leverage the unique properties of these memory technologies (non-volatility, endurance, no leakage current) to come up with an efficient alternative to DRAM technology. However, significant development at the cell/material level is warranted to get these memory technologies ready to attain DRAM-like performance. In addition, such extensive research and development efforts would also induce cost overhead for these technologies and it would be increasingly difficult to be commercially viable against very affordable DRAM technology. The other approach is to change the organization of conventional memory systems using the same DRAM technology. 3D stacking of DRAM dies appears to be a convenient way to increase capacity and moving the stack closer to the CPU on the same silicon interposer greatly helps to reduce the memory access latency. Silicon interposer also allows to have denser buses (e.g., 1024 bits) between memory stack and processing unit, effectively increasing the bandwidth. Currently, High bandwidth Memory (HBM) and Hybrid Memory Cube (HMC) are the two major protocols adopting that adopt 3D stacking DRAM model.

HMC [2] comes with an optional logic layer beneath the stack. This logic layer can be leveraged to implement simple processing units closer to the memory, which corresponds to the technique Processing-in-memory (PIM)[1], which builds on the idea of moving the execution of certain memory-intensive applications/kernels to the memory, instead of bringing the data to CPU for processing through expensive memory accesses. An upsurge of research activity investigating different PIM techniques indicates that it has significant potential to accelerate memory bound problems. Recent studies report significant improvements in performance and energy efficiency by adopting PIM techniques. However, these studies largely focus on and analyze the applications from Artificial Intelligence, Machine Learning, and Neural Network domains along with some generic kernels.

In this paper, we perform a preliminary evaluation of HPC kernels for Processing in Memory. We select HPC applications developed by the laboratories of US Department of Energy (DoE), identify their memory-intensive kernels and analyze their suitability and performance with processing-in-memory. We also analyze key factors affecting PIM efficiency.

The rest of the paper is organized as follows. Section 2 presents a background and related works on PIM techniques, Section 3 presents the workload characterization of target applications, Section 4 explains the experimental setup and methodologies used,

[1] Also known as Compute-in-Memory (CIM), processing-near-memory or processing-using-memory. For simplicity, we refer to it as Processing-in-memory (PIM) throughout the paper.

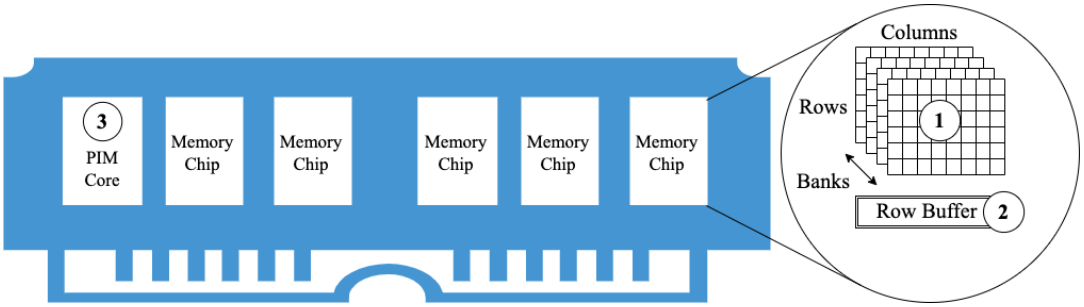


Figure 1: Generic representation of a Main Memory DIMM. Processing in memory can take place at the (1) memory array, (2) row buffer or in a simple processing unit (3) closer to the banks

Section 5 presents the results of the evaluation, and Section 6 summarizes the conclusions of the study.

2 BACKGROUND

Processing in memory generally refers to the idea of moving some of the executions to the memory unit, when moving the data to the CPU for processing deem to take more time, which is often the case for memory-intensive kernels. With various memory technologies and PIM optimization techniques present, there are several ways PIM can be adopted. Figure 1 shows a generic representation of a main memory DIMM module. A main memory DIMM (e.g., DDR4 DIMM) has several chips. Each chip has a number of banks (e.g., 4, 8, 16) and each bank has a specified number of rows and columns which constitutes the memory array (highlighted as (1) in Figure 1). Each intersection of a row and column typically represents a memory *cell* which holds single bit data. When a row address is requested to be accessed, that selected row is then loaded on to the row buffer (highlighted as (2) in Figure 1).

Several studies propose to use the memory array (1) for computing [3–8], mostly for matrix vector multiplication operations where the coefficients can be mapped to the rows and columns to compute dot products using the array. Baohua et al. [5] propose to improve in-memory multiplication with partition-based computation techniques for broadcasting and moving data within partitions. The authors exploit ReRAM's voltage controlled variable resistance to support logic gates. Long et al. [7] proposes another ReRAM based design particularly for Recurrent Neural Network (RNN) acceleration. The study proposes crossbar sub-arrays for matrix vector multiplication, multiplier sub-arrays for element-wise operations and special function units for nonlinear functions. Peng et al. [6] present a new mapping method and data flow to maximize reuse of weight and input data on a 8-bit ReRAM based PIM architecture. Imani et al. [4] report to have achieved graph processing and query processing capabilities along with machine learning acceleration with processing in memory.

The second category of PIM operations is proposed in the row buffer level (2); for example, by adding multiple row buffers and performing bit-wise operations among them. Lin et al. [9] present a PIM design proposing optimization at the row buffer level. The authors introduce a *Population Count Engine* which works on the data inside the sense amplifier and store the results back to the sense amplifier.

The third category of processing in memory suggests having a simple processing unit near the memory banks (marked as (3) in Figure 1) capable of executing a sub-set of CPU instructions. The main intention of this model is to reduce memory traffic to/from CPU [1, 10–20]. Some of these studies suggest using the logic die of 3D stacked memories to place Processing Elements (PE) directly beneath the DRAM dies with a shorter distance to the coefficient data residing in the DRAM layers for faster computation [1, 15, 16, 21]. Huang et al. [17] propose a heterogeneous PIM design incorporating memristors and CMOS based technologies, in order to accommodate the heterogeneity requirements of graph applications. DAMOV [20] is a PIM simulation infrastructure based on ZSim [22] and Ramulator [23] with offloading support. The authors develop a rigorous workload characterization methodology and quantify data movement bottleneck on an extensive set of applications and functions. Few studies evaluate PIM architectures with hardware implementation. Lee et al. [1] develop an industrial prototype of a PIM design that can seamlessly work with a variety of commercial processors without requiring any changes to the applications. The proposed PIM execution unit has a five-stage pipeline, 16-wide SIMD FPU, registers and controller. This design is based on an HBM stack fabricated with a 20nm technology process. PimCaffe [24] develops a PIM near memory design based on multiple FPGAs, implementing SIMD and systolic array compute engines for vector and matrix multiplications.

As a general observation, the first two approaches of PIM design (processing in memory array and row buffer) are inherently limited to certain compute operations. Since HPC application kernels perform complex computations that frequently includes floating point operations with various arithmetic functions, we believe the PIM execution model that would serve the best for HPC domain is the third approach explained above — to employ a simple processing unit where memory-intensive HPC kernels can be offloaded in its entirety to be executed. In this work, we evaluate HPC kernels on a simple processing unit closer to the memory as detailed in Section 4.

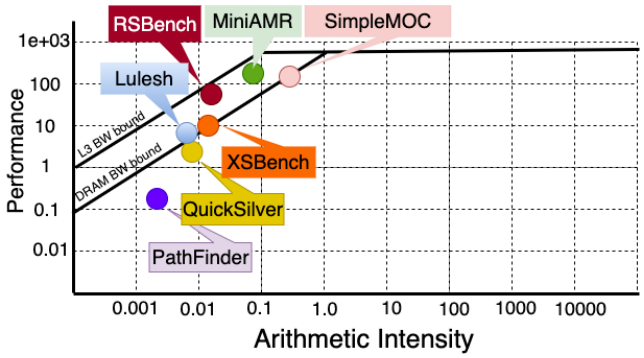


Figure 2: Approximate representation of HPC application kernels on the Roofline model.

3 WORKLOAD CHARACTERIZATION OF HPC APPLICATIONS

This section introduces the HPC application studied in this research and delves into characterizing them. The main source of these applications is the ongoing Exascale Proxy Application Project [25], commonly known as ECP proxy apps. Along with these proxy apps, Lulesh [26], another commonly used mini-app in the HPC community, is considered in this study. For characterization, at first, hotspot analysis is performed using Intel Vtune(2022.3.0) to identify the functions with high execution time. Then, Intel Advisor is used for generating Roofline models to identify the compute-memory intensity of those functions. Table 1 shows the application names along with the functions, and Figure 2 shows the position in the Roofline model. All the functions studied in this research are memory-intensive.

3.1 SimpleMOC

SimpleMOC mini-app is designed to demonstrate the performance and viability of the Method of Characteristics (MOC) for 3D neutron transport calculations for full-scale light water reactor simulation. In this application, *attenuate_fluxes* function is one of the significant contributors to execution time and is bound by DRAM Bandwidth (see Figure 2).

3.2 PathFinder

PathFinder is implemented for searching signatures in directed and cyclic graphs. It searches the path between signatures and labels in the graph. One of the main functions in PathFinder is *findAndRecordAllPaths*, which is DRAM bandwidth bound and has the lowest arithmetic intensity of all the functions studied in this research.

3.3 XSBench

XSBench represents a key computational kernel of the Monte Carlo neutron transport algorithm, which performs continuous energy macroscopic neutron cross-section lookup. The hotspot and Roofline analysis identify *calculate_micro_xs* as one of the major memory-bound functions.

Table 1: HPC Benchmarks and their Memory-Bound/Critical Functions identified

Bechmark	Function	Memory/compute bound
SimpleMOC	attenuate_fluxes	Main Memory BW bound
PathFinder	findAndRecordAllPaths	Main Memory BW bound
XSBench	calculate_micro_xs	Main Memory BW bound
RSBench	c_mul	L3 BW bound
MiniAMR	stencil_calc	L3 BW bound
QuickSilver	macroscopicCrossSection	Main Memory BW bound
Lulesh	CalcKinematicsForElems	Main Memory BW bound

3.4 RSBench

Similar to XSBench, RSBench also represents a kernel for Monte Carlo neutron transport algorithm. It represents the multipole method for performing continuous energy macroscopic neutron cross-section lookups. The hotspot and Roofline analysis reveals *c_mul* function as one of the major memory-bound functions.

3.5 MiniAMR

MiniAMR performs adaptive mesh refinement, which divides a unit cube computational domain and applies stencil calculation. The blocks have an equal number for cell distribution and communicate to their neighbors using ghost values. As expected, the *stencil_calc* function is the main memory-bound function for this application which is bound by level 3 cache bandwidth (see Figure 2).

3.6 Quicksilver

Quicksilver solves a simplified dynamic Monte Carlo particle transport problem to partially implement the Mercury workload. It replicates Mercury's memory access pattern, communication, and branching for problems using multi-group cross sections. The function *macroscopicCrossSection* is one of the hotspots, a memory-intensive kernel bound by DRAM bandwidth in the Roofline model.

3.7 Lulesh

Lulesh (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics) is a well-known HPC proxy app developed by Lawrence Livermore National Laboratory. This application exhibits different memory access patterns for a 3D mesh data structure where more than twenty functions are identified as memory bound [27], which makes it a feasible candidate for PIM study. In this study, we consider the function *CalcKinematicsForElems* for evaluation.

4 EXPERIMENTAL ENVIRONMENT

In this section, we present the simulation infrastructure and methodology used in the study as well as the metrics that are used to represent the results.

For our experiments, we use the recently released DAMOV-SIM simulation infrastructure [20]. DAMOV is based on ZSim system simulator [22] coupled with Ramulator memory simulator[23]. ZSim is used to simulate both host and PIM core's microarchitecture, cache hierarchy and coherence protocols. We use Hybrid Memory Cube (HMC) [2] to leverage its logic layer where PIM execution unit is realized. Ramulator memory simulator is used to model HMC

Table 2: Configurations used in the simulation

Unit	Configuration
Host CPU	1 out-of-order Core running at 2.4GHz,
L1 Cache	32KB, 8-way, 64B/Line, LRU policy
L2 Cache	256KB, 8-way, 64B/Line, LRU Policy
L3 Cache	8MB, 16-way, 64B/Line, LRU Policy
PIM Execution unit	1 out-of-order Core running at 2.4GHz,
L1 Cache	32KB, 8-way, 64B/Line, LRU policy
Main Memory	HMC v.2.0, 32 Vaults, 8 DRAM banks/vault, 256B row buffer, 8GB capacity.

main memory architecture, memory controller and main memory access protocols. The system configurations used in the simulations are summarized in Table 2. DAMOV-SIM uses pintool v. 2.14, on a Linux kernel v.3.8.

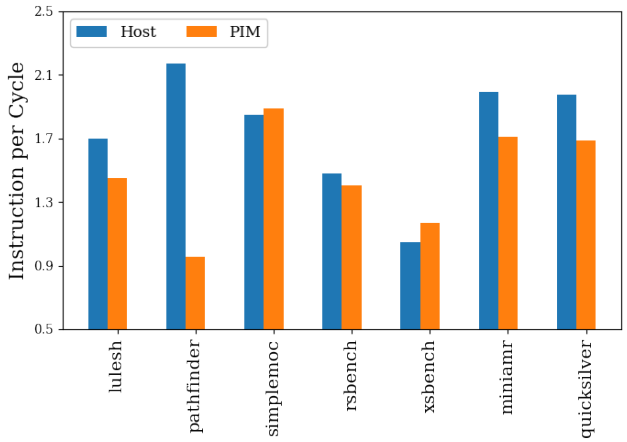
First, we identify the memory-intensive kernels of the benchmarks under study, as detailed in Section 3. We use DAMOV-SIM's offloading support to isolate identified kernels by inserting delimiter functions and run them for 1 billion instructions. In the configuration file, when `pimMode` is set to `true` Zsim simulates a PIM core with shorter latency to memory with high bandwidth. When `pimMode` is set to `false` it simulate a core with regular settings (host processor). At function-level offloading granularity, it is easier to determine and compare how the same function performs in PIM and host cores.

We analyze three important performance metrics: Instruction Per Cycle (IPC), Misses-Per-Kilo-Instruction (MPKI) and Last to First Miss Ratio (LFMR), which indicates the ratio of last level cache misses to first level cache misses.

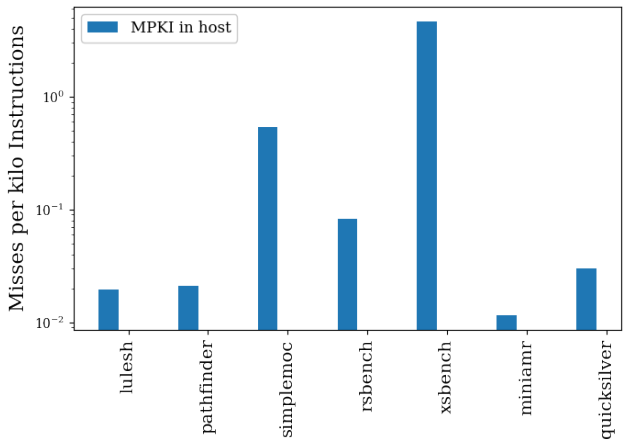
5 EVALUATION

In this section, we present the results from our simulations experimenting performance impact of executing memory-intensive kernels in the host processor (host) and PIM execution unit (PIM). Figure 3(a) shows the overall performance achievement of Host and PIM cores in terms of Instruction per Cycle (IPC) for all HPC kernels under study. The horizontal bars represent IPC counts for Host and PIM cores, whereas the *X* axis lists the benchmarks. The results show that two kernels from the application SimpleMOC and XSBench achieve slightly better performance (2.1% and 10.3% speed-up, respectively) when executed on the PIM core. Lulesh, RSBench, MiniAMR, and Quicksilver kernels' performance deteriorates by 17.2%, 5.2%, 16.2%, and 17.24%, respectively, in comparison to the host core. Pathfinder's kernels perform the worst, suffering a 126.9% degradation.

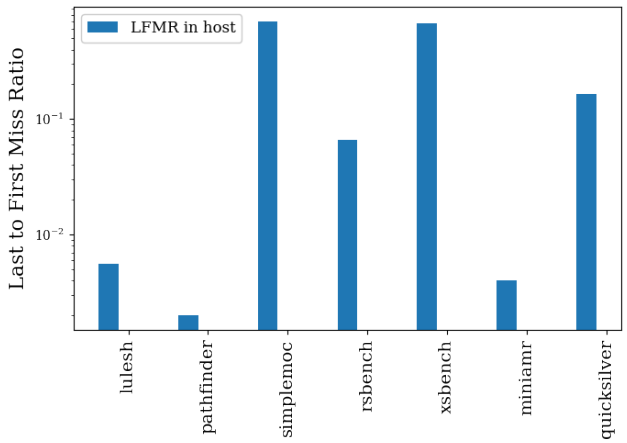
To investigate the variation in performance, we analyze L3 *Misses Per Kilo Instructions* (MPKI) for each kernel in the host core, presented in Figure 3(b), where *Y* axis lists the MPKI values in logarithmic scale, and *X* axis lists the name of the applications. The chart shows that the MPKI values for application kernels range from 0.011 (MiniAMR) to 4.65 (XSBench). SimpleMOC and XSBench, which perform better in the PIM core, have the highest MPKI values among the kernels under analysis. We further analyze *Last to First Miss Ratio (LFMR)*, which corresponds to the miss ratio between L3



(a) IPC.



(b) MPKI.



(c) LFMR.

Figure 3: Performance impact of HPC kernel executing on PIM core in comparison to the Host core. (a) Performance variation in terms of Instruction Per Cycle (IPC) for Host and PIM cores. (b) L3 Misses Per Kilo Instruction (MPKI) for all Kernels in host core (c) Last to First Miss Ratio (LFMR) on host core for kernels under analysis.

cache misses and L1 cache misses, presented in Figure 3(c). LFMR effectively indicates the efficiency of the cache hierarchy. For the HPC kernels under study, LFMR ranges from 0.002 (Pathfinder) to 0.7 (SimpleMOC). An interesting observation from this is that although miniAMR has lower MPKI (0.011) than Pathfinder (0.021), with higher LFMR, it performs relatively better on the PIM core.

6 CONCLUSIONS

In this paper, we perform a first-order evaluation of HPC kernels developed by laboratories of US Department of Energy (DoE) for processing-in-memory (PIM) technique. To that end, we characterize seven HPC applications and identify memory-intensive kernels which we run on the host core and PIM core with HMC main memory module using DAMOV-SIM simulation infrastructure. The results show that two of the seven HPC kernels achieve a performance gain executing in the PIM core, which is largely analogous to the L3 misses per kilo instructions (MPKI) of each kernel. The results also suggest that MPKI is not the only factor that determines if a kernel would benefit from executing on a PIM core. We observe that a kernel may perform relatively better on a PIM core than another kernel with a higher MPKI but having a lower LFMR.

REFERENCES

[1] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. Hardware architecture and software stack for pim based on commercial dram technology : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 43–56.

[2] Ramyad Hadidi, Bahar Asgari, Burhan Ahmad Mudassar, Saibal Mukhopadhyay, Sudhakar Yalamanchili, and Hyesoon Kim. Demystifying the characteristics of 3d-stacked memories: A case study for hybrid memory cube. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 66–75, 2017.

[3] Baohua Sun, Daniel Liu, Leo Yu, Jay Li, Helen Liu, Wenhan Zhang, and Terry Torng. Mram co-designed processing-in-memory cnn accelerator for mobile and iot applications.

[4] Mohsen Imani, Saransh Gupta, Yeseong Kim, Minxuan Zhou, and Tajana Rosing. Digitalpim: Digital-based processing in-memory for big data acceleration. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, page 429–434, New York, NY, USA. Association for Computing Machinery.

[5] Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky. Multpim: Fast stateful multiplication for processing-in-memory.

[6] Xiaochen Peng, Rui Liu, and Shimeng Yu. Optimizing weight mapping and data flow for convolutional neural networks on rram based processing-in-memory architecture. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

[7] Yun Long, Taesik Na, and Saibal Mukhopadhyay. Reram-based processing-in-memory architecture for recurrent neural network acceleration. 26(12):2781–2794.

[8] Anni Lu, Xiaochen Peng, Yandong Luo, Shanshi Huang, and Shimeng Yu. A run-time reconfigurable design of compute-in-memory–based hardware accelerator for deep learning inference. 26(6).

[9] Che-Chia Lin, Chao-Lin Lee, Jenq-Kuen Lee, Howard Wang, and Ming-Yu Hung. *Accelerate Binarized Neural Networks with Processing-in-Memory Enabled by RISC-V Custom Instructions.* Association for Computing Machinery.

[10] Ataberk Olgun, Juan Gómez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oğuz Ergin, and Onur Mutlu. Pidram: A holistic end-to-end fpga-based framework for processing-in-dram.

[11] Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, and Onur Mutlu. Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions.

[12] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture.

[13] Youngeun Kwon and Minsoo Rhu. Beyond the memory wall: A case for memory-centric hpc system for deep learning.

[14] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu. Processing-in-memory: A workload-driven perspective. 63(6):3:1–3:19.

[15] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 336–348.

[16] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu. Conda: Efficient cache coherence support for near-data accelerators. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 629–642, New York, NY, USA. Association for Computing Machinery.

[17] Yu Huang, Long Zheng, Pengcheng Yao, Jieshan Zhao, Xiaofei Liao, Hai Jin, and Jingling Xue. A heterogeneous pim hardware-software co-design for energy-efficient graph processing. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 684–695.

[18] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das. Scheduling techniques for gpu architectures with processing-in-memory capabilities. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 31–44.

[19] Jialiang Zhang, Yue Zha, Nicholas Beckwith, Bangya Liu, and Jing Li. Meg: A riscv-based system emulation infrastructure for near-data processing using fpgas and high-bandwidth memory. 13(4).

[20] Geraldo F. Oliveira, Juan Gómez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, and Onur Mutlu. Damov: A new methodology and benchmark suite for evaluating data movement bottlenecks, 2021.

[21] Naebeom Park, Sungju Ryu, Jaeha Kung, and Jae-Joon Kim. High-throughput near-memory processing on cnns with 3d hbm-like memory. 26(6).

[22] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. *SIGARCH Comput. Archit. News*, 41(3):475–486, jun 2013.

[23] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, 2016.

[24] Won Jeon, Jiwon Lee, Dongseok Kang, Hongju Kal, and Won Woo Ro. Pimcaffe: Functional evaluation of a machine learning framework for in-memory neural processing unit. 9:96629–96640.

[25] Ecp proxy application. https://proxyapps.exascaleproject.org/app/. Accessed: 2022-08-30.

[26] I. Karlin. Lulesh programming model and performance ports overview. Technical report, Lawrence Livermore National Lab.(LLNL), CA, United States), 2012.

[27] Mohammad Alaul Haque Monil, Seyong Lee, Jeffrey S Vetter, and Allen D Malony. Mapredict: Static analysis driven memory access prediction framework for modern cpus. In *International Conference on High Performance Computing*, pages 233–255. Springer, 2022.