Low-Precision SpMV and s-step SGD on Processing-in-Memory

Irene Simó Muñoz
Cornell University
isimo@cs.cornell.edu

***Abstract*—Emerging Processing-in-Memory (PIM) architectures offer order-of-magnitude improvements in memory bandwidth by placing compute directly inside DRAM, but their reliance on low-precision formats such as BF16 poses challenges for numerically sensitive scientific workloads. This work addresses two complementary aspects of this problem. First, we develop an efficient Sparse Matrix-Vector (SpMV) kernel for SK Hynix's Accelerator-in-Memory (AiM) and identify three microarchitectural modifications that remove key bottlenecks for sparse workloads. Second, we design a mixed-precision formulation of communication-avoiding s-step stochastic gradient descent (SGD) and study how reduced precision in the block Gram matrix affects convergence. Preliminary results show that our SpMV mapping achieves strong scaling speedups over both CPU and GPU baselines, while the s-step SGD profiles reveal precision can be reduced to improve throughput. Together, these efforts lay the groundwork for deploying robust, mixed-precision iterative scientific algorithms on PIM hardware.**

I. Introduction and Problem Statement

AI and ML workloads are shaping hardware design. Two of the most widely explored tendencies currently are placing compute next to memory (PIM) and trading numerical range for tensor-core throughput and wider SIMD lanes (adoption of the BF16 format), both throughput-oriented accelerator features. Even though the motivation of the hardware exploration lies in non scientific fields, many scientific computation applications share the same core kernels that define AI workloads, such as Sparse Matrix Vector multiplication (SpMV).

In this work, our overarching goal is to enable communication-avoiding iterative solvers to run efficiently on Processing-in-Memory (PIM) hardware. Achieving this requires addressing two key challenges: (1) developing efficient sparse linear algebra kernels—particularly sparse matrix–vector multiplication (SpMV)—that fully exploit the high memory bandwidth of PIM architectures, and (2) designing algorithms that can tolerate and remain numerically robust under low-precision arithmetic. PIM architectures address this bottleneck by placing compute units directly inside DRAM, increasing the available memory bandwidth.Our work consists of two complementary projects aligned with these challenges: first, mapping fundamental *sparse* primitives onto emerging PIM hardware to realize efficient execution; and second, designing the s-step stochastic gradient descent (SGD) algorithm from a mixed-precision perspective to ensure robustness under reduced precision.

A. Project 1: Sparse Matrix Vector Multiply on PIM

The memory-bound nature of general matrix-vector multiplication (GEMV), the dominant kernel in transformer inference, is the primary motivation for designing inference-specialized PIM architectures [1]. SK Hynix's AiM [2] is a hardware prototype of this class that has demonstrated speedup for GEMV over A100 GPUs [1]. As large language models (LLMs) have increased in scale, weight pruning has become a practical compression strategy; recent work has shown that accuracy is preserved at up to 50% sparsity [3]. This transforms the GEMV kernel into a SpMV kernel and its mapping onto PIM therefore directly relevant to LLM inference. Scientific computing presents an additional challenge: scientific data is naturally and irregularly sparse [4], making the mapping problem harder.

B. Project 2: Low-Precision s-step SGD

Here, we focus on s-step SGD as a representative scientific workload: it is iterative, involves SpMV as its dominant kernel, and has well-understood numerical sensitivity to precision loss. In distributed memory settings, the per-iteration synchronization cost of standard SGD dominates the execution time. Communication-avoiding s-step variants address this by unrolling s iterations into a single computational block, precomputing a basis of s search directions from one synchronization point, and deferring global communication until the block is complete [5]. The batching of s iterations introduces a precision challenge not present in single-step SGD: rounding errors compound across the block, and in low-precision hardware this accumulation can be severe enough to stall or destabilize convergence.

II. Preliminary Results

A. SpMV Kernel in AiM

AiM is a GDDR6-based PIM architecture in which each memory channel contains multiple DRAM banks,

each augmented with a lightweight processing unit capable of multiply-accumulate (MAC) operations. An inter-bank adder tree reduces partial results across banks within a channel, and a global buffer stages operands shared across banks. By executing arithmetic directly at the memory banks, AiM exploits the full internal DRAM bandwidth, which far exceeds the external bus bandwidth available to a conventional CPU or GPU. The 16 banks of a channel, however, share one command stream: on every cycle they must address the same column index within their respective rows. The cost of a 16-row slice therefore scales with its column union (the set of distinct columns any of its rows touches) rather than with its raw nonzero count, and pruning only pays off if surviving nonzeros co-locate into the same columns across the lockstep rows.

We co-design two components of a framework against this lockstep constraint. (i) A unified kernel, that handles SpMV and amortizes activation broadcast across cross-matrix batches (the attention triple W_Q, W_K, W_V and the MLP pair $W_{\text{gate}}, W_{\text{up}}$ share an input activation, so they share a column union). (ii) An architecture-aware pruning strategy that combines the RIA importance metric [6] with least-squares block reconstruction [3]; because LLM activations are heavily column-skewed, activation-aware scoring naturally drives surviving blocks into a small column union, satisfying the lockstep cost model without explicit hardware reasoning.

On a cycle-accurate AiMX simulator validated against SK Hynix hardware, we deliver a geomean $1.5\times$ per-layer speedup over dense AiM-GEMM and, for 7–13B models at $K{=}1$, sub-H100 per-layer latency at less than 20% of the H100 energy. Energy per generated token is 0.3–2.4 mJ across 7–110B models, roughly $5\times$ lower than the H100 cuBLAS/cuSPARSELt baseline. Perplexity on WikiText-2 stays within $1.2\times$ of dense on Mistral-7B (6.43 vs. 5.68) and within $1.5\times$ on LLaMA-2-13B (7.73 vs. 5.31) at 70% attention / 90% MLP density. The footprint reduction also pushes 7 of 11 evaluated $\geq$70B models across an AiMX-board boundary, eliminating up to 20% of the boards in a deployment and, in some cases, collapsing a multi-board configuration onto a single board.

B. Shared-Memory s-step SGD

The s-step formulation admits two implementation variants whose hot kernels differ: an *explicit* variant that forms the block Gram matrix $G = A_{\text{block}} A_{\text{block}}^{\top}$ via a SYRK call and is SYRK-bound, and a *matrix-free* variant that skips G and runs s GEMVs against A_{block} directly, trading $O(b^2 s^2 K)$ SYRK flops for $O(bsK)$ GEMV flops and becoming bandwidth-bound. Which variant wins depends on the (b, s) regime, not on precision.

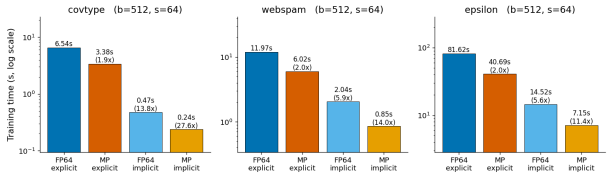

Fig. 1. Total execution time for 10k outer steps, batch size=512, step size=64. Each panel shows data for one dataset and contains four bars: FP64 explicit, MP explicit, FP64 implicit and MP implicit, with cumulative speedup annotated.

For the mixed-precision (MP) recipe A_{scaled} is stored in FP32, the SYRK and GEMV hot kernels run as SSYRK/SGEMV, while the iterate x and the weight-update DAXPY stay in FP64; one cast of x to FP32 per s-step block costs under 0.5% of total time. On AVX-512 this delivers a structural $2\times$ from hardware alone: twice the FP32 lanes per cycle for SSYRK and half the bytes streamed per element for SGEMV.

On an Intel Xeon W-1390P (8 cores, AVX-512, MKL 2023.2) over three LIBSVM benchmarks (covtype, webspam, epsilon) and a 9-point (b, s) grid, the MP/FP64 ratio holds at 1.9–$2.0\times$ inside each variant; the matrix-free/explicit ratio grows from near $1\times$ at small $b{\cdot}s$ to 5–$14\times$ at $b{=}512, s{=}64$; and the two effects compose to 11.4–$27.6\times$ end-to-end speedup of the matrix-free MP variant over the FP64 explicit baseline. The bottleneck shift is confirmed in the phase decomposition: the explicit variant spends 94–96% in SYRK, the matrix-free variant 81–88% in the GEMV recurrence at 80% of dual-channel DRAM peak. Held-out loss matches FP64 to 5–6 decimal places at every converged configuration.

III. Conclusion and Future Work

The two projects use the same methodology against different hardware constraints (lockstep PIM vs. wide-SIMD CPU) and different quality criteria (perplexity vs. held-out loss), and they share the same structure: a small set of independent slots, an additive or compositional quality model, and a discrete hardware assignment. They also share the same caveat that motivates the next step: the first is evaluated in cycle-accurate simulation against a single AiMX prototype, and the s-step CPU study is a single-workstation validation that the structural $2\times$ FP32 win composes with the matrix-free algorithmic win.

We will close the loop by applying the methodology jointly: a sparse mixed-precision iterative solver running natively on AiM-style PIM, in which the SpMV body exploits column-union scheduling, the s-step blocking structure amortizes per-iteration synchronization across host–PIM transfers, and the per-operation precision recipe extends to the additional PIM-side operations.

IV. Acknowledgements

The author would like to thank Prof. Giulia Guidi (PhD advisor), Cecilio C. Tamarit, Joseph Maheshe and Prof. José Martínez (PI) for their contributions to the AiM accelerator project, and Prof. Aditya Devarakonda (PI) for contributions to the s-step SGD project.

References

[1] Y. Kwon, K. Vladimir, N. Kim, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, G. Kim, B. An *et al.*, "System architecture and software stack for gddr6-aim," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE, 2022, pp. 1–25.

[2] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim *et al.*, "A 1ynm 1.25 v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 1–3.

[3] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International conference on machine learning*. PMLR, 2023, pp. 10 323–10 337.

[4] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *Acm transactions on mathematical software (toms)*, vol. 38, no. 1, pp. 1–25, 2011.

[5] A. Devarakonda and J. Demmel, "Avoiding communication in logistic regression," in *2020 IEEE 27th international conference on high performance computing, data, and analytics (HiPC)*. IEEE, 2020, pp. 91–100.

[6] Y. Zhang, H. Bai, H. Lin, J. Zhao, L. Hou, and C. V. Cannistraci, "Plug-and-play: An efficient post-training pruning method for large language models," in *The Twelfth International Conference on Learning Representations*, 2024.