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

Irene Simó Muñoz, Giulia Guidi (advisor)

CornellBowers
College of Computing + Information Science

Why

Current hardware trends such as **Processing-In-Memory (PIM)** and the use of **BF16 in GPU's tensor cores** are motivated by AI workflows. These can be exploited for scientific applications as well, but mapping a sparse or numerically sensitive solver onto them requires answering two prerequisite questions: **1.** ***Can sparse kernels meet PIM's lockstep execution model efficiently?*** **2.** ***Can iterative methods tolerate the reduced precision of ML formats?*** This poster reports on both, and together, it lays the groundwork for an energy-efficient sparse mixed precision iterative solver on PIM

Methodology

Project 1: Sparsity on Processing in Memory

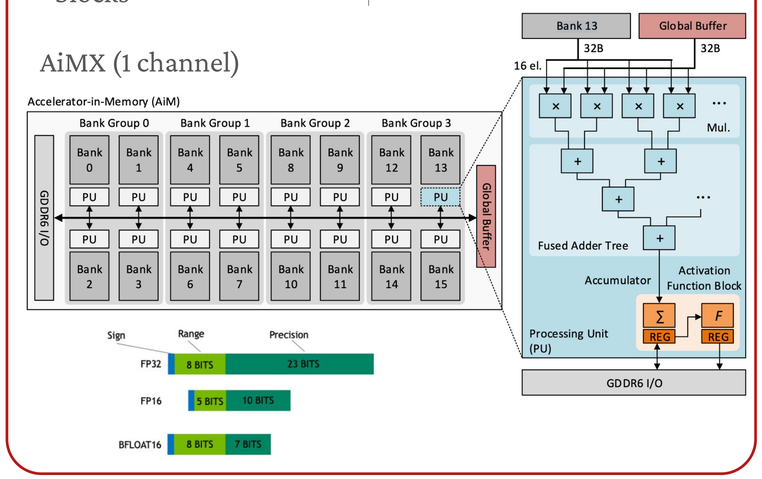
SK hynix

- AiMX prototype (dense inference)
- Bank parallelism, lockstep GEMV execution
- Lockstep execution forces input matrix discretization in 16x16 blocks

Project 2: Mixed precision scientific computing with ML precision formats

- *s*-step Stochastic Gradient Descent as example application
- 16 bits of precision (FP16) is typically break convergence for iterative methods

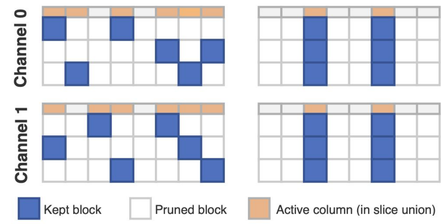
AiMX (1 channel)



1. Sparse Inference on PIM (AiMX)

Problem

Lockstep cost = column union, not nonzero count.
Naive scalar pruning leads to irregular sparsity patterns that enforce dense-like execution (GEMV) even for sparse data



Solution: AiMX-tuned LLM pruning and sparse matrix storage

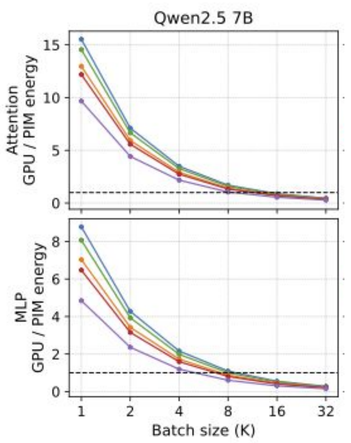
*Activation-aware pruning methods like **RIA** naturally produce column-aligned pruned matrices*

- Per model, we keep *N* boundary layers dense
- Per each layer. we prune attention sublayer matrices more aggressively than MLP ones
- Per attention and MLP sublayers, we prune downward projection matrices less aggressively

Results

Pruning strategy and AiMX specific storage allow for lower memory footprint that materialize in reductions of # of boards needed

Model	# boards Dense	# boards Ours	Pruning
Gemma-2-9B [59]	2	1	a70/m85
Gemma-2-27B [59]	4	3	a70/m85
InternLM2-20B [9]	3	2	a50/m75
LLaMA-3.1-70B [20]	8	7	a50/m75
Qwen-2.5-72B [26]	9	7	a50/m75
	9	8	a70/m85
Qwen-1.5-110B [7]	13	11	a50/m75
	13	12	a70/m85
Mistral-Large-123B [44]	15	12	a50/m75
	15	13	a60/m80
	15	14	a80/m90
Command-R+-104B [11]	12	10	a50/m75
	12	11	a80/m90



Energy consumption is ~5x less on average than an H100 running the same model both cuBLAS and cuSPARSELt for interactive batch sizes

2. Mixed Precision s-step SGD

Problem

standard SGD: GEMV | sync | GEMV | sync — *s* sync per *s* steps

s-step SGD: SYRK (Gram) | Inner step (*s* times) | sync — 1 sync per *s* steps

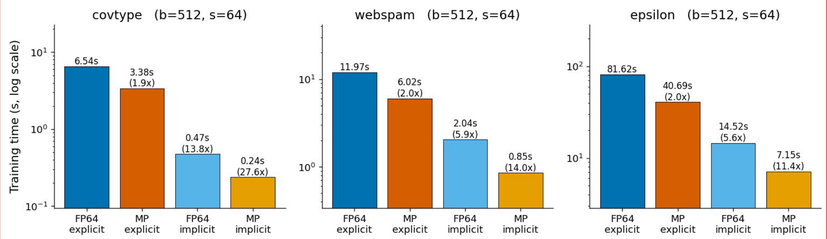
One SYRK replaces *s* memory passes, but FP32 rounding accumulates over *s* steps. **Can we run the Gram kernels in lower precision without breaking convergence?**

Solution: Mixed precision algorithm and implicit Gram construction

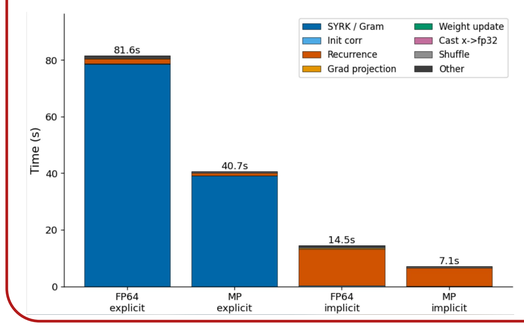
Operation	Precision (GPU)	Precision (CPU)
Storage A_scaled	BF16	FP32
SYRK/GEMV	BF16	FP32
Inner correction sum	FP32	FP64
Iterate *x* / DAXPY update	FP32	FP64

Explicit Gram computation is bottlenecked by SYRK, we propose an implicit variant that skips it and runs GEMVs against the raw block

Results



Mixed precision variant is always faster than baseline across datasets, batch sizes and s values



Bottleneck shift: the explicit variant spends most time in SYRK, the matrix-free variant in the GEMV recurrence

Future work

Joint application: sparse mixed-precision iterative solver on AiM-style PIM

- Explore multiboard setup with more AiMX cards
- Suggest architectural modifications for better sparse support

- Investigate how latest results translate to sparse datasets

Acknowledgements to collaborators Cecilio C. Tamarit, Joseph Maheshe, Mukund Gaur, José Martínez (Project 1, all Cornell University) and Aditya Devarakonda (Project 2, Wake Forest University)