Cheap Plans, Costly Trades: Host–Device Tradeoffs for SpGEMM on AiM

Calvin Jialin Zhang
Cornell University
Ithaca, NY, USA
cjz37@cornell.edu

Giulia Guidi (Advisor)
Cornell University
Ithaca, NY, USA
gguidi@cornell.edu

Abstract

Sparse matrix–matrix multiplication (SpGEMM) is bandwidth-bound, the kind of kernel a memory-side accelerator ought to absorb, but the processing-in-memory hardware that ships is built for AI inference. We map $C = A \cdot A$ onto SK hynix's Accelerator-in-Memory (AiM), whose fixed-function units multiply dense bfloat16 vectors but cannot compute an address, so the host must place every product in advance. The mapping trades host planning for device time; we measure the trade over four designs, from almost no planning to a simulator-in-the-loop search. A single packing pass cuts device time 7–122×, every further plan buys at most 8.5× more, and on a single product each plan costs more host time than it saves.

1 Introduction

SpGEMM is a core operation of sparse computing [5]: squaring an adjacency matrix underlies triangle counting, algebraic multigrid coarsens through a triple product, and GraphBLAS exposes it as a primitive [4]. The same codes stall on it, because it performs little arithmetic for every byte it moves.

Accelerators have been designed for SpGEMM [9], but none can be bought; the processing-in-memory devices that reach production serve what industry buys, putting dense-vector units beside DRAM for AI inference [6]. Scientific computing therefore does not get to choose its accelerator. We ask how much of our workload such a device can be made to run. For SpGEMM on SK hynix's Accelerator-in-Memory (AiM), the host must lay the computation out in advance, so the question becomes a trade: does planning that shrinks the device's work shrink the total time?

2 Mapping to AiM

AiM places an arithmetic unit beside each of a GDDR6 channel's 16 banks [6]. A unit pairs 32 B from its bank with 32 B broadcast from a global buffer, 16 bfloat16 lanes each, reducing the products into one accumulator; EWMUL skips the adder tree, writing its 16 products into a third bank [10].

We build C from rank-1 updates,

$$C = A \cdot A = \sum_k A(:,k)\, A(k,:), \tag{1}$$

multiplying only the nonzeros of column k by those of row k. An inner product would multiply AiM's fixed-width padding; Gustavson's scheme [3] needs indexed accumulation the units lack.

The outer product carries two costs. One is *routing*: EWMUL produces 16 products but cannot add them, so each reaches the accumulator owning its output row,

$$\mathrm{acc}[\ell] \mathrel{+}= A[r,c] \cdot A[c,j], \qquad \ell = r \bmod 16, \tag{2}$$

by a copy into the global buffer and a one-hot MAC keeping one lane of 16. The other is a *write floor*: both EWMUL operands come from banks, so every product needs an operand written across the external bus, and $A \cdot A$ performs 1.3–52 products per nonzero of A.

Table 1: Simulated device cycles for the four designs on 32 channels. Bold marks the fastest.

Matrix	Domain	Rows	S1 naive	S2 pack	S3 plan	S4 search
GD99_b	graph	64	11.8 K	1,597	782	**610**
GD96_1k	graph	1,096	556 K	22.7 K	4,525	**3,611**
can_256	structural	256	816 K	33.4 K	14.8 K	**13.3 K**
G32	random	2,000	7.23 M	59.0 K	7,042	**7,040**
SciMet	citation	3,084	6.29 M	480 K	101 K	**92.0 K**
G29	random	2,000	26.8 M	1.79 M	485 K	**483 K**
bitcoin	social	3,783	67.4 M	4.88 M	2.29 M	**2.10 M**
smallworld	small-world	100 K	1.26 B	11.7 M	3.07 M	**3.04 M**
roadNet-PA	road	1.09 M	2.58 B	53.6 M	6.66 M	**6.49 M**
belgium_osm	road	1.44 M	1.71 B	44.2 M	**5.22 M**	**5.22 M**
webbase-1M	web	1.00 M	4.03 B	**205 M**	280 M	261 M

3 Results

Only the host planning varies; the device is 32 simulated AiM channels. **S1** (naive) partitions output columns across channels and packs nothing. **S2** (pack) adds one layout pass: nonzeros of a row block (16 output rows, one per accumulator) on distinct lanes share a 32 B cell, and one command carries up to 32 cells; symmetric A uses its upper triangle. **S3** (plan) sorts rows by degree and merges lane-disjoint columns. **S4** (search) times candidate row orderings on the simulator. CENT's AiM extension of Ramulator 2.0 [2, 8] simulates device time for each trace over 11 SuiteSparse matrices [1], counting a cycle as a nanosecond at AiM's 1 GHz operating speed [7]. Outputs match a single-precision reference SpGEMM rounded to bfloat16, except on `webbase-1M`, which differs by at most 2^{-8} because AiM rounds each product to bfloat16 before accumulating. Host planning cost is CPU core-seconds on an Intel Xeon W-1390P or Core i9-13900K, single-threaded except S4's 8-worker search; it includes S4's candidate timings but not the final run, whose cycles are the device time.

Returns fall off steeply (Table 1): S2 cuts device time 7–122×, the static plan a further 2.0–8.5×, the search at most 1.28×. On `webbase-1M` they turn negative: S3 is 1.4× slower than S2 for 21× the planning. No plan repays itself on one product (Fig. 1): against S1 or the design before it, each costs more host time than it saves in device time.

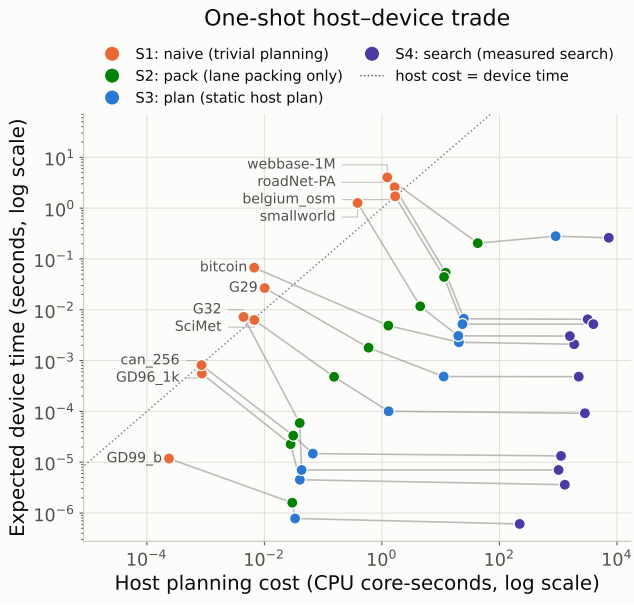


Figure 1: The host–device trade for a single product. Each line follows one matrix through the four designs: planning cost rises at every step and device time falls, except on `webbase-1M` from S2 to S3 and `belgium_osm` from S3 to S4. Expected device time is simulated cycles at one nanosecond per cycle.

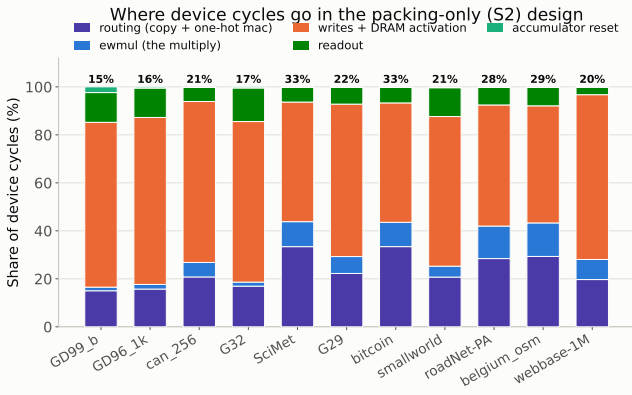


Figure 2: Device cycles in S2 by AiM opcode. Routing's share is printed above each bar.

4 Discussion and Conclusion

Fig. 2 attributes each command's controller cycles to its opcode. Routing, the copy and one-hot MAC, takes 15–33%, the EWMUL multiply at most 14%, readout and reset 3–15%. Routing is at most 7.0% of commands (Fig. 3) yet costly in cycles: every scatter waits out a reduction window.

The win comes from S2. Routing is the lever that remains and one that hardware could remove, worth up to about 1.5×: an instruction accumulating each EWMUL product directly into its output lane would delete both the copy and the one-hot MAC, needing 16

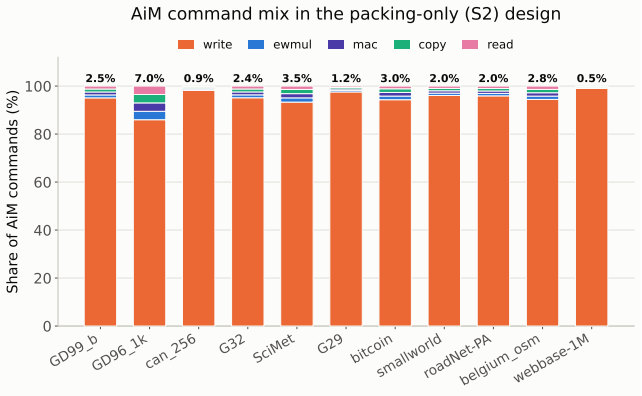


Figure 3: AiM command mix in S2, with routing's share of the commands printed above each bar. Writes dominate the count, routing at most 7.0% of it.

accumulators and 16 adders per unit on top of the adder tree that MAC still needs.

For a single product, then, planning that shrinks the device's work does not shrink the total: every plan we measured costs the host more than it saves. Whether the same holds for a general $A \cdot B$, or when the sparsity pattern repeats with new values as in algebraic multigrid setup, is the next question.

Acknowledgments

We thank Irene Simó Muñoz for her feedback on this project.

References

[1] Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (2011), 25 pages. doi:10.1145/2049662.2049663

[2] Yufeng Gu et al. 2025. PIM Is All You Need: A CXL-Enabled GPU-Free System for Large Language Model Inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25)*. 862–881. doi:10.1145/3676641.3716267

[3] Fred G. Gustavson. 1978. Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition. *ACM Trans. Math. Softw.* 4, 3 (1978), 250–269. doi:10.1145/355791.355796

[4] Jeremy Kepner et al. 2016. Mathematical foundations of the GraphBLAS. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. doi:10.1109/HPEC.2016.7761646

[5] Jeremy Kepner and John Gilbert. 2011. *Graph Algorithms in the Language of Linear Algebra*. Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898719918

[6] Daehan Kwon et al. 2023. A 1ynm 1.25V 8Gb 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep Learning Application. *IEEE Journal of Solid-State Circuits* 58, 1 (2023), 291–302. doi:10.1109/JSSC.2022.3200718

[7] Yongkee Kwon et al. 2022. System Architecture and Software Stack for GDDR6-AiM. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–25. doi:10.1109/HCS55958.2022.9895629

[8] Haocong Luo et al. 2024. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 23, 1 (2024), 112–116. doi:10.1109/LCA.2023.3333759

[9] Subhankar Pal et al. 2018. OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 724–736. doi:10.1109/HPCA.2018.00067

[10] SK hynix Memory Solution Product Design (MSPD) Division. 2022. Accelerator-in-Memory: SK hynix's GDDR6-based Processing-in-Memory Device. Whitepaper, SK hynix. https://product.skhynix.com/support/downloads/kits.go