

Fused Multiply-and-Prune SpGEMM for Bounded-Memory Iterative Graph Diffusion

André Tomitan Bocces, Alexandro Baldassin

¹Departamento de Estatística, Matemática Aplicada e Computação
Universidade Estadual Paulista (UNESP) — Rio Claro, SP, Brazil

at.bocces@unesp.br, alexandro.baldassin@unesp.br

Abstract. *Iterative sparse graph algorithms with a bounded-output constraint, requiring that only the top- L entries per output row be retained after each SpGEMM step, cannot be executed at scale by general-purpose sparse linear algebra libraries, which follow a model that materializes the full $O(NK^2)$ intermediate product before any row truncation, causing out-of-memory failure or OS thrashing even though the $O(NL)$ final output fits in available memory. We present an engine, named Sparse-CSR, that prevents the intermediate product from ever being instantiated. Evaluated across three hardware tiers on graphs up to $N=160,731$ nodes, the engine achieves a $400\times$ peak-memory reduction and up to $2.4\times$ wall-clock speedup.*

Keywords: SpGEMM, Manifold Learning, High Performance Computing, Memory Optimization, Graph Algorithms, Compressed Sparse Row.

1. Introduction

A broad class of iterative graph algorithms requires repeated Sparse General Matrix-Matrix Multiplication (SpGEMM) in which only the top- L entries per output row are retained after each iteration [Gustavson 1978, Buluç and Gilbert 2012]. This bounded-output constraint creates a fundamental tension with the compute-then-prune model of all SpGEMM libraries [Dalton et al. 2015, Nagasaka et al. 2019]: the complete intermediate product $C = A \times B$ must be materialized before any top- L selection, reaching $O(NK^2)$ non-zeros when each row of A has K non-zeros, irrespective of the $O(NL)$ final output. At representative parameters ($N=160,000$ nodes, $K=50$ non-zeros per row, $L=400$ retained entries), the intermediate product reaches 4×10^8 entries while the pruned result contains $\approx 6.4 \times 10^7$, producing two irreducible failure modes: OOM termination when the intermediate product exceeds physical RAM, and OS thrashing when mmap-backed swap drives high page-fault rates. These modes affect SuiteSparse:GraphBLAS [Davis 2019], Eigen [Guennebaud et al. 2010], and Intel MKL equally. Parallel CSR construction under high triplet-emission rates further introduces allocator lock contention, TLB pressure from 4 KB page granularity, and false sharing on per-thread control structures. No existing SpGEMM primitive addresses these jointly.

This paper presents a three-component engine, herein named Sparse-CSR, comprising: (i) a fused multiply-and-prune kernel enforcing an $O(NL)$ working-set bound; (ii) a BlockAllocator over 2 MB THP-aligned arenas reducing TLB miss rate by $512\times$; and (iii) a lock-free three-stage row-split CSR pipeline with no synchronization on the hot path. The engine is evaluated on the LHRR graph diffusion algo-

rithm [Pedronette et al. 2019] from the UDLF framework [Valem and Pedronette 2017], with output bitwise-identical to dense references.

2. Engine Architecture

2.1. Fused Multiply-and-Prune SpGEMM Kernel

For each output row i , the kernel maintains a thread-local dense accumulator `dense_acc[0..N-1]` and a companion `active_cols` list. Each non-zero pair (A_{ik}, B_{kj}) is added to `dense_acc[j]`, with j appended to `active_cols` on first access. When $|\text{active_cols}| > L$, `std::nth_element` (expected $O(|\text{active_cols}|)$) retains the L largest entries and zeroes the rest, so the $O(NK^2)$ intermediate product is never instantiated. The ≈ 640 KB accumulator fits within server L2 cache, keeping per-row zeroing DRAM-independent. Correctness follows because the kernel visits the identical non-zero pairs as a full-product computation, so the top- L output is bitwise-identical to the dense reference. Although invoking `std::nth_element` introduces an expected $O(|\text{active_cols}|)$ CPU overhead per row, this cost is amortized by the reduction in memory traffic, as the accumulator remains within the L2 cache, avoiding DRAM accesses.

2.2. BlockAllocator: THP-Aligned Thread-Local Arena

LHRR’s two-hop expansion phase emits up to NK^2 coordinate triplets in parallel. Standard `malloc` implementations often serialize on heap locks and require $NK^2/4096 \approx 244,000$ distinct 4KB TLB entries for $N=160,000$, $K=50$, far exceeding typical L2 TLB capacity. Each thread instead receives a private pool of 2 MB arenas registered via `madvise(MADV_HUGEPAGE)`: a single 2 MB entry covers 512 standard pages, reducing TLB pressure by $512\times$ under sequential writes, while allocation reduces to a pointer-bump increment and eliminates lock contention. A `PaddedAllocator` wrapper adds 128 bytes of per-thread footprint to prevent false sharing across the NUMA topology.

2.3. Lock-Free Row-Split CSR Construction

Three stages with no synchronization on the hot path convert T per-thread triplet partitions into a single CSR matrix. *Stage 1*: each thread counts row occurrences with strictly local writes. *Stage 2*: histograms are reduced in parallel to `row_ptr[0..N]` ($O(TN)$ work). *Stage 3*: threads write into disjoint row ranges, requiring no mutex or atomic operation. Post-fill intra-row sorting uses `C++17 std::execution::par_unseq`; rows already in sorted order skip this via a monotonicity check.

3. Experimental Evaluation

To evaluate our engine, we integrated Sparse-CSR into the execution flow of the LHRR algorithm provided by the UDLF framework, replacing the default compute-then-prune SpGEMM step with our fused multiply-and-prune kernel. LHRR is evaluated on ALOI [Geusebroek et al. 2005] ($N=72,000$), SUN397 [Xiao et al. 2010] ($N=108,754$), and VegFru [Hou et al. 2017] ($N=160,731$, ≈ 102 GB dense) across three environments: Env A, a 192 GB server ($2\times$ Xeon Gold 5220, 18 cores); Env B, the same machine restricted to 16 GB via cgroups (12 cores); and Env C, a consumer desktop (i5-10400F, 8 GB). All baselines are compiled with GCC 11 (`-O3 -march=native -fopenmp`); reported times are means of 10 runs ($\sigma < 2\%$).

Table 1. LHRR feasibility across hardware environments. Status: wall-clock time (s) or failure mode. Parameters: Env A $L = 2,000$, $K = 50$, $T = 5$; Env B $L = 1,000$, $K = 20$, $T = 5$; Env C $L = 1,000/400^\dagger$, $K = 20$, $T = 5$. OOM: out-of-memory. T/O: timeout >2 h (disk thrashing).

Collection	Backend	Env A (192 GB)	Env B (16 GB)	Env C (8 GB)
ALOI (72K)	Sparse-CSR	227.5 s	57.6 s	37.8 s
	Intel MKL	264.9 s	75.4 s	78.8 s
	GraphBLAS	39.8 s	16.2 s	25.6 s
	Eigen	241.5 s	47.5 s	45.0 s
	Dense-MMAP	211.4 s	T/O	T/O
	Dense-Swap	214.4 s	54.7 s	OOM
SUN397 (108K)	Sparse-CSR	579.0 s	197.1 s	68.6 s [†]
	Intel MKL	OOM	OOM	OOM
	GraphBLAS	305.4 s	OOM	OOM
	Eigen	178.3 s	OOM	OOM
	Dense-MMAP	1327.4 s	T/O	T/O
	Dense-Swap	1348.5 s	OOM	OOM
VegFru (160K)	Sparse-CSR	695.2 s	258.6 s	92.1 s [†]
	Intel MKL	OOM	392.7 s	OOM
	GraphBLAS	214.8 s	OOM	OOM
	Eigen	190.3 s	OOM	OOM
	Dense-MMAP	1651.4 s	T/O	T/O
	Dense-Swap	1667.6 s	OOM	OOM

[†] L reduced to 400 in Env C for SUN397 and VegFru.

Table 1 reports results for all baselines. On ALOI, all implementations run to completion; GraphBLAS is fastest across all three environments, while Sparse-CSR remains competitive with the MKL and Eigen baselines and completes without failure in every configuration. On SUN397, the failure boundary is sharp: in Envs B and C, every compute-then-prune baseline fails through OOM or thrashing. On VegFru, the same pattern holds except for Intel MKL, which completes in Env B at 392.7 s; Sparse-CSR completes in all configurations for both collections. The cause is structural: the two-hop expansion phase emits $O(NK^2)$ triplets that exhaust each library’s allocator before any pruning can occur. In Env A, Sparse-CSR is $2.4\times$ faster than Dense-MMAP on VegFru (695.2 s vs. 1651.4 s), with the gap reversing relative to ALOI (227.5 s vs. 211.4 s) as dataset size grows. Peak RSS on VegFru at $L=400$, $K=20$ is 256.4 MB against ≈ 102 GB for the dense pre-allocation, a $400\times$ reduction.

4. Conclusion

The bounded-output SpGEMM problem exposes a structural mismatch between the $O(NL)$ information content of iterative graph algorithms and the $O(NK^2)$ memory demanded by compute-then-prune libraries. The three-component engine presented here suppresses intermediate growth at accumulation time, eliminates TLB pressure and false sharing during triplet emission, and removes all synchronization from the CSR construction hot path. On LHRR it delivers a $400\times$ memory reduction and up to $2.4\times$

wall-clock speedup on workloads where most evaluated alternatives fail, with bitwise-identical output for all input configurations. Replacing the dense accumulator with a hash-based $O(K^2)$ structure to lift the parallel scaling ceiling and GPU porting via warp-level Expand-Sort-Compress integration are the principal directions for future work.

References

- Buluç, A. and Gilbert, J. R. (2012). Parallel sparse matrix–matrix multiplication and indexing: Implementation and experiments. *SIAM J. Sci. Comput.*, 34(4):C170–C191.
- Dalton, S., Olson, L. N., and Bell, N. (2015). Optimizing sparse matrix–matrix multiplication for the GPU. *ACM Trans. Arch. Code Optim.*, 12(1):1–29.
- Davis, T. A. (2019). Algorithm 1000: SuiteSparse:GraphBLAS: Graph algorithms in the language of sparse linear algebra. *ACM Trans. Math. Softw.*, 45(4):1–25.
- Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. M. (2005). The amsterdam library of object images. *Int. J. Comput. Vis.*, 61:103–112.
- Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. <http://eigen.tuxfamily.org>.
- Gustavson, F. G. (1978). Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw.*, 4(3):250–269.
- Hou, S., Feng, Y., and Wang, Z. (2017). VegFru: A domain-specific dataset for fine-grained visual categorization. In *ICCV*, pages 541–549.
- Nagasaka, Y., Matsuoka, S., Suzuki, A., and Endo, T. (2019). High-performance sparse matrix–matrix products on Intel KNL and multicore architectures. In *Proc. 48th ICPP*, pages 1–10.
- Pedronette, D. C. G. a., Valem, L. P., Almeida, J., and Torres, R. d. S. (2019). Multimedia retrieval through unsupervised hypergraph-based manifold ranking. *IEEE Trans. Image Process.*, 28(12):5824–5838.
- Valem, L. P. and Pedronette, D. C. G. a. (2017). Unsupervised distance learning by reciprocal kNN distance for image retrieval. *ACM SIGMultimedia Records*, 9(2):1–1.
- Xiao, J., Hays, J., Ehinger, K. A., Oliva, A., and Torralba, A. (2010). SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pages 3485–3492. IEEE.