Distributed-Memory Sparse Kernels for Machine Learning

Vivek Bharadwaj[1], Aydın Buluç[1,2], James Demmel[1]

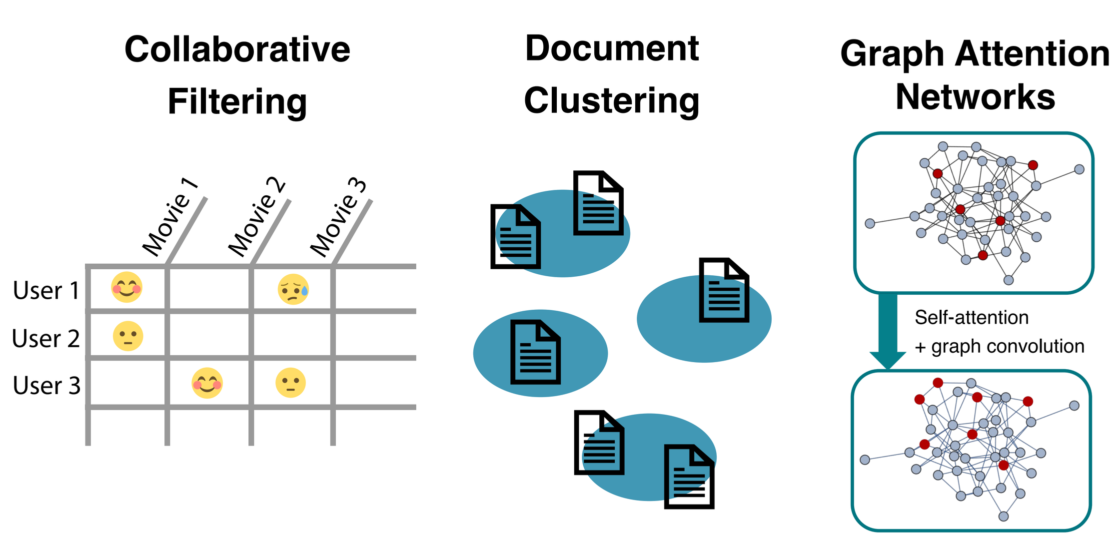
[1]UC Berkeley, [2]Lawrence Berkeley National Laboratory

Summary

- **We introduce the first distributed-memory, sparsity-agnostic, high-performance Sampled Dense-Dense Matrix Multiplication (SDDMM) algorithms. They can be used alone or in combination with Sparse-Times-Dense Matrix Multiplication (SpMM)**
- **We give strategies to reduce processor-to-processor communication in a sequence of SDDMM and SpMM calls, a pattern that applications commonly use**
- **We benchmark our algorithms on 256 KNL CPU nodes of LBNL Cori, a Cray XC40 supercomputer. We measure performance on collaborative filtering and graph attention network applications**

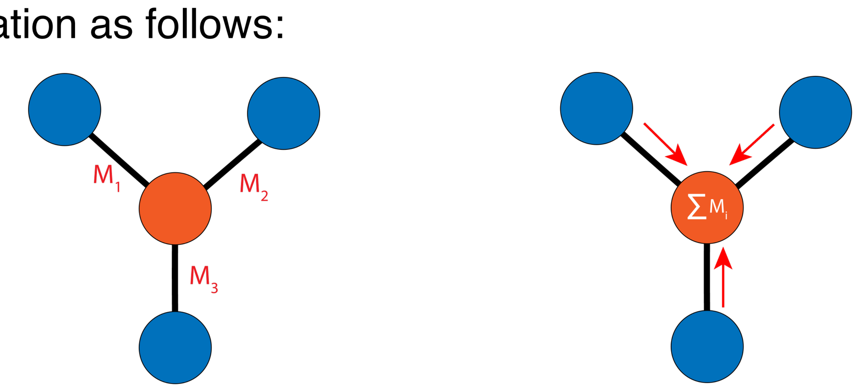
Background

Massive sparse matrices are ubiquitous in scientific computing and machine learning. Some examples:



These three applications share two intensive computational kernels: **Sampled Dense-Dense Matrix Multiplication** (SDDMM) and **Sparse-Times-Dense Matrix Multiplication** (SpMM).

Both involve **one sparse matrix** and a **pair of tall-skinny dense matrices**. In the special case that the sparse matrix represents the adjacencies of a graph, we interpret their operation as follows:



SDDMM **generates** messages on the edges of the graph

SpMM **aggregates** messages from edges to incident nodes

Objectives

Distributed-memory algorithms for general SpMM and SDDMM are heavily **communication-bound**. Our goals:

1. Build communication-avoiding algorithms for SDDMM based on existing designs for SpMM in the literature
2. Find strategies to reduce communication when performing SDDMM and SpMM in sequence, as many applications require.

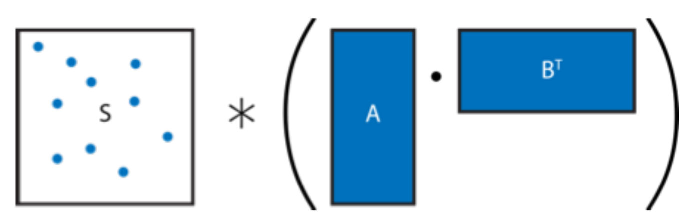
Definitions

Symbol	Definition
S, R	$m \times n$ sparse matrices
A	$m \times r$ dense matrix
B	$n \times r$ dense matrix
ϕ	The ratio $\mathrm{nnz}(S)/nr$

Symbol	Definition
p	Total processor count
$*$	Elementwise multiplication
$\cdot$	Matrix Multiplication

Define SDDMM as the function:

$$\mathrm{SDDMM}(S, A, B) = S * (A \cdot B^T)$$



Similarly, define two variants of SpMM:

$$\mathrm{SpMMA}(S, B) = S \cdot B \qquad \mathrm{SpMMB}(S, A) = S^T \cdot A$$

All three kernels have an **identical** data access pattern:

Every nonzero (i, j) of S requires an interaction between row i of A and row j of B.

$R \coloneqq \mathrm{SDDMM}(S, A, B)$	$A \coloneqq \mathrm{SpMMA}(S, B)$	$B \coloneqq \mathrm{SpMMA}(S, A)$
for $(i,j) \in S$: $R_{ij} \coloneqq S_{ij}(A_{i:} \cdot B_{j:}^T)$	for $(i,j) \in S$: $A_{i:} \mathrel{+}= S_{ij} B_{j:}$	for $(i,j) \in S$: $B_{j:} \mathrel{+}= S_{ji} A_{i:}$

Distributed-Memory SDDMM Algorithms

Several existing works give communication-avoiding, distributed-memory algorithms for SpMM. Using the identical data access patterns of the kernels, we observe:

Any distributed-memory algorithm for SpMM can be transformed into a distributed-memory algorithm for SDDMM with identical communication characteristics, and vice-versa.

Transformation Example: Begin with an SpMMA algorithm performing **no replication** of input / output operands.

1. **Convert the sparse input S to a 0-initialized output**
2. **Use A as an output buffer instead of an input buffer**
3. **Change each processor's local update to $S_{ij} \mathrel{+}= A_{ik} B_{jk}$**

1.5D, 2.5D variants of SUMMA / Cannon **replicate** input and output operands to reduce communication bandwidth between processors. Inputs are typically replicated via broadcasts, outputs require reduction. To handle replication:

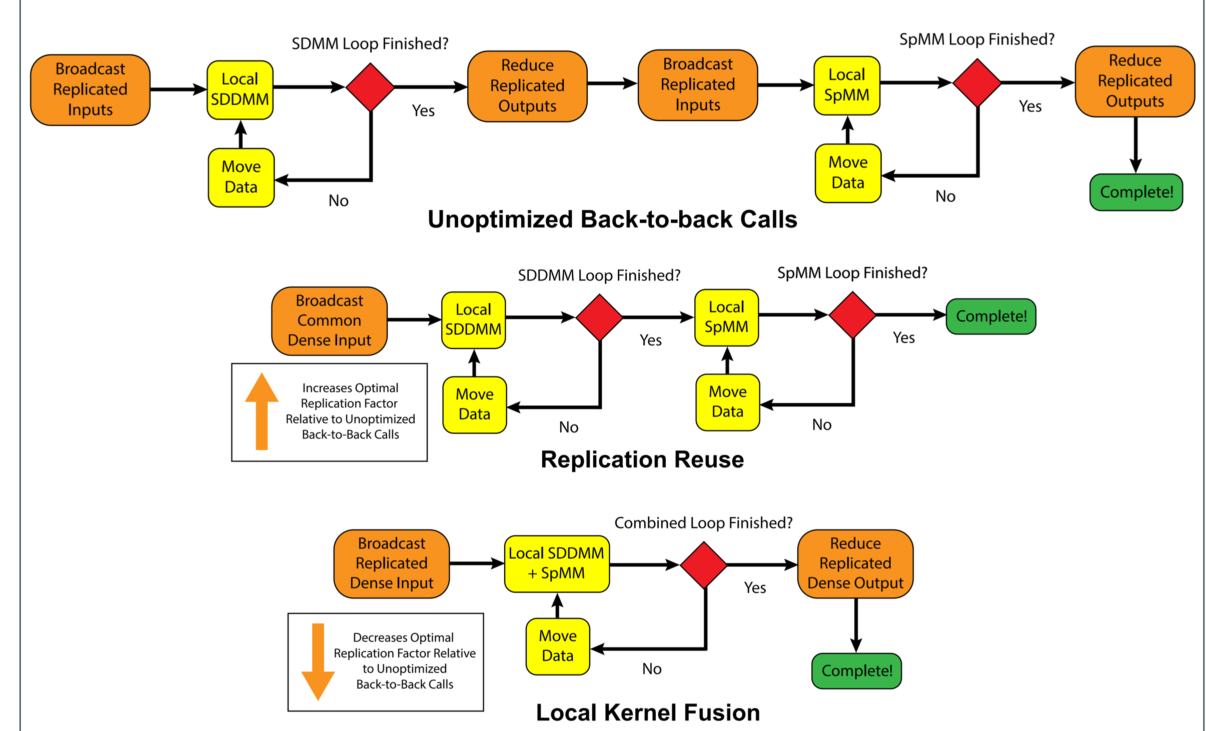
4. **Convert initial input broadcasts to terminal reductions**
5. **Convert terminal reductions to initial broadcasts**

Distributed-Memory FusedMM Algorithms

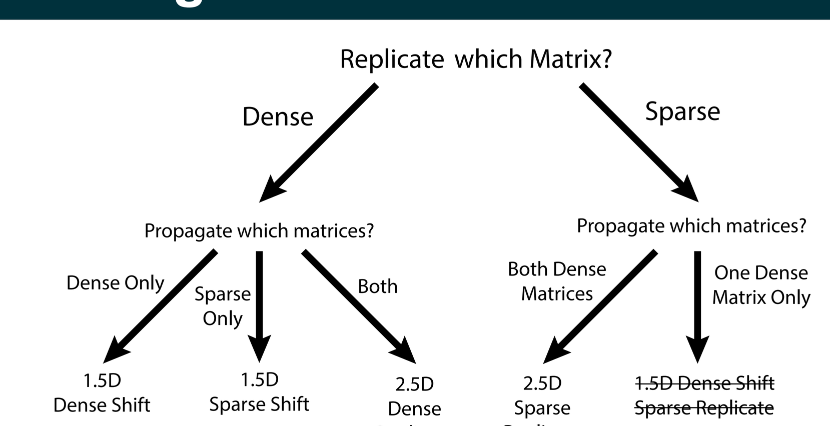
Several applications execute SDDMM and feed the sparse output to SpMM, an operation we define as **FusedMM**:

$$\mathrm{FusedMMA}(S, A, B) \coloneqq \mathrm{SpMMA}(\mathrm{SDDMM}(S, A, B), B)$$
$$\mathrm{FusedMMB}(S, A, B) \coloneqq \mathrm{SpMMB}(\mathrm{SDDMM}(S, A, B), A)$$

For FusedMM, we can **eliminate unnecessary communication rounds and adjust the degree of replication** through one of two strategies: replication reuse or local kernel fusion. Either approach lets us **adjust** the replication factor to further reduce communication costs.



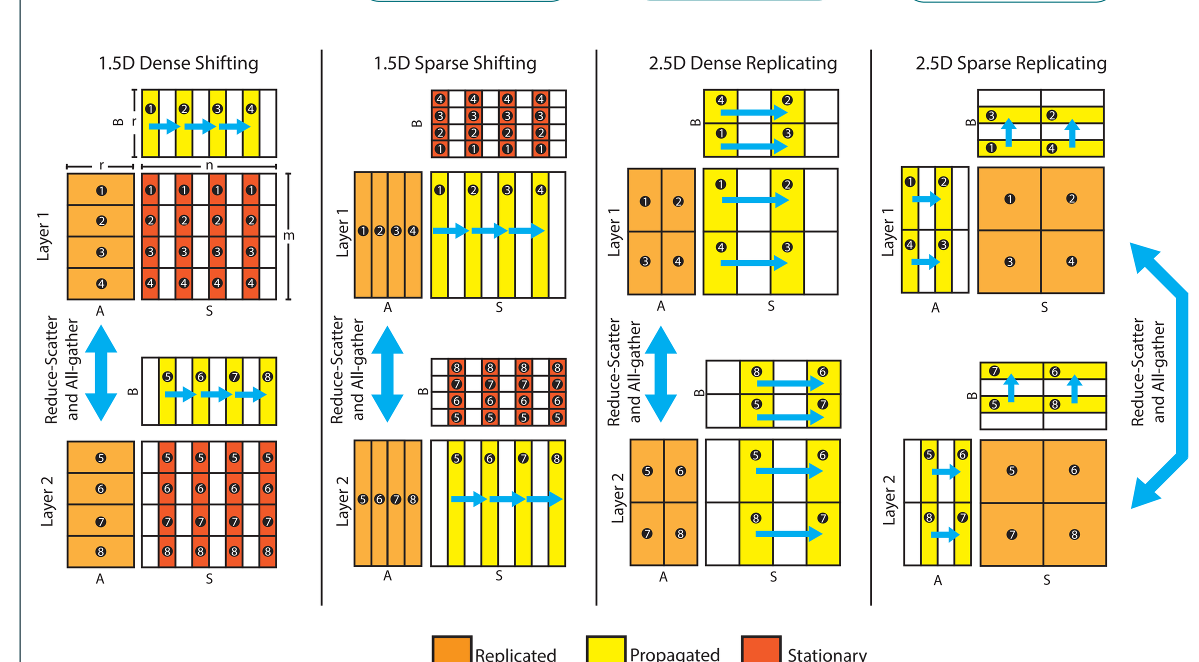
Algorithm Data Movement



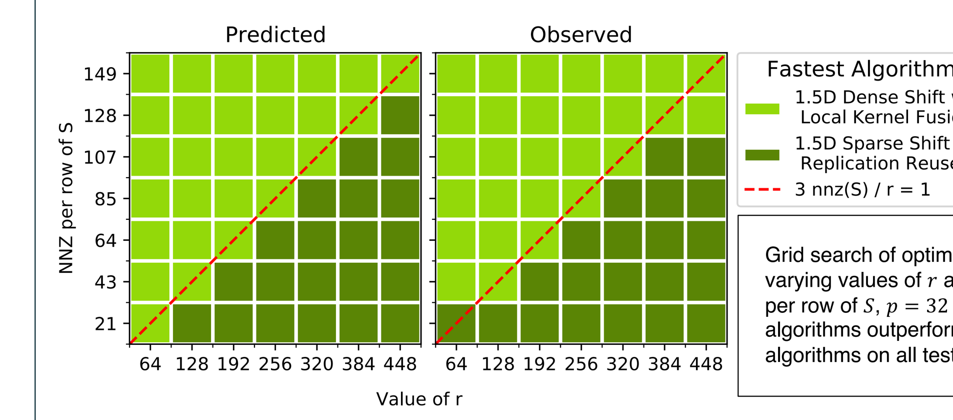
Data flow in our **sparsity-agnostic** algorithms is determined by choosing which operands we **replicate, propagate,** and keep **stationary**. These choices affect the **communication volume** for processors. Below, we assume $m \approx n$.

Asymptotic Communication Costs

$$O\left(\frac{nr}{p^{1/2}}\right) \quad O\left(\frac{nr\phi^{1/2}}{p^{1/2}}\right) \quad O\left(\frac{nr\phi^{2/3}}{p^{2/3}}\right) \quad O\left(\frac{nr\phi^{1/3}}{p^{2/3}}\right)$$



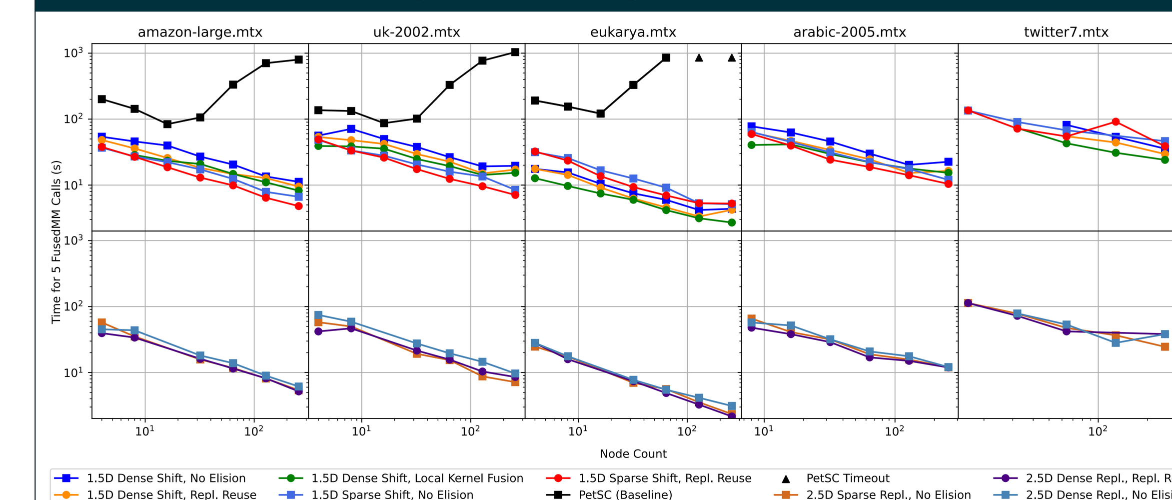
Optimal Algorithm Selection



Grid search of optimal algorithm for varying values of r and nonzero count per row of S, $p = 32$ KNL nodes. 1.5D algorithms outperformed 2.5D algorithms on all tested configurations.

The choice of optimal algorithm depends on ϕ. When ϕ is low, communicating the sparse matrix is cheap, so 1.5D sparse shifting / replicating algorithms are fastest. For higher values of ϕ, dense shifting / replicating algorithms run faster.

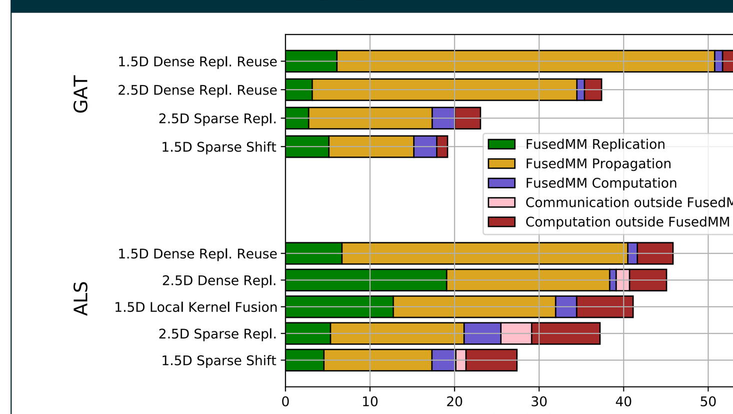
Strong Scaling Benchmarks



Matrix	Side Length	Nonzero Count	NNZ per Row
amazon-large.mtx	14,249,639	230,788,269	~16
uk-2002.mtx	18,484,117	298,113,672	~16
eukarya.mtx	3,243,106	359,744,161	~111
arabic-2005.mtx	22,744,080	639,999,458	~28
twitter7.mtx	41,652,230	1,468,365,182	~35

Benchmarks performed on up to 256 Knight's Landing nodes on LBNL Cori. Black lines indicate the performance of the `MatMatMult` function available in PETSc, with black triangles indicating that a benchmark took longer than 3 hours.

Application Benchmarks



Benchmarks reported for amazon-large.mtx. 20 iterative CG updates were performed for the collaborative filtering application: 10 to optimize matrix A and 10 to optimize matrix B. The GAT benchmark was run on a 3-layer network with uniform node embedding dimension 256. The layers had 4, 4, and 6 attention heads, respectively.

We used our algorithms to perform collaborative filtering with alternating least squares (**ALS**) and graph attention network inference (**GAT**). The benchmarks match our strong scaling experiments closely, with comparatively little time spent outside SDDMM / SpMM.

Acknowledgements

This work originally appeared as "Distributed-Memory Sparse Kernels for Machine Learning" at the 36th IEEE International Parallel Parallel and Distributed Processing Symposium (IPDPS), May 30 through June 3, 2022.

Scan the QR code to read the paper!

This material is based upon work supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0022158. This work is also supported by the Office of Science of the DOE under contract number DE-AC02-05CH11231. We used resources of the NERSC supported by the Office of Science of the DOE under Contract No. DE-AC02- 05CH11231.