Two-Face: Combining Collective and One-Sided Communication for Efficient Distributed SpMM

Charles Block*, Gerasimos Gerogiannis*, Charith Mendis, Ariful Azad, and Josep Torrellas

University of Illinois at Urbana-Champaign
& Indiana University Bloomington

coblock2@illinois.edu; gg24@illinois.edu

*Equal contributions. Order is alphabetical.
SpMM

- Sparse Matrix × Dense Matrix Multiplication
- Major operation in Graph Neural Networks and linear algebra solvers
- Large graph datasets are Terabyte-scale
  - Distributed SpMM is important
SpMM

- A’s sparsity pattern determines accesses to B & C
  - Nonzero column determines row of B
  - Nonzero row determines row of C
- Often irregular memory accesses in shared memory systems
- Irregular communication in distributed systems

\[ C[1,0:K-1] = C[1,0:K-1] + a \times B[6,0:K-1] \]
Distributed SpMM

- Partitioned matrices
  - In this work: 1D Partitioning
  - Each node holds a set of rows from each matrix
- Communication of input dense matrix B is required between nodes
Communication Patterns

(1) Sparsity-Unaware

- Typically executed with synchronous collective transfers (sync transfers)
- May transfer redundant data 😞

(2) Sparsity-Aware

- Typically executed with asynchronous one-sided transfers (async transfers)
- May require many round-trips 😞
- High software overhead 😞
The optimal choice is matrix-dependent
Region-Dependent Communication Patterns

- Nonzeros in the same sparse matrix are not evenly distributed.
- They form denser and sparser regions.

To optimize communication, we must utilize *different communication patterns* for *different sparse matrix regions*.

Matrix density maps from http://sparse.tamu.edu/
A Hybrid Communication Algorithm
Some nonzeros do not trigger remote transfers

The rest will be classified as either synchronous (sync) or asynchronous (async)
A Hybrid Communication Algorithm

- Nonzeros in denser sparse matrix regions are classified as sync.
A Hybrid Communication Algorithm

- Nonzeros in denser sparse matrix regions are classified as sync.
- The corresponding B row-blocks are transferred with coarse-grained multicast operations.

Dense rows broadcast from N2 to N0, N1 and N3.
A Hybrid Communication Algorithm

- The rest of the nonzeros are classified as async
A Hybrid Communication Algorithm

- The rest of the nonzeros are classified as async.
- The corresponding B rows are transferred with fine-grained async transfers.
A Hybrid Communication Algorithm

We materialize this hybrid communication pattern with an algorithm called **Two-Face**.
Two-Face Flow

- Preprocessor
- System Profile
- Dist. SpMM
Partitioning the Matrices

- **Main unit of communication:** *stripes*
  - Sparse stripes: groups of nonzeros that use the same communication flavor
  - Dense stripes: corresponding groups of input rows that will actually be transferred

- Sparse stripe is classified as sync or async
Preprocessing Step

Should we use sync transfers or async transfers?

- Partitioning algorithm goals:
  - Balance time for sync and async
  - Use best communication flavor for each stripe

- Stripe classification:
  - Linear cost model
  - Partitioning algorithm utilizing cost model estimates
Cost Model

Linear model of communication / computation costs

\[ Comm_A = \beta_A K L_A + \alpha_A S_A \]

- Communication cost of async transfers
- Number of columns in dense matrix
- Number of rows to be transferred
- Number of async stripes
Cost Model

Linear model of communication / computation costs

\[ Comm_A = \beta_A KL_A + \alpha_A S_A \]

Cost model coefficients are empirically determined by profiling and linear regression.

Depend on system characteristics

Depend on input matrices and stripe classification
Sparse Matrix Format

- Preproc classifies stripes as sync/async
  - Equivalently: non-zeros as sync/async

- Sync/local-input stored in one structure
- COO with additional tile information
- Row-major order optimizes for computation

- Async stored in separate structure
- COO with pointer to stripes
- Column-major order optimizes for communication
Two-Face

DoParallel

\[ tid \leftarrow \text{GetTID()} \]

\[ \text{tid in SyncThreads} \]

if \( tid = 0 \) then
    TransferSyncStripes()
end if

Collectives

WaitForSyncTransfers()
while \( n \leftarrow \text{GetNextSyncTile()} \) do
    ProcessSyncTile(\( n \))
end while

Compute

\[ \text{tid in AsyncThreads} \]

while \( n \leftarrow \text{GetNextAsyncStripe}() \) do
    ProcessAsyncStripe(\( n \))
end while

Fetch + Accumulate

EndParallel
Methodology

- **Evaluation system:**
  - Per-node: 128 CPU cores; 256 GB RAM
  - 1 to 64 nodes

- **Software:**
  - OpenMPI for inter-node parallelism
  - OpenMP for multi-threading

- **Baselines:**
  - Dense Shifting [Bharadwaj et al., 2022] (Sparsity-Unaware)
  - All-gather (Sparsity-Unaware)
  - Coarse- & fine-grained async-only (Sparsity-Aware)

- **Benchmarks**
  - Matrices from SuiteSparse with up to **3.6 billion nonzeros** and **214 million rows**
  - Dense matrices with between 32 and 512 columns
Performance Results

- For 128 dense columns: 2.11x speedup vs. best-choice dense shifting
- Across K={32, 128, 256}: 1.99x speedup for 32 nodes
Performance Scaling

![Graphs showing performance scaling for GAP-web and Queen_4147 datasets. The graphs display execution time vs. node count, highlighting the scalability of different configurations.]
Conclusion

- Distributed SpMM is an important communication-bound workload

- Sparse matrices often contain sparser & denser regions

- By scheduling communication differently according to internal characteristics, we can improve performance

- *Two-Face* combines collective and one-sided communication to achieve 2x speedup and better scalability over state-of-the-art
Two-Face: Combining Collective and One-Sided Communication for Efficient Distributed SpMM

Charles Block*, Gerasimos Gerogiannis*, Charith Mendis, Ariful Azad, and Josep Torrellas

University of Illinois at Urbana-Champaign & Indiana University Bloomington

coblock2@illinois.edu; gg24@illinois.edu

*Equal contributions. Order is alphabetical.
Performance Breakdown

[Bar chart showing normalized execution time for different benchmarks and categories: Sync Comp, Async Comp, Other, Sync Comm, Async Comm]
Sensitivity to Preprocessing Calibration

(a) Varying $\alpha_A$ and $\beta_A$

(b) Varying $\alpha_S$ and $\beta_S$

(c) Varying $\gamma_A$ and $\kappa_A$
Model Communication and Computation costs as linear functions

\[ Comm_S = S_S (\beta_S K W + \alpha_S) \]
\[ Comm_A = \beta_A K L_A + \alpha_A S_A \]
\[ Comp_A = \gamma_A K N_A + \kappa_A S_A \]

Attempt to balance the two sides:

\[ Comm_S = Comm_A + Comp_A \]