Exploring the Design Space of Distributed Parallel Sparse Matrix-Multiple Vector Multiplication

Optimizing the Communication Bottleneck in HPC

Prashant Singh

Roll No: 123cs0070

Course: Parallel and Distributed Computing

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (2024)

1. Introduction to SpMM

- **Mathematical Definition:** SpMM computes $C = A \times B$.
- **Dimensions:**
  - A is a sparse matrix of size $m \times k$.
  - B is a dense matrix (or collection of dense vectors) of size $k \times n$.
  - C is the resulting dense matrix of size $m \times n$.
- **The Core Difference:** Traditional SpMV (Sparse Matrix-Vector) computes $y = Ax$ where x is a single column. SpMM scales this up, executing the computation simultaneously for n columns, exposing massive arithmetic intensity.

The Sparsity Challenge

Matrix A may contain 99% zeros. While this reduces the raw FLOP count, the unpredictable memory access patterns make it notoriously difficult to achieve high hardware utilization.

2. Modern Applications of SpMM

SpMM is the computational backbone of Several High-Performance Computing (HPC) domains:

- **Graph Neural Networks (GNNs):** In GNN training, A represents the massive sparse adjacency matrix of the graph (who is connected to whom), and B contains the dense feature vectors of each node (e.g., user profiles). SpMM aggregates neighbor features.
- **Deep Learning Mini-Batching:** When training DNNs, calculating gradients for a mini-batch of n inputs simultaneously relies entirely on SpMM.
- **Scientific Solvers:** Block iterative methods (like Block Conjugate Gradient or LOBPCG) solve multiple linear systems at once to accelerate convergence in physics simulations.

3. The Distributed Memory Challenge

- **The HPC Environment:** We execute SpMM across supercomputer clusters (e.g., Summit, Frontier) where nodes do not share RAM. Data must be explicitly sent over the network (MPI).
- **The $\alpha - \beta$ Communication Model:** The time to send a message of size L is modeled as $T_{comm} = \alpha + \beta L$, where α is network latency and β is bandwidth inverse.
- **The Bottleneck:** For SpMM, the computation time is extremely fast, but transferring the dense matrix B triggers high βL penalties. Thus, **communication volume** dictates overall scaling efficiency.

4. The Traditional 1D Partitioning Approach

The SpMV Legacy Strategy

Standard libraries (like PETSc or Trilinos) treat SpMM exactly like SpMV, looping the operation n times without structural changes.

- **1D Row-wise Slicing:** The sparse matrix A is sliced horizontally into blocks of size $(m/p) \times k$, where p is the number of processors.
- **Independent Computation:** Each processor P_i computes its local chunk of the output matrix C_i independently based on its assigned rows.
- **Unpartitioned Dense Matrix:** The dense matrix B is completely unpartitioned in this setup. If a processor needs elements from column j of B, it is forced to fetch the entire row of B across the network.
- **Resulting Inefficiency:** This leads to heavy, redundant data transfers across the distributed system, which scales poorly as the number of dense vectors (n) increases.

5. Limitations of the 1D Approach for SpMM

- **Ignoring the n-Dimension:** The 1D approach only parallelizes across the rows of A (m-dimension). It completely ignores the n columns of matrix B.
- **Redundant Data Transfer:** If Processor 1 needs elements from column j of B, it must fetch the *entire row* of B across the network.
- **Linear Scaling Penalty:** The communication volume scales linearly with n. If we increase the batch size from 1 vector to 128 vectors, the network traffic becomes $128\times$ heavier, causing severe network congestion and CPU starvation.

6. Visualizing the 3D Iteration Space

To rethink parallelization, the authors model SpMM fundamentally as a 3D Iteration Space Cuboid representing the nested loops of matrix multiplication.

- m-**axis (Outer loop):** Iterating over the rows of sparse matrix A.
- k-**axis (Inner loop):** Iterating over the columns of A and rows of B (the dot product reduction axis).
- n-**axis (Batch loop):** Iterating over the columns of the dense matrix B.

The Goal of Parallelization

Parallelizing the algorithm mathematically means dividing this $m \times k \times n$ cuboid into sub-blocks and assigning each block to a different processor to minimize data dependency edges.

7. Exploiting the Missing n-Dimension

- **The Core Insight:** Treating the n-dimension (multiple vectors) as a partitionable axis unlocks a massive optimization opportunity.
- **Why it works:** By assigning different columns of B to different processors, we inherently divide the workload of transferring the dense matrix.
- **The Threshold:** When the number of dense vectors (n) is sufficiently large, partitioning **both** the sparse matrix and the dense vectors yields a mathematically smaller communication volume than just partitioning the sparse matrix alone.

8. Proposed Solution: 2D *mn*-Parallelization

Instead of a 1D grid ($p \times 1$), the paper proposes organizing the processors into a **2D Process Grid** of size $p_m \times p_n$.

- p_m**:** The number of process groups dividing the rows of A.
- p_n**:** The number of process groups dividing the columns of B.
- **Constraint:** The total processor count $p = p_m \times p_n$.

Why not full 3D?

A full 3D partitioning ($p_m \times p_n \times p_k$) involves splitting the k-axis (dot products). This forces expensive ‘MPI$_A$*llreduce*‘*operations*.*The*2*D*mn−*gridavoidsth*

9. Conceptual Comparison: 1D vs. 2D

Let's examine how communication shifts when processing 128 dense vectors on 16 processors:

- **1D Setup (16×1):** Processor P_1 calculates its rows for *all 128 vectors*. If it needs a remote element, it fetches 128 floats over the network.
- **2D Setup (8×2):** We split the processors into an 8×2 grid. Processor $P_{1,1}$ calculates its rows for only the *first 64 vectors*.
- **The Benefit:** $P_{1,1}$ now only fetches 64 floats. Network traffic for dense matrix transfers is immediately cut in half, alleviating interconnect bottlenecks.

[Image comparing 1D and 2D grid partitioning in parallel computing]

10. Introducing CRP-SpMM

Communication-Reduced Parallel SpMM (CRP-SpMM)

CRP-SpMM is an algorithmic framework designed to automatically determine the optimal $p_m \times p_n$ grid geometry for any given sparse matrix before execution begins.

- **Sparsity-Aware:** It analyzes the non-zero structure of matrix A (e.g., highly diagonal vs. randomly scattered).
- **Topology-Aware:** It factors in the physical number of nodes and network characteristics.
- **Automated Decision Making:** It calculates exactly when shifting from a 1D grid to a 2D grid is profitable, dynamically configuring the MPI sub-communicators.

11. Mathematical Cost Modeling

To decide the best grid geometry, CRP-SpMM uses a rigorous cost model to estimate total communication volume V:

$$V(p_m, p_n) = V_{dense}(p_m, p_n) + V_{sparse}(p_m, p_n)$$

- V_{dense}**:** The cost to route the dense matrix B. This volume is proportional to $(p_m - 1) \times n$. *(Decreases as p_m shrinks).*
- V_{sparse}**:** The cost to replicate parts of the sparse matrix A across the p_n processor groups. Proportional to $(p_n - 1) \times nnz(A)$. *(Increases as p_n grows).*

12. The Core Trade-off: Dense vs Sparse Transfer

- **Pushing towards p_m (1D Limit):** Minimizes the replication of A ($V_{sparse} \approx 0$), but maximizes the heavy traffic of dense matrix B.
- **Pushing towards p_n (Extreme 2D):** Drastically reduces the communication of B, but forces the system to broadcast millions of sparse indices (nnz) over the network.
- **The Solution:** The algorithm must find the saddle point on the cost curve where the sum of both penalties is minimized.

Graph Partitioning

The paper utilizes Hypergraph partitioners (like Zoltan or ParMETIS) to accurately predict V_{dense} based on the exact adjacency map of the matrix.

13. The Greedy Grid Optimization Algorithm

Checking every possible $p_m \times p_n$ combination using a heavy hypergraph partitioner is computationally prohibitive. CRP-SpMM uses a **Greedy Search Heuristic**:

1. **Initialization:** Assume the standard 1D partition: $p_m = p, p_n = 1$.
2. **Iteration:** Systematically double p_n (e.g., $1 \to 2 \to 4$) and correspondingly halve p_m.
3. **Evaluation:** At each step, merge contiguous row blocks of A. Recalculate the cost formula $V = V_{dense} + V_{sparse}$.
4. **Termination:** Stop the loop as soon as the calculated cost V begins to increase, identifying the local minimum communication point.

14. Time and Space Complexity Analysis

Efficiency of the Greedy Approach

The greedy algorithm reduces the parameter search space from $O(p)$ to $O(\log p)$. The setup phase takes mere milliseconds, even on supercomputers with thousands of nodes.

- **Space Complexity (Memory Limits):** The algorithm explicitly ensures local memory bounds are respected. Memory per node scales via $\mathcal{O}\left(\frac{nnz(A)}{p_m} + \frac{k \cdot n}{p_n}\right)$, preventing Out-Of-Memory (OOM) crashes.
- **Scalability:** Because it iteratively merges contiguous partitions, it reuses previous mapping states rather than calling the hypergraph partitioner from scratch on every loop.

15. Synergy with Node-Level Hardware Optimizations

- **Orthogonal Optimization:** CRP-SpMM solves the *distributed network* problem. This makes it completely independent from, and synergistic with, local CPU/GPU hardware optimizations.
- **Local Execution:** Once the data safely arrives at the correct MPI rank, local BLAS libraries (like Intel MKL or cuSPARSE) execute the math using:
  - Cache-blocking (optimizing L1/L2 hits).
  - CSR/CSC (Compressed Sparse Row/Column) memory layouts.
  - SIMD Vectorization (AVX-512) or GPU Tensor Cores.

16. Experimental Performance Results

The empirical evaluation of CRP-SpMM across distributed HPC clusters demonstrates profound improvements over state-of-the-art baselines:

- **Communication Reduction:** For matrices with highly irregular sparsity patterns (e.g., web-crawl graphs, social networks), total communication volume drops by factors of **2x to 5x**.
- **Execution Speedup:** End-to-end execution time of the SpMM kernel decreases drastically compared to 1D implementations in PETSc or Trilinos.
- **Strong Scaling:** The 2D *mn*-grid allows SpMM to scale effectively to tens of thousands of processor cores without hitting the dreaded "communication wall."

Thank You!

Any Questions?

Prashant Singh
Roll No: 123cs0070
Parallel and Distributed Computing