A Unified Multi-Dimensional Parallel Algorithm for Communication-Avoiding SpMM

Qi Wang[1,§], Shengguo Li[1,§], Xiaojian Yang[1], Deshun Bi[1], Chen Wang[2,3], Song Wang[3], Tiejun Li[1,*], Yujuan Tan[1,*], Bo Yang[1,*]

[1]*College of Computer Science and Technology, National University of Defense Technology, China*

[2]*National Engineering Research Center for Big Data Software, Tsinghua University, Beijing, China*

[3]*Energy and Internet Research Institute, Tsinghua University, Chengdu, China.*

Emails: {wqikiki, nudtlsg, yangxj, bds123, 5tjli, tanyujuan, yangbo}@nudt.edu.cn

wang_chen@tsinghua.edu.cn, wangsong@tsinghua-eiri.org

***Abstract*—Distributed sparse matrix–dense matrix multiplication (SpMM) is often bottlenecked by communication and performs poorly under irregular sparsity patterns. We propose a unified task–data parallel (TDP) framework that jointly models task decomposition and data distribution, enabling communication analysis across the SpMM design space. Within this framework, we identify two complementary 2D submodels, mn-r_Ar_B and kn-c_Ar_C, with distinct sparsity-dependent communication costs, and develop a fused MNKN algorithm that partitions the sparse matrix into tiles and assigns each tile to the lower-cost submodel using lightweight sparsity-aware metrics. This design enables fine-grained dense-side communication avoidance without additional storage overhead while naturally overlapping communication and computation. Experiments on large-scale SuiteSparse matrices show performance gains on 68% of 135 matrices, including up to an order-of-magnitude speedup over 2D SUMMA and an average speedup of $2.4\times$ over the sparsity-aware CRP baseline on ten representative matrices. We further demonstrate its effectiveness in polynomial-filtering eigensolvers.**

***Index Terms*—Communication-Avoiding, SpMM, Distributed Parallel, Task-Data Parallel**

I. Introduction

Sparse matrix–dense matrix multiplication (SpMM) is a core primitive in scientific computing and machine learning. Given $C = A \times B$ with a sparse matrix $A \in \mathbb{R}^{m\times k}$ and dense matrices $B \in \mathbb{R}^{k\times n}$ and $C \in \mathbb{R}^{m\times n}$, SpMM arises in block Krylov solvers [1], [2] and graph neural network (GNN) training [3], [4]. Like dense matrix multiplication (GEMM), SpMM is dominated by dense computation and communication costs [5], [6], but its dense accesses are governed by the irregular sparsity pattern of A, which makes communication modeling and parallel configuration selection substantially more difficult [7], [8].

Existing parallel SpMM methods broadly fall into two categories. The first reduces SpMM to sequences of SpMVs [9]–[11], leveraging graph and hypergraph partitioning techniques [12]–[14]. These methods are widely used on shared-memory platforms, where A is partitioned into row blocks [15] or 2D tiles [16], [17], but they provide limited guidance on distributing the dense matrix B or organizing its communication. The second category adapts distributed GEMM to SpMM using multi-dimensional process grids with inner- or outer-product decompositions. This includes *sparsity-agnostic* approaches based on Cannon and SUMMA [18], [19], as well as *sparsity-aware* approaches that align dense-matrix communication with the nonzero structure of A [5], [20]–[23]. However, sparsity-aware communication makes the analysis of both communication and computation costs substantially more complex.

Recent distributed sparse kernels are no longer confined to a single fixed parallel model; instead, they increasingly combine multiple decomposition/communication paths and even transfer ideas across kernels. Several studies already reflect this trend: TS-SpGEMM combines a 1D design with an additional communication dimension to reduce traffic [24], and FusedMM [25] uses an α–β–γ model to select among multiple candidate 2D models with different data distributions. This evolution makes the model-space description itself a central research problem: we need a representation that can compare, compose, and switch among heterogeneous strategies under irregular sparsity. The CRP model [6] captures task decomposition with a 3D cuboid and classifies 1D/2D/3D schemes [26], [27], but it does not explicitly encode data distribution, which is crucial for communication behavior and thus limits its ability to reason about multi-model fusion. Other studies classify the design space from complementary data-distribution perspectives, including dense/sparse-shifting based on replication and communication patterns [25], storage dimensionality (1D–3D) combined with blocked or collective layouts [28], and A/B/C-stationary schemes [20]. These complementary yet disconnected taxonomies motivate our unified framework, which systematically integrates prior schemes.

In this paper, we propose a **task–data parallel (TDP)** model that unifies task decomposition and data distribution within a single analytical framework. The TDP model characterizes distributed SpMM using both a three-dimensional task space (m, k, n) and a six-dimensional data-distribution descriptor, enabling more precise communication analysis across a broad range of parallelization strategies. Within this framework, we identify two complementary 2D submodels, mn-r_Ar_B and kn-c_Ar_C, whose communication costs depend

§Equal contribution.

*Corresponding author.

SC26, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-4789-7/26/$31.00 ©2026 IEEE

differently on the sparsity structure of A. Based on this observation, we introduce a **fused MNKN framework** that partitions the sparse matrix into tiles and dynamically assigns each tile to the more communication-efficient submodel using lightweight sparsity-aware metrics. This design enables fine-grained *dense-side* communication avoidance for B/C without changing the sparse A traffic or adding storage overhead, while also exposing opportunities to overlap dense B/C transfers with computation.

We evaluate the proposed TDP kernels on up to 128 nodes using 135 SuiteSparse matrices [29] with diverse sparsity patterns. The method reduces dense-side communication by 16.3% on average, with reductions exceeding 90% on some matrices, and scales to dense matrices with up to 8192 columns. Compared with CRP, CombBLAS SUMMA [30], and SpComm3D [22], TDP-$mnkn$ achieves up to an order-of-magnitude speedup over 2D SUMMA and an average of $2.4\times$ end-to-end speedup over the sparsity-aware CRP baseline on ten representative matrices. We also demonstrate practical value in polynomial-filtering eigensolvers, where reusable execution plans deliver $1.37\times$–$1.61\times$ speedups.

We summarize our contributions as follows:

- **Unified TDP Framework.** We propose a task-data parallel model that jointly captures computation and data distribution, enabling systematic sparsity-aware communication analysis across the distributed SpMM design space.
- **Fused MNKN Algorithm.** We develop a tile-based SpMM algorithm that adaptively combines mn-$r_A r_B$ and kn-$c_A r_C$ to achieve fine-grained communication avoidance without extra storage, while naturally overlapping dense communication with computation.
- **Performance and Applicability.** We demonstrate dense-side communication reduction and substantial runtime gains, including up to an order-of-magnitude improvement over 2D SUMMA and effective reuse in polynomial-filtering eigensolvers.

II. THE DESIGN SPACE OF PARALLELIZING SPMM

We study the parallelization of sparse matrix–dense matrix multiplication (SpMM),

$$C = A \times B, \quad A \in \mathbb{R}^{m\times k},\ B \in \mathbb{R}^{k\times n},\ C \in \mathbb{R}^{m\times n}, \quad (1)$$

where A is sparse and B and C are dense. The dense index

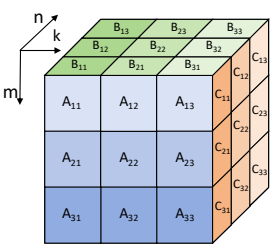
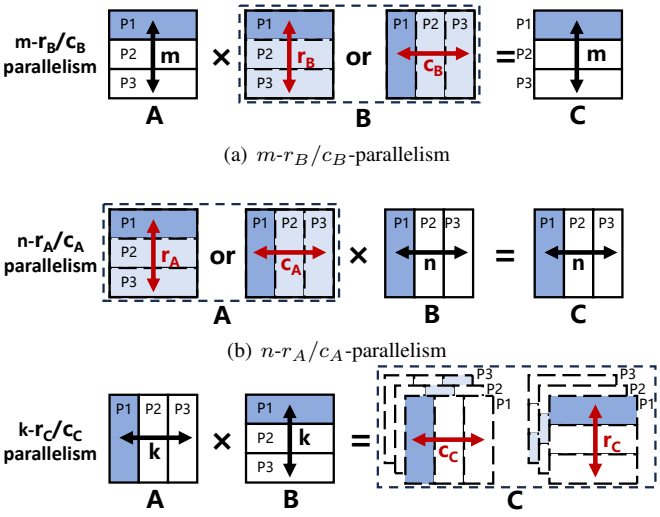


Fig. 1. Three-dimensional task space (m, k, n) for SpMM.

space underlying SpMM contains $O(mkn)$ potential multiply–accumulate points and can be viewed as a three-dimensional



(a) m-r_B/c_B-parallelism

(b) n-r_A/c_A-parallelism

(c) k-r_C/c_C-parallelism

Fig. 2. Three fundamental 1D parallelization schemes and their corresponding data distribution choices. Subfigures (a), (b), and (c) illustrate m-, n-, and k-**parallelism**, respectively, for distributed SpMM with $p = 3$. Dark blue shading denotes locally stored data, while light blue blocks require inter-process communication. Black arrows indicate task-parallel dimensions along which no communication occurs; red arrows denote data-partitioning dimensions along which matrices are communicated.

task space (m, k, n), as shown in Figure 1. Each point in this space corresponds to one multiply–accumulate operation, and each entry C_{ij} is obtained by accumulating contributions along the k dimension.

A. Task Parallelization of SpMM

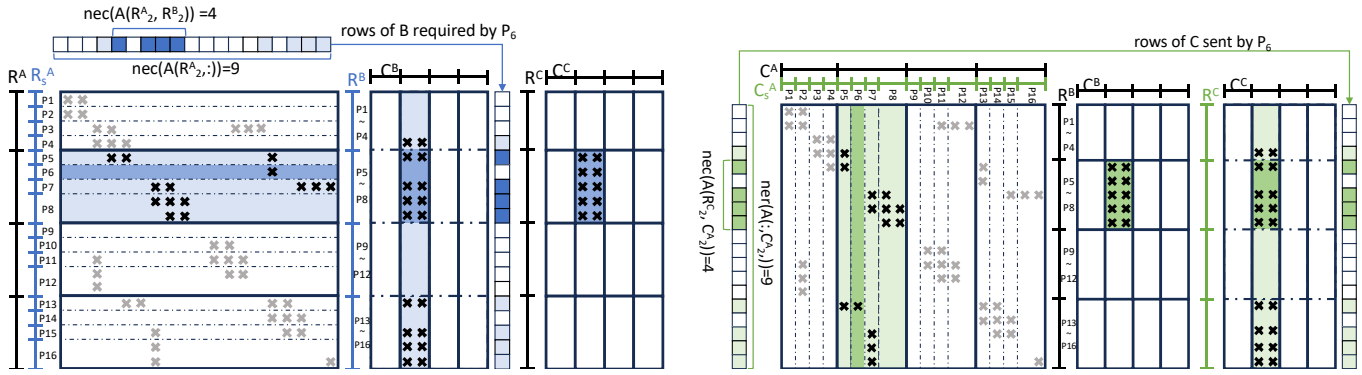
Parallelizing SpMM can be viewed as partitioning the (m, k, n) task space along one or more dimensions in Figure 1 into sub-tasks and assigning them to different processes, which forms 1D/2D/3D task parallelism schemes.

Figure 2 illustrates the three basic 1D schemes. We use the strip-partitioned matrices to represent this partitioning. For 1D m**-parallelism**, this scheme determines that matrices A and C are partitioned by rows (see Figure 2(a)). Each process must gather the full B matrix prior to performing the calculation. n**-parallelism** assigns disjoint column blocks of B and C to p_n processes, so the entire matrix A must be gathered. k-**parallelism** assigns each process a disjoint column block of A and a row block of B to produce a partial contribution to C.

The recent CRP work [6] addresses task parallelization. While this purely *task*-based abstraction captures many distributed SpMM formulations, it still leaves a large design space for communication strategies. As shown in Figure 2(a), there are two feasible layouts for matrix B under m-parallelism: a row-wise layout (r_B) and a column-wise layout (c_B). These choices lead to fundamentally different communication patterns and costs.

B. Task-Data Parallel SpMM Model

We propose a *task-data* parallel (TDP) model for distributed SpMM. We define data distribution using a set of six pa-



(a) mn-r_Ar_B distribution (b) kn-c_Ar_C distribution

Fig. 3. Data layouts and execution paths for mn-r_Ar_B (left) and kn-c_Ar_C (right) on a $(p_r, p_c) = (4, 4)$ grid (A on a 1D layout). Processes are numbered left to right within each row of (p_r, p_c): $P(1,:) = \{P_1, \ldots, P_4\}$. Dark blocks are owned data; light blocks are obtained by communication; arrows follow gather-A then dense B/C movement. Dense traffic follows nonempty columns of A in (a) and rows in (b).

TABLE I
MAIN SYMBOLS USED IN COMMUNICATION MODELING.

Symbol	Meaning
p_m, p_k, p_n	Process-grid sizes along m, k, and n dimensions
r_X, c_X	Partition type of matrix X: row/column
R^X, C^X	Row/column block groups of matrix X
I_i^X, J_i^X	The i-th row/column index set of matrix X
$nnz(\cdot)$	Number of nonzero entries
$nec(\cdot)$	Number of nonempty columns
$ner(\cdot)$	Number of nonempty rows
N_{tile}	Number of tiles along the row/column dimension
n	Number of columns of dense matrices B, C
W_A, W_B, W_C	Communication volumes of matrices A, B and C
S_{rel}	Communication-saving ratio

rameters, $\{r_A, c_A, r_B, c_B, r_C, c_C\}$. Here, r_X and c_X denote the selected partitioning dimensions of matrix X across processes: r_X denotes row-wise partitioning, whereas c_X denotes column-wise partitioning. Each matrix $X \in \{A, B, C\}$ is partitioned independently along rows (r_X), columns (c_X) or mixed (r_X,c_X), such that the matrices are **stored exactly once** across all processes before and after computation.

The mapping from these partitions to p_r row processes and p_c column processes is defined as

$$R^X = \{I_1^X, \ldots, I_{p_r}^X\}, \qquad C^X = \{J_1^X, \ldots, J_{p_c}^X\},$$

where I_i^X and J_j^X are the row and column index sets of matrix X and R_i^X denotes the i-th row block of X.

We adopt MATLAB-style slice notation for index sets. $(i\!:\!j)$ denotes a contiguous index range from i to j (inclusive), and ":" denotes all indices along a given dimension. For example, $R_i^A = I_i^A = \{1\!:\!4\}$ denotes rows 1–4 of A.

Table I summarizes the main symbols. Using this notation, we next reformulate distributed SpMM across different parallelization dimensions.

1) 1D Task-Data Parallel Model: In the 1D model, processes are arranged along a single dimension, and matrices A, B, and C are each partitioned along one dimension in row(r_X) or column(c_X), forming a triple of data-partition choice. For example, Figure 2(a) shows two schemes under m-r_B/c_B, namely m-r_B and m-c_B, with data distributions (r_A, r_B, r_C) and (r_A, c_B, r_C), respectively. Since $A \in \mathbb{R}^{m\times k}$ and $C \in \mathbb{R}^{m\times n}$, the m dimension corresponds to the rows of A and C and thus determines the r_A and r_C partitions, while r_B/c_B indicates that the parallelization requires communicating row/column blocks of B.

Applying the same rule to n- and k-parallelism yields six 1D task–data models:

- m-r_B/c_B: communication of B, with A and C partitioned by rows (r_A, r_C);
- k-r_C/c_C: communication of partial sums of C, with B partitioned by rows (r_B) and A by columns (c_A);
- n-r_A/c_A: communication of A, with B and C partitioned by columns (c_B, c_C).

2) 1.5D/2D Task-Data Parallel Model: In 1.5D/2D settings, processes are arranged in a 2D process grid (p_r, p_c). We denote by $P(i,:)$ the set of processes in the i-th row process group and by $P(:,j)$ the set of processes in the j-th column process group. By viewing each process row/column as a group, 2D grid parallelization reduces to mutually nested 1D schemes within groups: as in Figure 3(a), mn-r_Ar_B combines 1D m– and n–parallelism and communicates A in process group $P(2,:) = \{P_5, P_6, P_7, P_8\}$ and then B in $P(:,2) = \{P_2, P_6, P_{10}, P_{14}\}$.

The data-partition configuration can be interpreted as a six-tuple formed by combining two 1D model triples. For example, combining m-r_B and n-c_A yields mn-c_Ar_B, with tuple

$$(r_A, r_B, r_C) \times (c_A, c_B, c_C) = (r_A, c_A, r_B, c_B, r_C, c_C).$$

In this case, the selected dimensions induce both row and column partitions on each involved matrix, resulting in a checkerboard-like layout corresponding to a true 2D data distribution.

However, not every combination of two 1D partitions for A, B, and C produces a fully checkerboard distribution. When a matrix retains an effectively one-dimensional distribution while task parallelism is two-dimensional, we classify the scheme as 1.5D. As illustrated in Figure 3(a), consider the

TABLE II
1D AND 2D TASK–DATA PARALLEL SCHEMES IN THE SPMM DESIGN SPACE.

Data Layout	1D		Data Layout	2D-B&C		1D-B 2D-C		1D-C 2D-B		1D-B&C
	Submodel	Comm. Data		1D-A	2D-A	1D-A	2D-A	1D-A	2D-A	2D-A
1D	m-r_B m-c_B	B Rows (r_B) B Cols (c_B)	2D	mn-r_Ar_B	mn-c_Ar_B	mn-r_Ac_B	mn-c_Ac_B			
	k-r_C k-c_C	C Partial Sum (r_C) C Partial Sum (c_C)		kn-c_Ar_C	kn-r_Ar_C			kn-c_Ac_C	kn-r_Ac_C	
	n-r_A n-c_A	A Rows (r_A) A Cols (c_A)			mk-c_Bc_C		mk-r_Bc_C		mk-c_Br_C	mk-r_Br_C

process grid (p_m, p_n). Combining m-r_B and n-r_A with data distributions (r_A, r_B, r_C) and (r_A, c_B, c_C) yields the six-tuple

$$(r_A, r_A, r_B, c_B, r_C, c_C),$$

which contains two occurrences of r_A with different roles. The first r_A corresponds to the partition $R^A = \{I_1, \ldots, I_{p_m}\}$ induced by m-parallelism, which defines the row-wise decomposition of the SpMM task. Accordingly, each row process group $P(i,:)$ gathers the block $A(R_i^A,:)$ before local computation. The second r_A arises from n-parallelism, which further partitions each row block $A(R_i^A,:)$ into p_n sub-blocks, forming a finer partition

$$R_s^A = \{I'_1, \ldots, I'_{p_m p_n}\},$$

distributed across all $p_m p_n$ processes.

For example, consider process P_6 in Figure 3(a). It initially stores $A((R_s^A)_6,:)$ and must gather the larger block $A(R_2^A,:)$ within the row process group $P(2,:) = \{P_5, P_6, P_7, P_8\}$, where $(R_s^A)_6 = I'_6 = \{6\}$ and $R_2^A = \{5{:}9\}$.

Table II summarizes the 1D and 2D schemes in the TDP design space. The 1.5D/2D schemes arise from pairwise combinations of distinct 1D parallelization strategies (combinations of identical task-parallel schemes are excluded), for a total of 12 schemes. Scheme names are formed from the constituent 1D schemes to indicate the communication pattern; e.g., mn-r_Ar_B denotes communication of the row blocks of A and B. Within each table row, schemes share the same task-parallelization strategy; within each column, they share the same data-partitioning dimension.

This *task-data* framework captures the characteristics of different data-distribution schemes within a unified description. To our knowledge, this is the first model that can systematically distinguish 1.5D and 2D SpMM algorithms from a parallelization viewpoint.

3) 2.5D/3D Task-Data Parallel Model: Layered 3D SpMM can be understood as extending a 2D process grid with an additional layer dimension of size c. Across these c layers, selected inputs (typically A and/or B) are replicated, so each layer performs only about $1/c$ of the original work and communicates only about $1/c$ of the corresponding intra-layer communication volume. This alleviates the communication bottleneck of a single 2D layer. Under this classification, 2.5D and 3D methods are conceptually similar and mainly differ in the number of layers. As the dimensionality increases, balancing memory footprint, communication volume, and computational load becomes substantially more difficult.

C. Communication Volume of TDP mn-r_Ar_B and kn-c_Ar_C

We next derive the TDP communication costs of mn-r_Ar_B and kn-c_Ar_C, the two 1.5D schemes that underlie the fused algorithm in Section III-B. We instantiate them on 2D process grids (p_m, p_n) and (p_k, p_n), respectively. Their data layouts and communication patterns are illustrated in Figure 3, where dark shading denotes data in the initial layout and lighter shading denotes blocks obtained through intra-group communication.

In both models, processes within each row process group $P(i,:)$ first communicate to assemble the required strip of A: a full row block $A(R_i^A,:)$ for Figure 3(a), or a column block $A(:, C_i^A)$ for Figure 3(b). The two schemes then differ in dense-data movement. For mn-r_Ar_B, once $A(R_i^A,:)$ is available, each process identifies the required rows of B from its local sparsity pattern and fetches them along the process column $P(:,j)$, followed by local SpMM to produce C. For kn-c_Ar_C, each process already owns column blocks of B and C; after aligning A, it performs local SpMM to generate partial sums in $C(:, C_i^C)$, which are then transferred across processes. In short, both schemes first gather A. mn-r_Ar_B then moves B before SpMM, while kn-c_Ar_C moves the partial sums of C after SpMM.

We denote the number of nonzeros by $nnz(X)$, the number of nonempty columns by $nec(X)$, and the number of nonempty rows by $ner(X)$. For communication metrics, W_X denotes the total communication volume associated with matrix X. The following expressions quantify the resulting sparsity-aware communication volumes.

First, in both schemes, each sparse matrix block of A within a row process group $P(i,:)$ is copied to the remaining $p_n - 1$ column positions, so

$$W_A^{(mn)} = W_A^{(kn)} = (p_n - 1) \cdot nnz(A). \tag{2}$$

For mn-r_Ar_B, each block of B is copied within the column process group that owns C_i^B to the other $p_m - 1$ row positions. Each row block R_i^A requests only the rows of B indexed by the nonempty columns of $A(R_i^A,:)$. Among them, rows already stored in the local block $B(R_i^B, C_j^B)$ do not require communication, so the communicated rows are

$$nec(A(R_i^A,:)) - nec(A(R_i^A, R_i^B)).$$

Each row involves communication across n/p_n columns; calculating the volume for all $p_m \times p_n$ processes yields

$$\sum_{i=1}^{p_m} \sum_{j=1}^{p_n} \Big(nec(A(R_i^A, :)) - nec(A(R_i^A, R_i^B)) \Big) \cdot \frac{n}{p_n}.$$

Eliminating the summation over p_n yields

$$W_B^{(mn)} = \sum_{i=1}^{p_m} \Big(nec(A(R_i^A, :)) - nec(A(R_i^A, R_i^B)) \Big) \cdot n. \quad (3)$$

For kn-$c_A r_C$, the communicated dense object changes from rows of B to rows of partial sums in C. For each process block within C_i^A, local SpMM generates row support

$$ner(A(:, C_i^A)).$$

Rows that already belong to the local destination block R_i^C are retained locally, while the remaining rows are communicated for reduction. Hence,

$$W_C^{(kn)} = \sum_{i=1}^{p_k} \Big(ner(A(:, C_i^A)) - ner(A(R_i^C, C_i^A)) \Big) \cdot n. \quad (4)$$

The mn-$r_A r_B$ and kn-$c_A r_C$ schemes are sensitive to the column and row sparsity patterns of A, respectively. When A is symmetric and $R^A = C^A$ and $R^C = R^B$, the two schemes incur identical communication, i.e., $W_B = W_C$. For asymmetric sparsity, however, the two costs generally differ, i.e., $W_B \neq W_C$. The proposed TDP cost model therefore predicts which scheme is preferable for a given sparsity pattern. Due to space constraints, we omit the cost analysis of other models; instead, related approaches are unified under the TDP framework in Section V.

III. Model and Combination

A. Combination Motivation

The 1D TDP model is limited to single-dimensional communication, whereas 2.5D and 3D models are substantially more complex. Accordingly, this paper focuses on the 1.5D/2D TDP setting, where communication occurs across multiple process-group dimensions. Matrix A is naturally stored in compressed formats (CSR/CSC), making 1D partitioning of A easier to implement and maintain than checkerboard layouts. In iterative SpMM workloads with evolving dense blocks (e.g., block Krylov methods and polynomial filtering), B and C must share a compatible dense layout so that each output can be reused as the next input without redistribution. Among the 1.5D schemes analyzed in Section II-C, mn-$r_A r_B$ and kn-$c_A r_C$ meet this constraint. We next analyze how their dense-side volumes depend on local sparsity and how their execution pipelines complement each other, which jointly motivate a fused TDP-$mnkn$ design.

The dense-side communication volumes depend on $nec(\cdot)$ and $ner(\cdot)$ for each matrix block of A. Both W_B and W_C in Eqs. (3)–(4) traverse the entire sparse matrix; for example, W_B iterates over all p_m row blocks. If $A(R_i^A, :)$ and $A(:, C_j^A)$ are further partitioned into smaller sub-blocks, their local $nec(\cdot)$ and $ner(\cdot)$ values may differ, as illustrated by the dark-blue and dark-green sub-blocks in Figure 4. When $nec(\cdot) < ner(\cdot)$, the mn scheme, which communicates B, is preferable; when $nec(\cdot) > ner(\cdot)$, the kn scheme, which communicates partial sums of C, is more efficient. Therefore, selecting the better scheme for each sub-block can reduce communication volume.

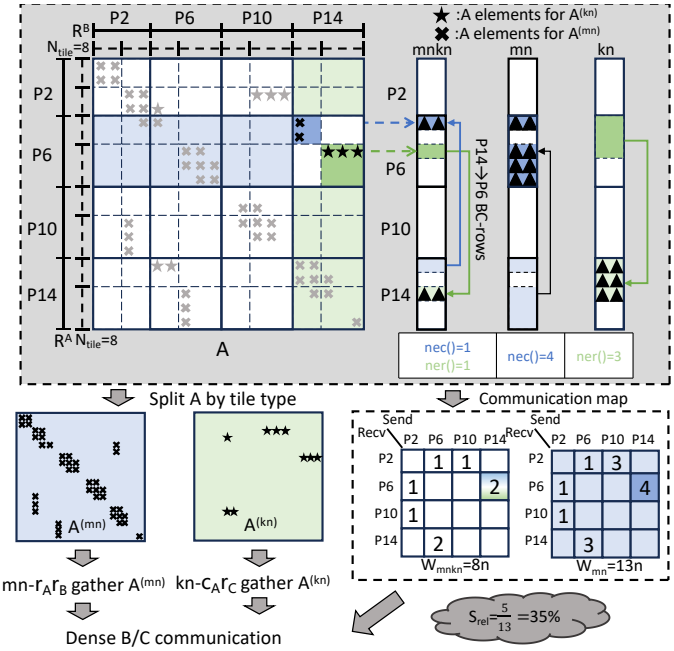


Fig. 4. Illustration of tile-based splitting and communication reduction in TDP-mnkn. The left panel shows Step 1 of Algorithm 2, where a square sparse matrix A is partitioned into an $N_{\text{tile}} \times N_{\text{tile}}$ tile grid. Tiles are classified by Eq. (7) and split into the tile-disjoint components $A^{(\text{mn})}$ and $A^{(\text{kn})}$, which are assigned to TDP-mn and TDP-kn, respectively. The right panel shows the communication maps generated from $\text{nec}(\cdot)$ and $\text{ner}(\cdot)$ statistics over (R^A, R^B)-partitioned subblocks. The fused TDP-mnkn schedule avoids redundant B- and C-row communication by classifying tiles to kn-$c_A r_C$ and mn-$r_A r_B$.

From an execution viewpoint, mn-$r_A r_B$ follows a communicate-B-then-compute pipeline, whereas kn-$c_A r_C$ follows a compute-then-communicate-C pipeline. These complementary orderings naturally expose opportunities to overlap communication with computation. As shown in Figure 5, the two pipelines can be interleaved to shorten the critical path. This observation further motivates us to partition the sparse matrix into two groups of tiles according to $nec(\cdot)$ and $ner(\cdot)$.

B. Communication-Avoiding TDP-$mnkn$ SpMM

To integrate the two algorithms, we execute them on a common grid as follows:

$$(p_r, p_c) = (p_m, p_n) = (p_k, p_n),$$

so that communication of matrix A remains within row process groups, while communication of matrices B and C occurs within column process groups. As illustrated in Figure 3, due to data-alignment requirements of $C(R_i^C, C_j^C) = A(R_i^A, C_k^A) \times B(R_k^B, C_j^B)$, combination of mn-$r_A r_B$ and kn-$c_A r_C$ satisfies the following constraints

$$R^A = R^C, \quad C^A = R^B, \quad C^B = C^C.$$

We further require $R^B = R^C$ so that the input block $B(R_i^B, C_i^B)$ and the output block $C(R_i^C, C_j^C)$ have the same local layout on process $P(i,j)$ enabling direct reuse of C as the next input in iterative algorithms. This alignment also ensures that Eqs. (3) and (4) use the same block decomposition of A.

Based on this fusion strategy, we derive the TDP-$mnkn$ algorithm by splitting A into two disjoint sparse components:

$$A = A^{(mn)} + A^{(kn)}, \tag{5}$$

where

$$\operatorname{supp}(A^{(mn)}) \cap \operatorname{supp}(A^{(kn)}) = \varnothing,$$
$$\operatorname{supp}(A^{(mn)}) \cup \operatorname{supp}(A^{(kn)}) = \operatorname{supp}(A).$$

For any dense input B, linearity of SpMM gives

$$C = AB = A^{(mn)}B + A^{(kn)}B \equiv C^{(mn)} + C^{(kn)}. \tag{6}$$

Equation (6) shows that the two branches produce complementary partial contributions to the same output matrix C, which makes fused execution possible.

To realize this decomposition, as shown in the left panel of Figure 4, we partition checkerboard blocks $A_{ij} = A(R_i^A, R_j^B)$ (divided for communication map) into tiles A_{xy} and assign each tile to either mn-$r_A r_B$ or kn-$c_A r_C$. Let

$$A_{xy} = A(R_x^A, C_y^A), \qquad x, y \in \{1, 2, \ldots, N_{tile}\},$$

denote the splitting of a sparse tile where $N_{tile} > p_r$ and N_{tile} controls the finer-grained splitting of each process-grid block of A. We classify each tile by comparing its numbers of nonempty rows and nonempty columns:

$$\begin{aligned} A^{(\text{mn})} &= \sum_{ner(A_{xy}) > nec(A_{xy})} A_{xy}, \\ A^{(\text{kn})} &= \sum_{ner(A_{xy}) < nec(A_{xy})} A_{xy}. \end{aligned} \tag{7}$$

When $ner(A_{xy}) = nec(A_{xy})$, we apply a round-robin rule to distribute such tiles evenly between the two branches for load balance. For tiles in diagonal block $A(R_i^A, R_j^B)$ with $i = j$, we assign them to the $A^{(\text{mn})}$ with priority over round-robin for efficient local computation. The proposed method naturally extends to non-square matrices A by allowing the maximum tile indices, N_{tile} along the x- and y-directions to differ, thereby maintaining square tile partitions.

To keep the model and implementation consistent, we explicitly control the tile count by N_{tile}. We start from $N_{tile} = p_r$, that is, tile rows are initially aligned with process-row groups. Based on Eqs. (2)–(4) and the decomposition in Eq. (6), the communication volume of TDP-$mnkn$ is

$$\begin{aligned} W_A^{(mnkn)} &= (p_n - 1) \cdot nnz(A), \\ W_{BC}^{(mnkn)} &= W_C\left(A^{(\text{kn})}\right) + W_B\left(A^{(\text{mn})}\right). \end{aligned} \tag{8}$$

Thus the sparse A communication volume is identical; the two schemes differ only in dense-side movement of B or partial sums of C. In particular, when $N_{tile} = p_r$, the tile partition is aligned with the process-row partition. Each coarse tile pair (i,j) can then choose the cheaper branch independently, while the tiling does not introduce additional cuts beyond the original process-row structure. Hence, the fused dense-side communication is bounded by the better of the two single-branch volumes:

$$W_{BC}^{(mnkn)} \le \min\big(W_B^{(mn)}(A),\, W_C^{(kn)}(A)\big).$$

When N_{tile} increases beyond p_r, some coarse tiles that previously mixed mn-favorable and kn-favorable regions are split into finer tiles. This typically creates an initial improvement region, because each finer tile can choose a more suitable branch. However, if N_{tile} continues to grow, $nec(\cdot)$ and $ner(\cdot)$ are evaluated on increasingly small tiles, and cross-entry reuse is gradually lost. In the extreme case, the accumulated terms approach a near nonzero-level traversal bound, so further refinement no longer improves communication and may even increase it. This trend is consistent with Figure 4: splitting initially improves branch selection, but excessive refinement eventually leads to saturation or degradation.

This achieves **dense-side** communication avoidance. We define the communication-saving ratio as

$$S_{rel} = 1 - \frac{W_{BC}^{(mnkn)}}{\min\big(W_B^{(mn)}(A),\, W_C^{(kn)}(A)\big)}. \tag{9}$$

Algorithm 1: Lightweight adaptive N_{tile} selection

Input: initial $N_{tile} = p_r$, upper bound N_{max}
Output: selected N_{tile}

$N_{tile} \leftarrow p_r$, compute $S_{rel}(N_{tile})$;
while $2N_{tile} \le N_{max}$ **do**
  $N'_{tile} \leftarrow 2N_{tile}$, compute $S_{rel}(N'_{tile})$;
  if $S_{rel}(N'_{tile}) < S_{rel}(N_{tile})$ **break**;
  else $N_{tile} \leftarrow N'_{tile}$;
end
return N_{tile};

Algorithm 1 gives a lightweight adaptive search for N_{tile}. Rather than seeking a global optimum, we target a low-cost operating point before overrefinement. Starting from $N_{tile} = p_r$, we increase N_{tile}, evaluate S_{rel}, and stop as soon as S_{rel} decreases.

As shown in Figure 4, the sparsity pattern of A guides sparsity-aware communication, leading to the same communication topology. Specifically, we partition A according to (R^A, R^B) and analyze the $nec(\cdot)$ and $ner(\cdot)$ values within each block to construct the communication pattern among processes P_2, P_6, P_{10}, and P_{14} in a column process group. The same procedure applies to the other column process groups.

Furthermore, TDP-$mnkn$ refines this block-level partition into smaller tiles. Each tile is then assigned according to whether $nec(\cdot)$ or $ner(\cdot)$ is more communication-efficient, yielding the communication map of the fused algorithm. This

Algorithm 2: Sequential TDP-$mnkn$ Kernel

Input: Sparse matrix A, dense input B, partition (R, C, R_s^A, C_s^A), process grid (p_r, p_n), $p_r = p_m = p_k$

Output: Distributed output C with $C = AB$

1) Preprocess (for fixed A, once):
compute N_{tile} using Algorithm 1, split tiles A_{xy}, and build $A^{(mn)}$ and $A^{(kn)}$;
scatter local matrices A_{ij} and B_{ij} to $P(i,j)$;
2) Ensure the required A views in $P(i,:)$:
perform allgather of $A^{(mn)}(R_i,:)$ and $A^{(kn)}(:,C_i)$;
compute MPI tags for B/C from the compressed `colidx` of $A^{(mn)}(R_i,:)$ and the compressed `rowidx` of $A^{(kn)}(:,C_i)$;
3) Execute mn in each $P(:,j)$:
foreach P_i **do**
  pack the tag-selected rows of $B(:,C_j)$ required by $P(:,j)$, $i \neq j$;
  send/recv the selected rows of B to/from $P(:,j)$, $i \neq j$;
  copy the local block B_{ij} into a contiguous local buffer $B(:,C_j)$;
  perform local SpMM $C_{ij} = A^{(mn)}(R_i,:)B(:,C_j)$;
  accumulate C_{ij} into C;
end
4) Execute kn in each $P(:,j)$:
foreach P_i **do**
  perform local SpMM $C(:,C_j) = A^{(kn)}(:,C_i)B_{ij}$;
  send/recv partial sums $C(:,C_j)$ to/from $P(:,j)$, $i \neq j$;
  copy the local block $C(R_i,C_j)$ into C;
  accumulate the received partial sums into $C(R_i,C_j)$, $i \neq j$;
end

tile-level splitting enables TDP-mnkn to avoid redundant communication.

C. Sequential TDP-$mnkn$

We first implement a *sequential* TDP-$mnkn$ algorithm that executes the two subkernels TDP-mn and TDP-kn back-to-back. Algorithm 2 first constructs the split-A views and communication metadata (steps 1–2), and then launches the full TDP-mn and TDP-kn kernels in sequence (steps 3–4). The two kernels interact only through accumulation into the shared distributed matrix C, as shown in Eq. (6). Both TDP-mn and TDP-kn follow the task-data layouts described in Section II.

We optimize the two subkernels independently. Specifically, we store $A^{(\mathrm{mn})}$ in **CSR** and $A^{(\mathrm{kn})}$ in **CSC**, enabling direct row- and column-oriented access without conversion overhead. Communication schedules are derived from compressed `colidx` (CSR) and `rowidx` (CSC), making the implementation inherently sparsity-aware.

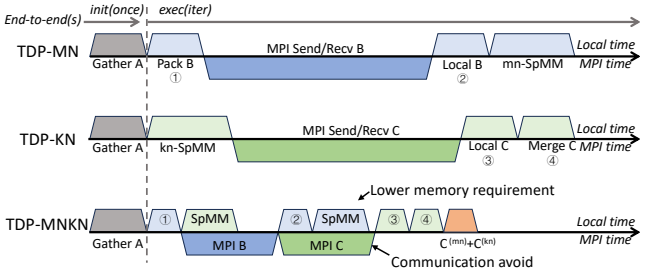


Fig. 5. Timeline of fused TDP-$mnkn$. Left of the dashed line: one-time sparse A exchange (MPI A), unchanged by fusion and introducing no extra A traffic. Right: the end-to-end fused window, where iterative dense MPI B/C transfers overlap with SpMM on the per-iteration execution track.

Phase 1 (Step 1). After tile labeling, we scatter A_{ij} and B_{ij}. The CSR/CSC structures directly extend the tile coloring and data distribution described above.

Phase 2 (Step 2). Sparsity-aware communication is driven by MPI *tags* computed from compressed indices: the column indices in CSR specify the rows of B that must be moved in the mn branch, while the row indices in CSC specify the rows of C that must be reduced in the kn branch.

Phase 3 (Step 3): TDP-mn. Because the tags are derived from $A^{(\mathrm{mn})}$, the received submatrix messages of B are row-contiguous, since $A^{(\mathrm{mn})}$ is encoded through compressed column indices. This improves both network transmission efficiency and local dense computation. It also explains why we prioritize diagonal and near-diagonal tiles in the mn branch. When constructing the local SpMM panel, we mainly insert complete contiguous received blocks, or only a few coarse strides determined by the tags, thereby avoiding fine-grained unpack/scatter overhead.

Phase 4 (Step 4): TDP-kn. The local SpMM produces partial sums in $C(:,C_j) = A^{(\mathrm{kn})}(:,C_i)B_{ij}$. Because the relevant row support is directly encoded in the compressed `rowidx` of $A^{(\mathrm{kn})}$, the rows of $C(:,C_j)$ to be reduced are already stored contiguously in memory. As a result, almost no additional packetization overhead is required during communication. These partial sums are finally reduced into the same C buffer updated in Phase 3.

In implementing TDP-mn and TDP-kn, we reduce the data-processing overhead before and after communication through index compression. This is a code-level optimization derived from the algorithmic structure. Our main focus, however, is the integration of the two kernels.

D. Fused TDP-$mnkn$

Figure 5 makes the timing split explicit: a one-time sparse A exchange (MPI A) is the same with and without fusion—no extra A traffic is introduced. The fused dense MPI B/C transfers are overlapped with local SpMM.

Before communication starts, TDP-$mnkn$ packs the dense rows of B required by TDP-mn into contiguous buffers. Each process first initiates a nonblocking exchange of B using compressed-column tags. During the B exchange, the process overlaps computation by performing the local SpMM of TDP-kn using local A and B. Once the B communication com-

Algorithm 3: Filtered Subspace iteration framework

Input: Symmetric matrix A; initial matrix $\hat{V}^{(0)}$; filtering function $\rho(x)$; interval of interest $[a, b]$
Output: Computed eigenpairs $(\hat{\lambda}_j, \hat{v}_j)$ with $\hat{\lambda}_j \in [a, b]$

1 $i = 1$;
2 **while** *not converged* **do**
3 $\quad \hat{X}^{(i)} = \rho(A)\hat{V}^{(i-1)}$;
4 $\quad Q^{(i)} = \mathrm{orth}(\hat{X}^{(i)})$;
5 $\quad [\hat{V}^{(i)}, \hat{\Lambda}^{(i)}] = \text{Rayleigh-Ritz}(A, Q^{(i)})$;
6 $\quad \hat{\lambda}_j = \hat{\Lambda}^{(i)}_{j,j},\ \hat{v}_j = \hat{V}^{(i)}_{:,j}$;
7 $\quad i = i + 1$;
8 **end**
9 **function** `Rayleigh-Ritz`(A, Q)
10 $\quad B = Q^H A Q$;
11 $\quad$ compute the eigendecomposition $B = \hat{U}\hat{\Lambda}\hat{U}^H$;
12 $\quad \hat{V} = Q\hat{U}$;
13 $\quad$ **return** $\hat{V}, \hat{\Lambda}$;
14 **end**

pletes, the local partial results of TDP-kn are communicated as partial sums of C. Meanwhile, the process executes the local SpMM of TDP-mn using the newly received rows of B. The schedule ends with a local reduction of partial sums in C.

At the same time, because matrix A is split into $A^{(mn)}$ and $A^{(kn)}$, each branch contains no more nonzeros than the original matrix. Consequently, the working sets involved in communication (for B and C) and local SpMM, as illustrated in Figure 5, are smaller than in TDP-mn or TDP-kn alone. TDP-$mnkn$ also lowers peak memory pressure during message staging and local SpMM execution.

E. Application: Accelerating Polynomial Filtering

Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ and an interval of interest $[a, b]$, let $\{\lambda_j\}_{j=1}^{n}$ denote the eigenvalues of A with corresponding eigenvectors v_j satisfying $\langle v_i, v_j\rangle = \delta_{ij}$. We seek eigenpairs (λ_j, v_j) satisfying

$$Av_j = \lambda_j v_j, \tag{10}$$

and $\lambda_j \in [a, b]$.

Subspace iteration (SI) is a classical eigensolver, but it is not well suited for interior eigenvalue problems. Polynomial filtering techniques [31] address this limitation by constructing a filter function $\rho(x)$ that amplifies target eigenvalues of A while suppressing non-target components, where $\rho(x)$ is typically chosen as a polynomial. The filtered subspace iteration framework (Algorithm 3) consists of three main steps: filtering $\rho(A)\hat{V}^{(i-1)}$ (line 3), orthogonalization (line 4), and the Rayleigh–Ritz procedure (line 5). Since $\hat{V}^{(i)}$ contains multiple vectors, the filtering stage reduces to a sequence of SpMM operations, which can be accelerated by our TDP-$mnkn$ algorithm.

Since sparse iterative and eigensolver packages such as PETSc [32], Hypre [33], and SLEPc [34] typically employ 1D row-wise partitioning, our TDP-$mnkn$ algorithm starts from a 1D-partitioned matrix layout and then executes Algorithm 2.

TABLE III
MATRICES USED IN THE FIXED-GRID, SCALING AND EIGENSOLVER EXPERIMENTS: DIMENSIONS AND COMMUNICATION-SAVING RATIO (%), SORTED IN ASCENDING ORDER OF COMMUNICATION SAVINGS. SETTINGS: $p_m = p_n = 8$, $n = 1024$. ID 11–15 ARE FOR THE POLYNOMIAL-FILTERING EIGENSOLVER EXPERIMENTS.

ID	Matrix	Rows	Cols	nnz(A)	S_{rel} (%)
1	wb-edu	9,845,725	9,845,725	57,156,537	0.00
2	Hook_1498	1,498,023	1,498,023	59,374,451	13.83
3	cage15	5,154,859	5,154,859	99,199,551	32.70
4	in-2004	1,382,908	1,382,908	16,917,053	41.83
5	ss	1,652,680	1,652,680	34,753,577	45.35
6	delaunay_n23	8,388,608	8,388,608	50,331,568	45.69
7	inline_1	503,712	503,712	36,816,170	50.35
8	FullChip	2,987,012	2,987,012	26,621,983	79.49
9	pkustk12	94,653	94,653	7,512,317	79.69
10	circuit5M	5,558,326	5,558,326	59,524,291	95.75
11	Ge87H76	112,985	112,985	7,892,195	7.25
12	Ge99H100	112,985	112,985	8,451,395	7.72
13	Si41Ge41H72	185,639	185,639	15,011,265	8.36
14	Si87H76	240,369	240,369	10,661,631	7.68
15	Ga41As41H72	268,096	268,096	18,488,476	7.84

During filtered subspace iteration, deflation [1] may reduce the number of columns in the right-hand dense matrix $\hat{V}^{(i)}$. When this happens, we switch from TDP-$mnkn$ to a classical CSR-based SpMM with $\hat{V}^{(i)}$ stored in column-major format. The numerical results in Section IV-F show that this strategy also improves performance for eigenvalue problems.

IV. Numerical Experiments

A. Experimental Setup

Experiments are conducted on a cluster whose nodes contain two Intel Xeon Gold 5320 CPUs (2.20 GHz), providing 56 cores and 256 GB DDR4 memory, connected by 100 Gbps InfiniBand. We compile the code with the Intel oneAPI C/C++ compiler (v23.0.0), MPICH 4.0.2, and Intel MKL (v23.0.0), and run an MPI + OpenMP hybrid implementation.

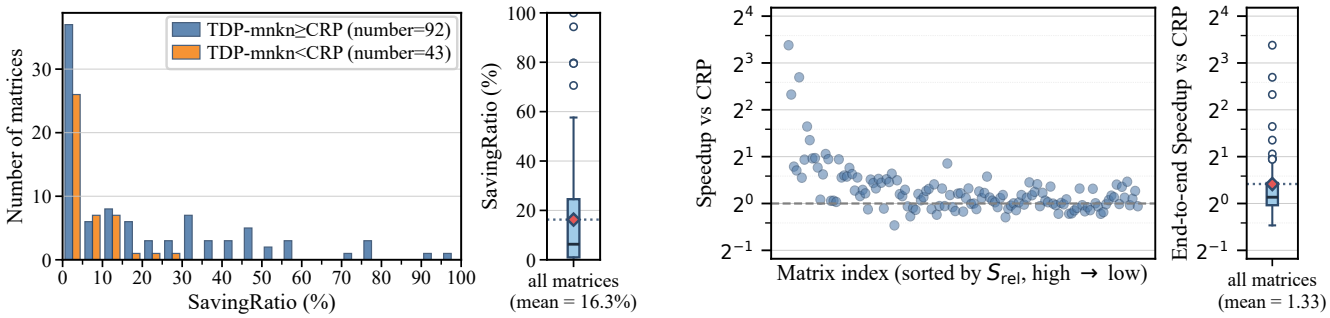
Our test set contains 135 sparse matrices from SuiteSparse [29], selected to cover a wide range of sparsity structures and communication scales. Table III lists ten representative matrices for the fixed-grid end-to-end studies; the scaling experiments in Section IV-E use a five-matrix subset, and Section IV-F additionally includes five matrices for polynomial-filtering eigensolver experiments.

As baselines, we use CRP [6] as the sparsity-aware mn-$r_A r_B$ reference, CombBLAS 2.0 [30] in both 2D C-stationary and 1.5D A-stationary SUMMA configurations, and SpComm3D [22] on a 3D grid where A may be replicated across layers. Against these baselines, we report the performance of TDP-mn, TDP-kn, and fused TDP-$mnkn$.

Unless otherwise stated, all experiments use an 8×8 process grid on 32 nodes, with 64 MPI ranks and 28 OpenMP threads per rank, $n = 1024$, and timings averaged over ten runs.

B. Model-Based Savings and Kernel Analysis

We first compare TDP-$mnkn$ against CRP on the full set of 135 matrices, excluding the one-time preprocessing cost of TDP-$mnkn$, namely the construction of $A^{(\mathrm{mn})}$ and



(a) Comparison of TDP and CRP on 135 SuiteSparse matrices; markers "≥" indicate TDP outperforms CRP, while "<" means the opposite.

(b) Speedup of TDP-$mnkn$ over CRP on 135 SuiteSparse matrices.

Fig. 6. Model-based savings and kernel performance on 135 large-scale SuiteSparse matrices.

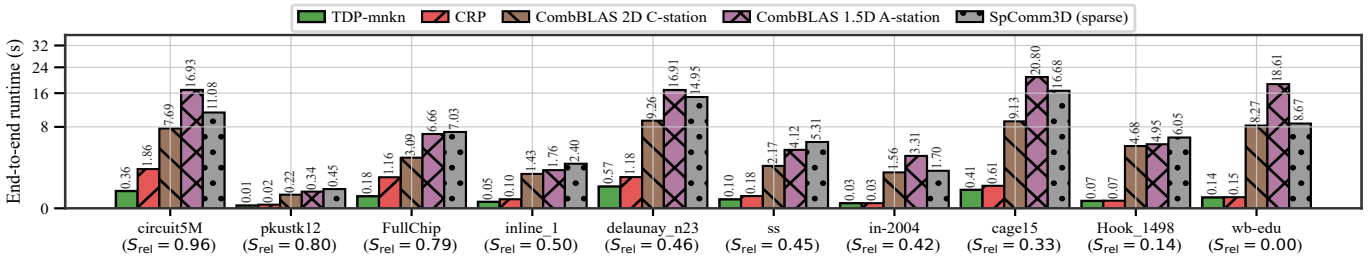


Fig. 7. Runtime of different SpMM kernels on a fixed 8×8 process grid with $n = 1024$.

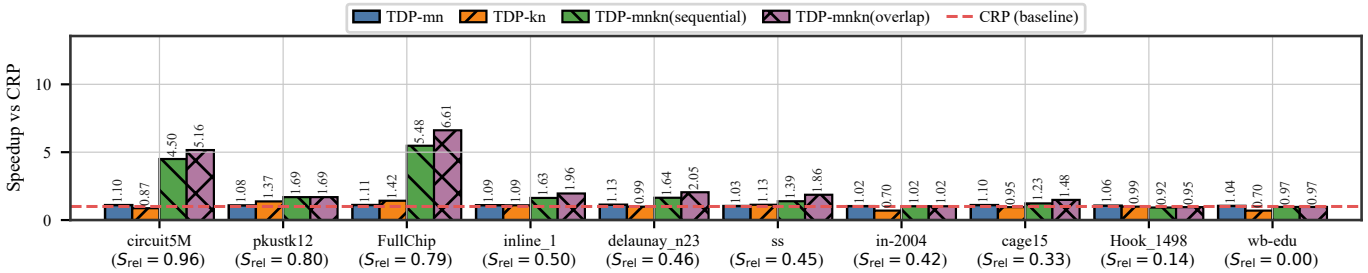


Fig. 8. Speedup over CRP on an 8×8 process grid with $n = 1024$ for different TDP kernels.

$A^{(\mathrm{kn})}$. Figure 6 summarizes both the modeled communication-saving ratio and the measured speedup over CRP on the 135-matrix dataset. Figure 6(a) shows the distribution of S_{rel}, with an average dense-side saving of 16.3% and some matrices exceeding 90%. Figure 6(b) shows that TDP-$mnkn$ achieves an average speedup of $1.33\times$, with a maximum of $10.4\times$, and accelerates 68% of the matrices.

C. End-to-End Comparison

We next compare TDP-$mnkn$ with CombBLAS (2D C-stationary and 1.5D A-stationary SUMMA, which supports only square grids) and with SpComm3D using layer depth 2. Figure 7 reports the end-to-end SpMM runtime on *ten* representative matrices. Following the timing split in Figure 5, we separate one-time sparse A exchange (init) from the fused execution window (dense B/C communication overlapped with local SpMM).

TDP-$mnkn$ and CRP are consistently faster than CombBLAS and SpComm3D on these cases. For `circuit5M`, the TDP model predicts that approximately 96% of the dense-side communication can be removed, and TDP-$mnkn$ is about $5\times$ faster than CRP. Across the ten representative matrices in Figure 7, TDP-$mnkn$ achieves an average end-to-end speedup of $2.4\times$ over CRP.

We further use CRP as the baseline in the ablation study of TDP-mn, TDP-kn, TDP-$mnkn$ without overlap, and TDP-$mnkn$ with overlap. As shown in Figure 8, TDP-mn consistently delivers speedups, mainly due to the communication-preparation optimization enabled by index compression in Phase 3 (Section III-C). TDP-kn achieves similar speedups because of the same index-compression preparation. The fused TDP-$mnkn$ kernel provides stable speedups, while the overlap optimization further improves performance on most cases.

For very short runs, local CPU overheads can dominate total runtime. For example, `inline_1` and `pkustk12` are extremely short runs, so communication savings translate into only modest end-to-end gains. For `pkustk12`, communication accounts for only 27% of the 0.0043 s SpMM runtime.

Figure 9 shows the same comparison on a 16×4 process grid. Changing the process grid may alter the adaptive N_{tile} selection and hence the resulting S_{rel}. For example, relative to

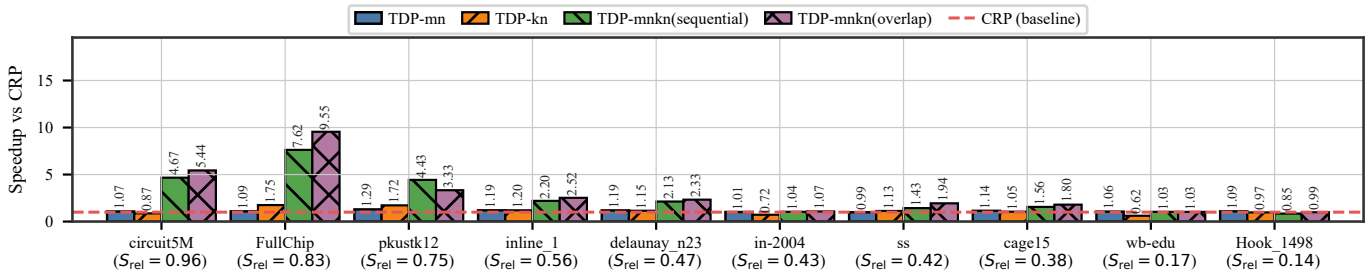


Fig. 9. Speedup over CRP on a 16×4 process grid with $n = 1024$ for different TDP kernels.

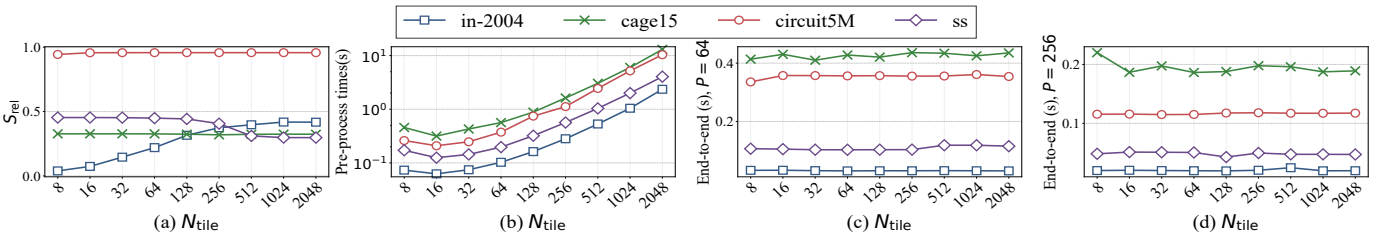


Fig. 10. Effects of tile-based partitioning: (a) communication-saving ratio S_{rel} under different N_{tile}; (b) preprocessing overhead; (c)–(d) impact of N_{tile} under different process counts with square process topology.

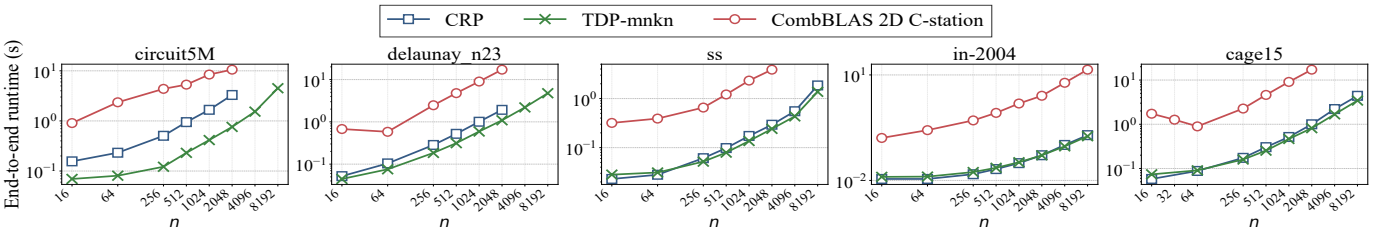


Fig. 11. Runtime versus the number of columns n in B and C on a fixed p_m=p_n=8 grid.

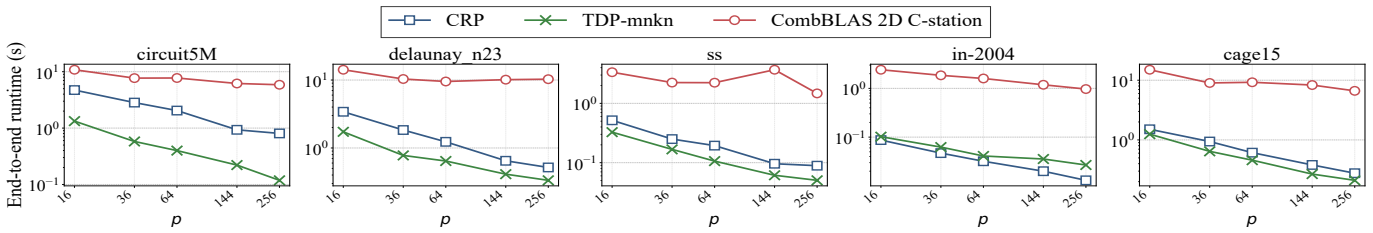


Fig. 12. Runtime versus the number of processes on square process grids with $n = 1024$.

the same matrix on the 8×8 grid, the communication volume for `wb-edu` decreases by about 15% after the grid adjustment. Compared with the 8×8 grid, the 16×4 grid increases the number of dense columns stored and communicated within each column process group, i.e., $1024/4 = 256$, and splitting the row process groups from 8 to 16 also increases dense-row communication. As a result, intra-column-group communication becomes more significant, and our algorithm attains larger speedups under the 16×4 configuration.

D. Tile Count N_{tile}

Figure 10(a) illustrates the trade-off associated with N_{tile}. For matrices such as `in-2004`, too small an N_{tile} fails to fully exploit dense-side communication avoidance, whereas overly large N_{tile} eventually reduces the predicted S_{rel}. At the same time, Figure 10(b) shows that excessively fine tiling sharply increases the preprocessing cost associated with label construction and redistribution of A.

TABLE IV
RUNTIME COMPARISON OF POLYNOMIAL FILTERING WITH TDP-$mnkn$ AND CLASSICAL CSR SPMM.

Matrix	N_{ev}	a	b	T_{TDP}	T_{CSR}	**Speedup**
Ge87H76	1039	−0.64	0.80	61.10	84.76	1.39×
Ge99H100	1125	−0.65	0.80	59.81	82.13	1.37×
Si41Ge41H72	738	−0.64	0.60	73.00	117.48	1.61×
Si87H76	677	−0.66	0.40	96.41	146.52	1.52×
Ga41As41H72	100	−0.64	−0.30	119.21	187.61	1.57×

The time to transform A into $A^{(mn)}$ and $A^{(kn)}$ is typically on the order of 0.1 s, with the dominant cost arising from sorting and partitioning during matrix slicing. Figure 10(c)–(d) further show that choices of N_{tile} do not change the SpMM runtime by orders of magnitude, and the lightweight preprocessing strategy provides a favorable trade-off.

E. Scaling

Figure 11 evaluates five representative matrices while scaling the number of columns n in matrices B and C, with the process grid and process count fixed. As n increases, both the amount of dense replicated data and the message sizes increase, so the communication-avoiding advantage of our method becomes more pronounced.

For `circuit5M` and `delaunay_n23`, TDP-$mnkn$ maintains a clear advantage across the entire range. For `ss` and `cage15`, TDP-$mnkn$ and CRP are competitive at some intermediate scales, but our method dominates at large column counts. The curves for `in-2004` remain close due to its small $nnz(A)$ and the share of MPI overhead. These results show that our method is particularly effective when communication becomes the dominant bottleneck in distributed SpMM.

In addition, TDP-$mnkn$ has lower memory requirements and can handle problems with up to 8192 columns, whereas CRP and CombBLAS encounter memory limits on the same platform. Figure 12 presents a strong-scaling study at $n = 1024$, where the number of MPI processes is scaled up while maintaining two ranks per node. As the process count grows, the runtime of our method decreases steadily. For `in-2004`, however, the small problem size limits the gains, and the resulting speedup is modest.

F. Benchmarking of Polynomial Filtering Eigensolvers

In this section, we apply TDP-$mnkn$ to the polynomial-filtering framework for interior eigenvalue problems; see Algorithm 3 in Section III-E. We evaluate five sparse symmetric matrices, namely `Ge87H76`, `Ge99H100`, `Si41Ge41H72`, `Si87H76`, and `Ga41As41H72`, from the SuiteSparse collection [29], as summarized in Table III.

These experiments are conducted on 32 nodes with 64 MPI ranks and 28 OpenMP threads per rank. Both TDP-$mnkn$ and the baseline use a 64×1 process grid. The baseline is a conventional 1D row-parallel CSR SpMM (m-r_B), implemented in the high-performance YHAMG package [35], [36].

Table IV reports the number of computed eigenvalues (N_{ev}), the target interval $[a, b]$, and the end-to-end runtimes obtained with TDP-$mnkn$ and CSR SpMM, denoted by T_{TDP} and T_{CSR}, respectively. The final column gives the speedup of TDP-$mnkn$ over the CSR baseline.

A complete polynomial-filtering solve typically requires hundreds to thousands of SpMM operations, so we do not include the one-time initialization cost of TDP-$mnkn$ in Table IV, since it usually accounts for less than 1.0% of the total runtime.

In the early stages of polynomial filtering, the dense block width is approximately $1.5\,N_{\text{ev}}$, and in this regime TDP-$mnkn$ achieves speedups of $1.37\times$–$1.61\times$ over the CSR baseline.

V. Related Work

We review recent distributed SpMM methods through the lens of the task–data parallelization (TDP) model.

Early GEMM-style distributed SpMM schemes were analyzed by Koanantakool et al. [28], who covered 1D, 2D, and 3D variants and provided algorithm-selection guidance based on replication factor and sparsity. Building on this line, MPI_FAUN [37] adopts a 2D TDP scheme (mk-$r_C r_B$), while CAGNET [38] develops a 2D bulk-synchronous MPI algorithm for tall-skinny matrices (mn-$c_B c_C$).

Selvitopi et al. [20] generalized such formulations into A-, B-, and C-stationary schemes and analyzed both synchronous and asynchronous communication costs. Mukhopadhyay et al. [21] further incorporated sparsity-aware communication and graph partitioning to reduce communication volume and load imbalance. In higher-dimensional settings, SpComm3D [22] is the first to implement sparsity-aware communication in 2.5D/3D SpMM and also extends the kernel to SDDMM. Block et al. [23] decomposed SpMM communication into sparsity-aware and sparsity-agnostic components and combined collective and one-sided communication in a 1D formulation.

Compared with these studies, which mainly analyze or optimize fixed task–data configurations, our work uses TDP as a unified abstraction that encodes the task- and data-distribution dimensions in one six-tuple descriptor, and further exploits the complementary communication behavior of mn-$r_A r_B$ and kn-$c_A r_C$ through a fused communication-avoiding design.

VI. Conclusion

This paper presents a unified task–data parallel (TDP) framework for distributed SpMM and a fused TDP-$mnkn$ algorithm that combines sparsity-aware tiling with tile-level adaptive path selection. The proposed design enables fine-grained communication avoidance without additional storage overhead, while also exposing opportunities for overlapping communication and computation. Through large-scale experiments on 135 SuiteSparse matrices, we demonstrate reductions in dense-side communication volume and corresponding runtime improvements, including up to an order-of-magnitude speedup over 2D SUMMA and an average end-to-end speedup of 2.4× over CRP on ten representative matrices. We further show the practical value of the proposed approach in polynomial-filtering eigensolvers, where reusable execution plans deliver $1.37\times$–$1.61\times$ speedups. Overall, the TDP framework provides a practical and scalable foundation for communication-efficient distributed SpMM and suggests a promising direction for integrating task decomposition, data layout, and sparsity-aware communication optimization in other sparse matrix kernels.

Acknowledgments

The authors would like to thank the referees for their valuable comments and Yuhui Ni for conducting the polynomial-filtered eigenvalue experiments. This work was supported in part by the National Key R&D Program of China under Grant 2023YFB3001702 and by the National Natural Science Foundation of China (NSFC) under Grants 62502048, 62472058, 12002382, 11275269, and 42104078.

REFERENCES

[1] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. D. Vorst, *Templates for the solution of linear systems: Building blocks for iterative methods*. Philadelphia, PA: SIAM, 1994.

[3] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-supervised learning with graph learning-convolutional networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 313–11 320.

[4] W. L. Hamilton, *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[5] S. Acer, O. Selvitopi, and C. Aykanat, "Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems," *Parallel Computing*, vol. 59, pp. 71–96, 2016.

[6] H. Huang and E. Chow, "Exploring the design space of distributed parallel sparse matrix-multiple vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 11, pp. 1977–1988, 2024. [Online]. Available: https://doi.org/10.1109/TPDS.2024.3452478

[7] T. Hoefler and J. L. Traff, "Sparse collective operations for MPI," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–8.

[8] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[9] R. Vuduc, J. W. Demmel, and K. A. Yelick, "OSKI: A library of automatically tuned sparse matrix kernels," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 521.

[10] A. Buluç and J. R. Gilbert, "The Combinatorial BLAS: Design, implementation, and applications," *The International Journal of High Performance Computing Applications*, vol. 25, no. 4, pp. 496–509, 2011.

[11] Y. Zhang, S. Li, F. Yuan, D. Dong, X. Yang, T. Li, and Z. Wang, "Memory-aware optimization for sequences of sparse matrix-vector multiplications," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 379–389. [Online]. Available: https://doi.org/10.1109/IPDPS54959.2023.00046

[12] A. Yzelman and R. H. Bisseling, "Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 3128–3154, 2009. [Online]. Available: https://doi.org/10.1137/080733243

[13] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999. [Online]. Available: https://doi.org/10.1109/71.780863

[14] G. Ballard, A. Druinsky, N. Knight, and O. Schwartz, "Hypergraph partitioning for parallel sparse matrix-matrix multiplication," in *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, 2015, pp. 86–88.

[15] M. Pang, X. Fei, P. Qu, Y. Zhang, and Z. Li, "A row decomposition-based approach for sparse matrix multiplication on GPUs," in *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 2024, pp. 377–389.

[16] S. E. Kurt, A. Sukumaran-Rajam, F. Rastello, and P. Sadayyapan, "Efficient tiled sparse matrix multiplication through matrix signatures," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14.

[17] C. Hong, A. Sukumaran-Rajam, I. Nisa, K. Singh, and P. Sadayappan, "Adaptive sparse tiling for sparse matrix multiplication," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019, pp. 300–314.

[18] L. E. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, College of Engineering, Montana State University, 1969.

[19] R. A. V. De Geijn and J. Watts, "SUMMA: Scalable universal matrix multiplication algorithm," *Concurr. Comput.: Pract. Exper.*, vol. 9, no. 4, pp. 255–274, 1997.

[20] O. Selvitopi, B. Brock, I. Nisa, A. Tripathy, K. Yelick, and A. Buluç, "Distributed-memory parallel algorithms for sparse times tall-skinny-dense matrix multiplication," in *Proceedings of the 35th ACM International Conference on Supercomputing*, 2021, pp. 431–442.

[21] U. Mukhopadhyay, A. Tripathy, O. Selvitopi, K. Yelick, and A. Buluç, "Sparsity-aware communication for distributed graph neural network training," in *Proceedings of the 53rd International Conference on Parallel Processing (ICPP '24)*. ACM, 2024, pp. 1–10.

[22] N. Abubaker and T. Hoefler, "SpComm3D: A framework for enabling sparse communication in 3D sparse kernels," 2024, arXiv:2404.19638. [Online]. Available: https://github.com/nfabubaker/SpComm3D

[23] C. Block, G. Gerogiannis, C. Mendis, A. Azad, and J. Torrellas, "Two-face: Combining collective and one-sided communication for efficient distributed SpMM," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 1200–1217.

[24] I. Ranawaka, M. T. Hussain, C. Block, G. Gerogiannis, J. Torrellas, and A. Azad, "Distributed-memory parallel algorithms for sparse matrix and sparse tall-and-skinny matrix multiplication," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–17.

[25] V. Bharadwaj, A. Buluç, and J. Demmel, "Distributed-memory sparse kernels for machine learning," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 47–58.

[26] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 3, pp. 866–901, 2011. [Online]. Available: https://doi.org/10.1137/090769156

[27] H. Jia-Wei and H.-T. Kung, "I/O complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 1981, pp. 326–333.

[28] P. Koanantakool, A. Azad, A. Buluç, D. Morozov, S.-Y. Oh, L. Oliker, and K. Yelick, "Communication-avoiding parallel sparse-dense matrix-matrix multiplication," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 842–853. [Online]. Available: https://doi.org/10.1109/IPDPS.2016.117

[29] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[30] A. Azad, O. Selvitopi, M. T. Hussain, J. R. Gilbert, and A. Buluç, "Combinatorial BLAS 2.0: Scaling combinatorial algorithms on distributed-memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 989–1001, 2022. [Online]. Available: https://doi.org/10.1109/TPDS.2021.3094091

[31] Y. Saad, *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

[32] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc Web page," https://www.mcs.anl.gov/petsc, 2019. [Online]. Available: https://www.mcs.anl.gov/petsc

[33] U. M. Yang *et al.*, "BoomerAMG: A parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002. [Online]. Available: https://doi.org/10.1016/S0168-9274(01)00115-5

[34] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 351–362, 2005. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1089014.1089019

[35] F. Yuan and S. Li, "YHAMG," 2022. [Online]. Available: https://gitee.com/e-level-parallel-algorithm/yhamg

[36] F. Yuan, X. Yang, Y. Huang, D. Dong, C. Xu, J. Liu, X. Yue, S. Li, and H. Wang, "CRAMG: A Communication-Reduced Algebraic Multigrid Method," in *Proceedings of the 39th ACM International Conference on Supercomputing*, New York, NY, USA, 2025, pp. 397–411. [Online]. Available: https://doi.org/10.1145/3721145.3725764

[37] R. Kannan, G. Ballard, and H. Park, "MPI-FAUN: An MPI-based framework for alternating-updating nonnegative matrix factorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 544–558, 2017.

[38] A. Tripathy, K. Yelick, and A. Buluç, "Reducing communication in graph neural network training," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00074