CoLa: Towards Communication-efficient Distributed Sparse Matrix-Matrix Multiplication on GPUs

Lixing Zhang
Beijing University of Posts and Telecommunications
Beijing, China
zhanglixing@bupt.edu.cn

Yingxia Shao*
Beijing University of Posts and Telecommunications
Beijing, China
shaoyx@bupt.edu.cn

Shigang Li
Beijing University of Posts and Telecommunications
Beijing, China
lishigang@bupt.edu.cn

Abstract

Sparse Matrix-Matrix Multiplication (SpMM) is a critical operator in many applications, such as graph neural networks (GNNs). However, when SpMM is scaled to multiple GPUs, existing works face significant challenges: (1) massive redundant communication and (2) unawareness of heterogeneous links. To address the issue of massive redundant communication, we introduce a communication redundancy-free distributed SpMM algorithm that efficiently reutilizes fetched remote data to reduce communication volume. To tackle the unawareness of heterogeneous links, we propose two link-aware optimization techniques: communication fusion, which leverages local GPUs as embedding caches to reduce communication over slow links; and request-coalesced communication, which coalesces necessary requested remote data into bulk transfers to maximize bandwidth utilization and minimize communication volume over proxy-based links. Based on these techniques, we develop CoLa, a highly communication-efficient distributed SpMM framework. Extensive evaluations on real-world datasets under different multi-GPU settings demonstrate that CoLa achieves geomean speedups of **8.56**×, **9.12**×, and **57.97**× over CAGNET, MGGCN, and MGG, respectively.

CCS Concepts

• **Computing methodologies** → **Distributed computing methodologies**.

*Corresponding author

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3730425

Keywords

Sparse Matrix, SpMM, Distributed Algorithms

ACM Reference Format:
Lixing Zhang, Yingxia Shao, and Shigang Li. 2025. CoLa: Towards Communication-efficient Distributed Sparse Matrix-Matrix Multiplication on GPUs. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721145.3730425

1 Introduction

Sparse Matrix-Matrix Multiplication (SpMM) is a key operator in many scientific and graph learning applications. For example, as a growing family of graph learning techniques, graph neural networks (GNNs) [25, 35, 40] use the spatial structure of graph data to learn embeddings for tasks including node classification [18], graph classification [48], and link prediction [47]. In GNNs, each target node aggregates the embeddings of neighboring nodes to update its own embedding. This process can be represented as SpMM, where the sparse matrix is the adjacency matrix of a graph, and the dense input and output matrices are original embeddings and updated embeddings, respectively. In this paper, we name one row of an input or output dense matrix as an embedding for simplicity.

Recently, the increasing memory demands for SpMM on real-world graphs exceed the capacity of a single GPU [7, 33, 38]. One prevalent approach is scaling SpMM on multiple GPUs. However, developing an efficient distributed SpMM on multiple GPUs faces the following two challenges:

Massive Redundant Communication. Existing distributed SpMM solutions [7, 9, 26, 31, 33, 38] have high communication cost due to the massive redundant GPU-to-GPU communication. They either transfer unnecessary remote embeddings or transfer the same remote embeddings repeatedly. SUMMA-like [34] algorithms, including CAGNET [33] and MGGCN [7] among others [9, 26, 31] employ coarse-grained collective communication strategies, broadcasting all local embeddings from one GPU to others at each stage. While the SUMMA paradigm is originally designed for dense-dense matrix multiplication, it is sparsity-unaware and results in unnecessary communication of remote embeddings that are not

required for subsequent computation in distributed SpMM. By contrast, based on multi-GPU shared memory [2–4, 44], MGG [38] introduces a sparsity-aware fine-grained communication pipeline, where each communication request fetches one remote embedding at a time. It adopts a row-wise SpMM pattern, where each worker processes a sparse row independently and only fetches the required remote embeddings according to the column indices of nonzeros within the sparse row. Despite avoiding the communication of unnecessary remote embeddings, MGG's row-wise SpMM pattern still results in redundant communication. Specifically, when multiple workers process different sparse rows containing nonzeros with the same column indices, the same remote input embeddings are fetched repeatedly.

Unawareness of Heterogeneous Links. Multi-node multi-GPU platforms typically have heterogeneous links across GPUs [28]. These diverse links bring in two critical issues: (1) *Frequent communication over slow links.* There are diverse links with different communication bandwidths, including NVLink (300 GB/s), PCIe (20 GB/s), QPI (12 GB/s), and IB (18 GB/s). Given a typical multi-node multi-GPU setting in modern HPC clusters, there exists a significant bandwidth gap that the intra-node link of NVLink (300 GB/s) offers an order of magnitude higher bandwidth than the inter-node link of IB (18 GB/s). However, existing distributed SpMM methods, including CAGNET, MGG, MGGCN, and others [7, 9, 31, 33, 38] do not consider the bandwidth gap, resulting in frequent embedding communication over slower links and performance degradation. (2) *Inefficient communication over proxy-based links.* Communication over heterogeneous links relies on different transport mechanisms. While NVLink, PCIe, and QPI enable on-board transport across GPUs, inter-node links like IB depend on proxy-based transport (e.g., IBRC, UCX) across GPUs [5]. Proxy-based transport introduces additional overhead, which significantly degrades communication throughput when handling frequent small messages. Without considering the different transport mechanisms of heterogeneous links, MGG [38] sticks to its fine-grained communication pipeline over proxy-based links, suffering from severe performance degradation. On the other hand, CAGNET and others [9, 26, 31] adopt coarse-grained collective communication over all types of links, missing the opportunity to leverage fine-grained communication over on-board transport for reducing redundant transfers.

To eliminate redundant communication, we first conduct an in-depth analysis of communication redundancy associated with different communication strategies for distributed SpMM. Based on the analysis, we propose a communication redundancy-free distributed SpMM algorithm, where each necessary remote embedding is fetched exactly once, and the communication volume is significantly reduced. This new algorithm eliminates redundant communication by efficiently reutilizing the shared remote embeddings among nonzeros from the same sparse columns.

To address the two issues caused by the unawareness of heterogeneous links, we further introduce two link-aware optimization techniques: (1) communication fusion to reduce *frequent communication over slow links*. For communication fusion, we group GPUs into different workgroups based on multi-GPU communication topology, where inter-workgroup links are slower than intra-workgroup links. We leverage GPUs within a workgroup as embedding buffers to cache remote embeddings from other workgroups, avoiding repeated inter-workgroup communication for the same remote embeddings across different workgroups, thereby effectively reducing expensive inter-workgroup communication over slow links. (2) request-coalesced communication to optimize *inefficient communication over proxy-based links*. We coalesce the requested remote embeddings into a large message for communication over proxy-based links. This approach simultaneously achieves minimal communication volume by transferring only necessary embeddings while maintaining high bandwidth utilization by avoiding the overhead of frequent small messages over proxy-based links.

Finally, we craft a highly communication-efficient distributed SpMM framework, namely CoLa [1] (**Co**mmunication redundancy-free and **L**ink-**a**ware). Additionally, CoLa applies pipeline optimization and community-aware graph reordering to further boost the overall performance. With extensive evaluations on a multi-GPU cluster, CoLa consistently outperforms the state-of-the-art baselines across various real-world datasets. Under different multi-GPU settings, CoLa achieves geomean speedups of **8.56**× over CAGNET [33], **9.12**× over MGGCN [7], and **57.97**× over MGG [38]. We summarize our contributions as follows:

- We give an in-depth analysis of the redundant communication for remote embeddings under different communication strategies in distributed SpMM (Section 3.1).
- We introduce a communication redundancy-free algorithm to eliminate redundant communication for remote embeddings in distributed SpMM (Section 3.2).
- We present two link-aware optimization techniques for the challenges of heterogeneous links: communication fusion to reduce *frequent communication over slow links* (Section 4.2) and request-coalesced communication to optimize *inefficient communication over proxy-based links* (Section 4.3).
- We craft CoLa, a high-performance distributed SpMM framework, with careful design of pipeline optimization and community-aware graph reordering to further boost the performance. Extensive evaluations demonstrate that CoLa achieves significant speedups over state-of-the-art baselines across various real-world datasets.

[1] https://github.com/zzzlxhhh/CoLa

Table 1: Notations

Notations	Description
$A_{n\times n}$	input sparse matrix
$B_{n\times k}$	input dense matrix
$C_{n\times k}$	output dense matrix
$\mathcal{P}$	number of total GPUs
$A_{i,j}$	ith row and jth column sparse chunk
P_i	the ith GPU
A_R^i	remote sparse chunk in P_i
A_L^i	local sparse chunk in P_i
$\mathcal{W}$	number of workgroups
$\mathcal{G}$	number of GPUs in a workgroup
$A^{i,j}$	the sparse data for P_i's stage-j SpMM
$A_R^{i,j}$	the remote sparse chunink for P_i's stage-j SpMM
$A_L^{i,j}$	the local sparse chunk for P_i's stage-j SpMM

2 Preliminaries

In this section, we provide preliminaries about SpMM computation, basic distributed SpMM procedure, and communication on multi-GPU platforms. The frequently used notations in this paper are listed in Table 1.

2.1 SpMM Computation

In this paper, SpMM is represented as $A_{n\times n}B_{n\times k} = C_{n\times k}$, where elements in these matrices are 4-byte. There are two SpMM computing methods on GPUs: row-wise and col-wise (column-wise) SpMM.

In row-wise SpMM, one worker (e.g., a thread warp) is allocated with a sparse row of A as illustrated by Figure 1 (a). First, the worker sequentially loads and aggregates the embeddings from B based on the column indices of nonzeros in the sparse row, buffering the aggregated embedding in the worker's scratchpad memory (e.g., local registers). Second, once the aggregation is complete, a single write operation accumulates the aggregation results to C.

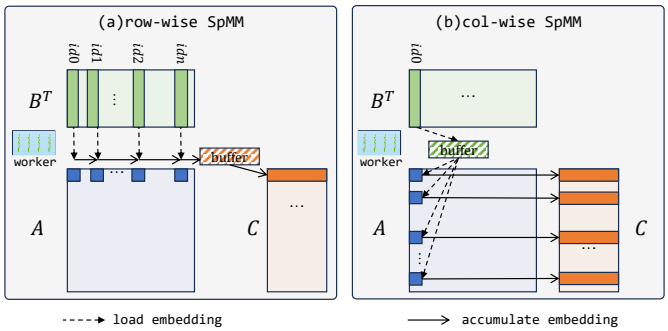

Figure 1: Examples of row-wise and col-wise SpMM

In col-wise SpMM, a worker is allocated a sparse column of A as demonstrated by Figure 1 (b). First, the embedding requested from B is buffered in scratchpad memory based on the column index of the allocated sparse column. Second, the worker loads the buffered embedding to compute partial results and sequentially accumulates the partial results to C for each nonzero. An essential feature of col-wise SpMM is reusing the embedding among nonzeros from the same sparse column.

2.2 Basic Distributed SpMM Procedure

Supposing we have $\mathcal{P}$ GPUs, where P_i is the i-th GPU with i ranging from 0 to $\mathcal{P}-1$. For a basic distributed SpMM, A is partitioned into $\mathcal{P}\times\mathcal{P}$ equal-sized sparse matrix chunks. B and C are partitioned into $\mathcal{P}$ equal-sized row panels. The partitioning is defined as:

$$A = \begin{pmatrix} A_{0,0} & \cdots & A_{0,\mathcal{P}-1} \\ \vdots & \ddots & \vdots \\ A_{\mathcal{P}-1,0} & \cdots & A_{\mathcal{P}-1,\mathcal{P}-1} \end{pmatrix} \quad (1)$$

$$B = \begin{pmatrix} B_0 \\ \vdots \\ B_{\mathcal{P}-1} \end{pmatrix},\ C = \begin{pmatrix} C_0 \\ \vdots \\ C_{\mathcal{P}-1} \end{pmatrix} \quad (2)$$

where $A_{i,:}$, B_i, and C_i are placed on P_i.

Subsequently, $A_{i,:}$ on GPU P_i is arranged into a local sparse chunk (**LSC**) A_L^i and a remote sparse chunk (**RSC**) A_R^i. The **LSC** and **RSC** of $A_{i,:}$ on P_i are formally defined as:

$$A_{i,:} = (A_L^i, A_R^i),\ A_L^i = A_{i,i},\ A_R^i = \sum_{j=0,j\neq i}^{\mathcal{P}-1} concat(A_{i,j}) \quad (3)$$

For **LSC** SpMM on A_L^i, GPU P_i can directly access all required embeddings from its local memory B_i. For **RSC** SpMM on A_R^i, GPU P_i needs to fetch remote embeddings as they are out of the range of local embeddings B_i.

The computation of C_i on P_i is formulated as:

$$C_i = \underbrace{A_L^i B}_{\textbf{LSC SpMM}} + \underbrace{A_R^i B}_{\textbf{RSC SpMM}} = A_{i,i}B_i + \sum_{j=0,j\neq i}^{\mathcal{P}-1} A_{i,j}B_j \quad (4)$$

where the computation consists of **LSC** SpMM and **RSC** SpMM.

2.3 Communication on Multi-GPU Platforms

For distributed SpMM on multi-GPU platforms, input and output embeddings B and C are partitioned across GPUs. Both fine-grained communication and coarse-grained communication can be utilized to exchange embeddings between GPUs.

For fine-grained communication in distributed SpMM, the granularity of a transferred message is a single remote embedding. This approach leverages the distributed shared memory paradigm [2–4, 44], where GPUs share a unified memory space. It enables direct GPU-to-GPU data access within CUDA kernels, eliminating CPU-side overhead. Each worker (e.g., a thread warp) can independently fetch remote embeddings on demand without CPU intervention, allowing precise control of each embedding movement. Therefore, fine-grained communication for distributed SpMM (e.g., MGG [38]) is sparsity-aware, where direct remote embedding access can be utilized to exchange only necessary embeddings across GPUs based on the sparsity structure of $\boldsymbol{A}$.

For coarse-grained communication, the granularity of a transferred message is multiple remote embeddings. Following the message passing paradigm, it enables GPU-to-GPU data transfers in bulk but requires CPU-side orchestration and synchronization. Compared to fine-grained communication, it introduces additional synchronization overhead and lacks the flexibility of precise access for each remote embedding within CUDA kernels. Some existing sparsity-unaware distributed SpMM methods [7, 9, 31, 33] apply coarse-grained collective communication (e.g., broadcast, allreduce), without considering the sparsity structure of $\boldsymbol{A}$.

3 Communication Redundancy-free Distributed SpMM

In this section, we first provide an in-depth analysis of the communication redundancy of different communication strategies in distributed SpMM. Then, we detail the algorithm of our communication redundancy-free distributed SpMM.

3.1 Analysis of Communication Redundancy

Following the basic distributed SpMM procedure illustrated in Section 2.2, there are different communication methods to load remote embeddings.

3.1.1 Coarse-grained Collective Communication. Existing works like MGGCN [7] and CAGNET [33] utilize coarse-grained collective communication for distributed SpMM. The distributed SpMM is divided into $\mathcal{P}$ stages, where each GPU takes turns broadcasting its local embeddings of dense matrix $\boldsymbol{B}$. For instance, when it is P_i's turn, it needs to broadcast $\boldsymbol{B_i}$. However, not all the broadcasted embeddings are necessary for subsequent computing due to the sparsity of $\boldsymbol{A}$. Therefore, distributed SpMM based on coarse-grained collective communication is sparse-unaware, causing redundant communication for the embeddings of $\boldsymbol{B}$. The communication volume in the context of coarse-grained collective communication is represented as:

$$V_{CG} = \frac{4n}{\mathcal{P}} k(\mathcal{P}-1)\mathcal{P} = 4nk(\mathcal{P}-1) \tag{5}$$

where k is the column width of $\boldsymbol{B}$, and each GPU broadcasts $\frac{n}{\mathcal{P}}k$ elements of its whole local embeddings to other $(\mathcal{P}-1)$ GPUs in each stage, for a total of $\mathcal{P}$ stages.

3.1.2 Fine-grained communication. As discussed in Section 2.3, fine-grained communication in multi-GPU distributed shared memory systems enables co-scheduling of computation and communication within CUDA kernels, where remote embeddings are accessed on-demand during computation. This tight coupling between computation and communication means that the **RSC** SpMM computation pattern directly shapes the communication behavior. Consequently, fine-grained communication for distributed SpMM is naturally aware of the sparsity structure of $\boldsymbol{A}$. In this section, with fine-grained communication, we analyze its communication volume for **RSC** SpMM under row-wise and col-wise computation pattern, respectively.

Row-wise RSC SpMM. When performing row-wise SpMM on **RSC** (e.g., MGG [38]), each worker processes a single sparse row and fetches remote embeddings based on the column indices of nonzeros within that row. However, this approach leads to redundant communication since some remote embeddings are repeatedly accessed. More specifically, given an **RSC**, when multiple nonzeros in different rows share the same column index, they require the same remote embedding. However, since each worker operates independently, these shared embeddings must be fetched repeatedly. The resulting communication volume is:

$$V_{FR} = 4E_R k \tag{6}$$

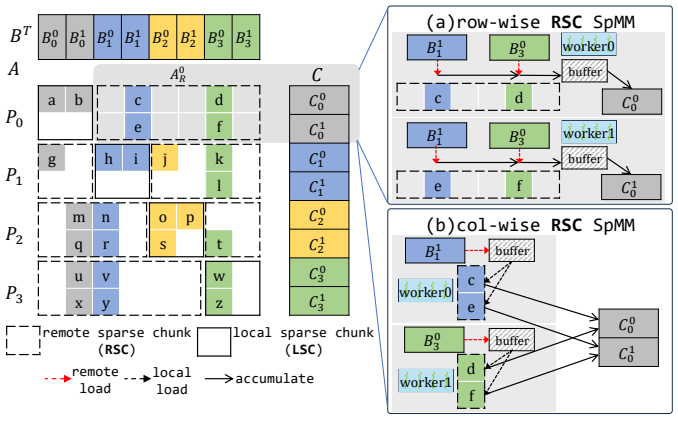

Figure 2: Row-wise and col-wise RSC SpMM on A_R^0 over multi-GPU distributed shared memory.

where E_R is the total number of nonzeros from all the **RSC** of $\boldsymbol{A}$. An example of row-wise **RSC** SpMM on $\boldsymbol{A}_R^0$ is illustrated in Figure 2 (a). Despite {c,e} both requiring the remote embedding $\boldsymbol{B}_1^1$, two different workers have to access $\boldsymbol{B}_1^1$ separately. It is a similar case for {d,f} and $\boldsymbol{B}_3^0$. This inability to reuse the embeddings leads to four remote embedding loads for **RSC** SpMM on $\boldsymbol{A}_R^0$.

Col-wise RSC SpMM. We observe that nonzeros in one sparse column require the same remote embedding. This characteristic presents a significant optimization opportunity: fetching each remote embedding once and enabling its reuse across nonzeros in the corresponding column. This optimization eliminates redundant communication, which is overlooked by existing approaches [7, 9, 26, 31, 33, 38].

The above opportunity aligns perfectly with the feature of the col-wise SpMM, where each worker reuses the buffered embedding across a sparse column of nonzeros. Consequently, col-wise **RSC** SpMM can fetch only necessary remote embeddings for the nonempty columns of **RSC**. The communication volume is:

$$V_{FC} = 4N_R k \tag{7}$$

where N_R is the total number of nonempty columns from all the **RSC** of $\boldsymbol{A}$. Figure 2 (b) demonstrates the col-wise **RSC** SpMM on $\boldsymbol{A}_R^0$. $\boldsymbol{B}_1^1$ and $\boldsymbol{B}_3^0$ are reused among the nonzeros in the same columns by two workers, resulting in two remote embedding loads, fewer than the row-wise **RSC** SpMM.

3.1.3 Comparison of Communication Volume and Communication Redundancy. Supposing that the **RSC** has no empty columns, we have $V_{FC} = V_{CG}$ since $N_R = n(\mathcal{P} - 1)$. However, due to the sparsity of real-world matrices, many empty columns may emerge in **RSC** after partitioning for distributed SpMM as illustrated in Figure 2, where V_{FC} is less than V_{CG}. V_{CG} is thus the upper bound of V_{FC}. Assuming another extreme scenario that each nonempty column in **RSC** contains only one nonzero, with $E_R = N_R$, there is $V_{FR} = V_{FC}$. V_{FC} is thus the lower bound of V_{FR}. Consequently, col-wise **RSC** SpMM theoretically has the least communication volume:

$$V_{FC} = O(V_{CG}),\ V_{FR} = \Omega(V_{FC}). \tag{8}$$

For distributed SpMM, the column indices of nonempty columns in **RSC** determine which remote embeddings are necessary for SpMM on **RSC**. Therefore, the necessary communication volume for these necessary embeddings is thus defined as:

$$V_R = 4N_R k \tag{9}$$

where V_R is the communication volume for the necessary remote embeddings, and N_R is the number of nonempty columns in **RSC**. Obviously, we have $V_R = V_{FC}$, where col-wise **RSC** SpMM is thus communication redundancy-free.

We conduct empirical comparisons of the communication volumes of these three communication patterns on 16 GPUs (2 nodes, 8 GPUs per node) with a column width of $k = 32$. The results are presented in Table 2, where the details of these real-world sparse matrices are listed in Table 5. The ratio of redundant communication of each communication pattern is also listed in parentheses. It is clear that V_{FC} consistently has the minimal communication volumes, highlighting the effectiveness of col-wise RSC SpMM in removing redundant communication.

Table 2: Communication volume (MB) and the ratio of redundant communication of different communication patterns when SpMM is run on 16 GPUs (2 nodes, 8 GPUs per node).

Matrix	V_{CG}	V_{FR}	V_{FC}
CLJ	7320.49 (89%)	2481.61 (66%)	**835.425(0%)**
COK	5625.81 (69%)	12964.64 (87%)	**1723.121(0%)**
CYT	2078.05 (96%)	271.85 (66%)	**92.75(0%)**
CIT	5361.26 (95%)	858.12 (67%)	**285.828(0%)**
PRD	4484.31 (89%)	2714.10 (82%)	**497.932(0%)**
SLJ	8876.17 (91%)	2681.62 (69%)	**822.492(0%)**
SPK	2989.75 (81%)	1491.62 (61%)	**575.964(0%)**
SSO	4764.36 (83%)	2910.63 (72%)	**818.91(0%)**

3.2 Algorithm of Communication Redundancy-free Distributed SpMM

Based on the analysis in Section 3.1, we adopt col-wise SpMM on **RSC** to avoid redundant communication for remote embeddings. From the perspective of a worker, Algorithm 1 details the communication redundancy-free distributed SpMM on P_i, where $h = n/\mathcal{P}$ is the height of $\boldsymbol{B_i}$ and $\boldsymbol{C_i}$.

For **LSC** SpMM, the worker is allocated a sparse row of nonzeros in **LSC** (Line 3), whose row index also identifies the target output embedding position $Crow$ in $\boldsymbol{C_i}$ (Line 4). For each nonzero element in the row, it retrieves the corresponding local embedding from $\boldsymbol{B_i}$ based on the local column index $Acol$ (Lines 6-7). To minimize global memory access overhead, the algorithm maintains an intermediate buffer $Cemb$ in the worker's scratchpad memory to accumulate partial results (Line 9). Once all nonzeros in the sparse row are processed, the final aggregated result is written to $\boldsymbol{C_i}[Crow][:]$ (Line 11).

For **RSC** SpMM, the worker is allocated a sparse column of nonzeros in **RSC** (Line 13-14). It then identifies the position of the remote input embedding by (t, pos) (Line 15-16), where pos indicates the embedding's position in target GPU P_t's $\boldsymbol{B}^t$. Notably, the remote embedding is fetched once and buffered in the worker's scratchpad memory without consuming global memory (Line 18), enabling its reuse across all

Algorithm 1 Communication Redundancy-free Distributed SpMM

```
1:  function DistSpMM(LSC, RSC, B_i)
2:      /*LSC SpMM*/
3:      LNZ ← nonzeros in a row of LSC
4:      Crow ← the row index of LNZ
5:      for x ∈ LNZ do
6:          Acol ← x.colIdx%h
7:          Bemb ← B_i[Acol][:]
8:          /*Buffer the aggregated embedding in Cemb*/
9:          Cemb ← Cemb + x.val * Bemb
10:     end for
11:     C_i[Crow][:] ← C_i[Crow][:] + Cemb
12:     /*RSC SpMM*/
13:     RNZ ← nonzeros in a column of RSC
14:     Acol ← the common column index of RNZ
15:     t ← Acol/h
16:     pos ← Acol%h
17:     /*Buffer the remote embedding in Bemb*/
18:     Bemb ← B_t[pos][:]
19:     for x ∈ RNZ do
20:         Crow ← x.rowIdx
21:         C_i[Crow][:] ← C_i[Crow][:] + x.val * Bemb
22:     end for
23: end function
```

nonzeros in the sparse column. For each nonzero, the algorithm computes and directly accumulates the partial result to the corresponding position in $\boldsymbol{C_i}$ (Lines 19-22).

4 Link-aware Optimization Techniques

In this section, we first present the optimization opportunities for two issues caused by heterogeneous links, including **Issue #1**: *frequent communication over slow links*, and **Issue #2**: *inefficient communication over proxy-based links*. Subsequently, for **Issue #1**, we introduce communication fusion to fuse the communication over slow links, thereby improving communication efficiency. Lastly, for **Issue #2**, we propose request-coalesced communication to further improve the communication efficiency over proxy-based links.

4.1 Optimization Opportunities for Heterogeneous Links

In this section, we analyze the performance issues caused by heterogeneous interconnects and identify key optimization opportunities. The experiments in this section to demonstrate the performance issues are conducted on a typical multi-node multi-GPU setting commonly found in modern HPC, which has inter-node links of IB and intra-node links of NVLink.

Table 3: Communication volumes (MB) over IB on a 2-node setting with 8 GPUs per node. VI_{CG}: coarse-grained communication; VI_{FR}: row-wise SpMM; VI_{FC}: column-wise SpMM; VI_{CoLa}: CoLa with link-aware optimizations.

Matrix	VI_{CG}	VI_{FR}	VI_{FC}	VI_{CoLa}
CLJ	309.65	1078.84	421.88	164.75
COK	3904.26	5415.27	891.75	306.01
CYT	1108.29	124.03	46.86	25.77
CIT	2859.34	381.14	132.09	86.50
PRD	165.37	808.43	195.89	99.67
SLJ	2391.63	1580.39	436.14	194.61
SPK	129.43	618.48	281.52	109.98
SSO	894.95	1184.79	381.87	140.34

Issue #1: *frequent communication over slow links*. The bandwidth gaps among heterogeneous links significantly impact distributed SpMM performance, where frequent communication over slow links degrades its performance. To quantify this issue, we conduct experiments on a 2-node setting (8 GPUs per node) with NVLink for intra-node and IB for inter-node communication. We measure the communication volumes over IB (VI) under different communication strategies introduced in Section 3.1. They all demonstrate massive IB communication volumes across all matrices. The table also presents our optimized VI_{CoLa}. VI_{CoLa} achieves the minimal IB communication volume, demonstrating the effectiveness of our proposed optimization.

Opportunity #1. The above **Issue #1** motivates us to propose a communication optimization strategy considering the communication topology. GPUs can be organized into different workgroups based on the communication topology, where intra-workgroup links are faster than inter-workgroup links. This organization can be applied to common multi-GPU settings with heterogeneous links. For multi-node platforms with NVLink and IB, a node with multiple NVLink-connected GPUs is a workgroup. For a single-node NUMA platform with PCIe and QPI, each NUMA domain with multiple PCIe-connected GPUs is a workgroup.

Consider the basic distributed SpMM in Section 2.2, where **RSC** SpMM on $\boldsymbol{A_R^i}$ is $\sum_{j=0,j\neq i}^{\mathcal{P}-1} \boldsymbol{A_{i,j}B_j}$ (Equation 4). When GPUs P_i and P_k are in the same workgroup connected via fast intra-workgroup link, while P_j is in a different workgroup connected to P_i and P_k through slow inter-workgroup link, $\boldsymbol{A_{i,j}}$ and $\boldsymbol{A_{k,j}}$ assigned to P_i and P_k both require embeddings of B_j from P_j. This presents an opportunity for communication fusion: instead of having P_i and P_k communicate repeatedly with P_j over the slow link, we can buffer B_j to P_i through an inter-workgroup transfer, then allowing P_k to access $\boldsymbol{B_j}$ from P_i via the fast intra-workgroup link. This

Table 4: Latency (ms) for transferring various data sizes (MB) using fine-grained (FG) and coarse-grained (CG) communication over NVLink and IB.

Data size	0.25	0.5	1	2	4	8
FG-NVLink	17.13	18.10	19.93	25.55	29.01	41.46
CG-NVLink	19.00	23.48	26.50	25.69	32.30	44.31
FG-IB	3607.78	7477.82	15414.77	31248.01	63377.97	128245.72
CG-IB	64.47	79.67	104.81	160.20	274.22	495.09

insight leads to our communication fusion strategy detailed in Section 4.2.

Issue #2: *inefficient communication over proxy-based links.* To demonstrate this performance issue, we conduct profiling experiments [2] by comparing the data transfer latencies of fine-grained communication and coarse-grained communication over NVLink and proxy-based IB. The results are presented in Table 4. While fine-grained communication excels on NVLink, its performance severely degrades on proxy-based IB due to proxy queue congestion caused by frequent small messages.

Opportunity #2. **Issue #2** naturally suggests to use coarse-grained communication over proxy-based links for higher bandwidth utilization. However, existing approaches using naive coarse-grained collective communication [7, 9, 26, 31, 33] are sparsity-unaware, leading to redundant embedding transfers as analyzed in Section 3.1.1. This dilemma motivates our key idea: we can achieve both high bandwidth utilization and minimal communication volume by coalescing the necessary remote embeddings into a bulk message before communication. This approach not only maximizes bandwidth utilization by avoiding proxy queue congestion but also eliminates redundant communication by making the coarse-grained communication sparsity-aware. Building on this insight, we introduce request-coalesced communication in Section 4.3 to improve communication efficiency over proxy-based links.

4.2 Communication Fusion over Slow Links

In this section, we introduce our communication fusion strategy to reduce communication over slow links. As claimed in **Opportunity #1**, GPUs can be grouped into different workgroups based on the communication topology, where intra-workgroup links are faster than inter-workgroup links. Correspondingly, the distributed SpMM presented in Algorithm 1 can be divided into multiple stages according to the number of workgroups, each consisting of *inter-workgroup embedding distribution* involving communication over slow inter-workgroup links and *intra-workgroup SpMM* involving communication over fast intra-workgroup links. Communication fusion is applied to *inter-workgroup embedding distribution* to reduce communication over slow inter-workgroup links.

4.2.1 Multi-stage Distributed SpMM. Given a multi-GPU platform with $\mathcal{P}$ GPUs, they are divided into $\mathcal{W}$ workgroups, where each workgroup contains $\mathcal{G}$ GPUs (i.e., $\mathcal{P} = \mathcal{W}\mathcal{G}$). On GPU P_i, the computation of $\boldsymbol{C_i}$ is divided into $\mathcal{W}$ stages:

$$C_i = \sum_{j=0}^{\mathcal{W}-1} C^{i,j} = \sum_{j=0}^{\mathcal{W}-1} \sum_{k=0}^{\mathcal{G}-1} A_{i,\mathcal{G}j+k} B_{\mathcal{G}j+k} \tag{10}$$

where $\boldsymbol{C^{i,j}}$ represents the partial results computed during stage-j, requiring remote embeddings from workgroup-j.

4.2.2 Inter-workgroup Embedding Distribution. Communication fusion is achieved by leveraging local GPUs within a workgroup to buffer remote embeddings from other workgroups, avoiding different GPUs within a workgroup from repeatedly communicating for the same embeddings from another workgroup, thereby reducing slow inter-workgroup communication. We name this buffering process as *inter-workgroup embedding distribution.*

More specifically, before entering stage-j SpMM in P_i, stage-j *inter-workgroup embedding distribution* distributes remote embeddings from GPUs in workgroup-j to GPUs in P_i's workgroup. The distribution follows a local-rank-matching principle: each GPU buffers embeddings from its counterpart GPU with the same local rank in a remote workgroup. Consequently, all GPUs within P_i's workgroup can access remote embeddings required for stage-j SpMM computation through fast intra-workgroup links.

4.2.3 Intra-workgroup SpMM. Following stage-j *inter-workgroup embedding distribution*, the stage-j SpMM on P_i can then be formulated as *intra-workgroup SpMM*:

$$A^{i,j} = (A_L^{i,j}, A_R^{i,j}),\ A_L^{i,j} = A_{i,\mathcal{G}j+i},\ A_R^{i,j} = \sum_{k=0}^{\mathcal{G}-1} concat(A_{i,\mathcal{G}j+k}) \tag{11}$$

$$C^{i,j} = \underbrace{A_L^{i,j}B}_{\text{LSC SpMM}} + \underbrace{A_R^{i,j}B}_{\text{RSC SpMM}} = A_{i,\mathcal{G}j+i}B_{\mathcal{G}j+i} + \sum_{k=0,k\neq i}^{\mathcal{G}-1} A_{i,\mathcal{G}j+k}B_{\mathcal{G}j+k} \tag{12}$$

where $\boldsymbol{A^{i,j}}$ is the sparse data processed by P_i in stage-j. $\boldsymbol{A_L^{i,j}}$ and $\boldsymbol{A_R^{i,j}}$ are the local and remote sparse chunk processed by P_i in stage-j, respectively. Notably, the communication redundancy-free Algorithm 1 can be directly applied to P_i's stage-j *intra-workgroup SpMM*, whose SpMM is divided into **LSC** SpMM and **RSC** SpMM, and the communication redundancy-free property is maintained.

[2] In this profiling, fine-grained communication is implemented with GPU-side *nvshmemx_float_get_warp* where each warp transfers a 32-float embedding. Coarse-grained communication is implemented with CPU-side *nvshmemx_float_get_on* to transfer all data at once.

Algorithm 2 Request-coalesced Communication for *inter-workgroup embedding distribution*

```
1:  function CoalComm(i, t, recvBuf)          ▷ i is the source GPU's rank, t and recvBuf are the target GPU's rank and buffer address
2:      Cols ← ∅, sendBuf ← ∅
3:      /*Indices of necessary embeddings*/
4:      for j = 0 to G − 1 do
5:          k = j + ⌊t/G⌋ * G
6:          neCols ← indices of nonempty columns in A_{k,t}
7:          Cols ← Cols ∪ neCols
8:      end for
9:      /*Coalesce embeddings based on col*/
10:     for col ∈ Cols do
11:         Bcol ← col%h
12:         sendBuf.append(B_i[Bcol])
13:     end for
14:     Put(sendBuf, recvBuf, t)
15: end function
```

4.3 Request-coalesced Communication over Proxy-based Links

In this section, we introduce the request-coalesced communication for the efficient embedding communication over proxy-based links based on **Opportunity #2**.

For *inter-workgroup embedding distribution* (Section 4.2.2), its communication needs to be optimized when the inter-workgroup links are proxy-based (e.g., IB). With request-coalesced communication, requested embeddings for *inter-workgroup embedding distribution* are coalesced into a bulk message before communication. It improves the communication throughput over proxy-based links through coarse-grained communication, simultaneously maintaining sparsity-awareness by coalescing the necessary remote embeddings.

Algorithm 2 details the request-coalesced communication. For a given source-target GPU pair (P_i, P_t), the algorithm begins by initializing empty sets *Cols* and *sendBuf* (Line 2). *Cols* indicate the necessary embeddings in B_i required by P_t's workgroup for *inter-workgroup embedding distribution*, while *sendBuf* is the buffer to store the coalesced embeddings in P_i's local memory that are later sent to P_t. It then collects indices of necessary embeddings to *Cols* by examining the nonempty column indices of the sparse matrix chunk $\boldsymbol{A_{k,t}}$, where k ranges over the ranks of GPUs within P_t's workgroup (Lines 4-8). Subsequently, for each collected index *col* in *Cols*, it is first transformed to local embedding index in $\boldsymbol{B_i}$ using *col%h*, and the corresponding embedding in $\boldsymbol{B_i}$ is then appended to *sendBuf* (Lines 10-13). Finally, the coalesced embeddings are transferred to the target GPU P_t's receiving buffer *recvBuf* through a single bulk communication operation (Line 14). This algorithm significantly improves communication throughput for *inter-workgroup embedding distribution* by coalescing the requested embeddings into a single bulk transfer, while maintaining minimal communication volume by transferring only the necessary embeddings. Notably, we use PUT initiated by P_i rather than GET initiated by P_t as the GPU-to-GPU communication primitive, as the coalescing of requested embeddings from $\boldsymbol{B_i}$ can only be achieved within the memory of the source GPU P_i.

5 Implementation of CoLa

We present CoLa, a high-performance distributed SpMM framework that integrates our three key optimization techniques: communication redundancy-free algorithm, communication fusion, and request-coalesced communication. In this section, we further introduce pipeline optimization and community-aware graph reordering to improve the performance of CoLa.

5.1 Pipeline Optimization

As introduced in Section 4.2, communication fusion divides the complete distributed SpMM into $\mathcal{W}$ stages, each consisting of *intra-workgroup SpMM* and *inter-workgroup embedding distribution*. This multi-stage structure presents an opportunity for pipeline optimization, where the execution of *intra-workgroup SpMM* and *inter-workgroup embedding distribution* can be overlapped.

To avoid confusion, we distinguish between two terms: "stage" refers to the logical decomposition of multi-stage distributed SpMM described in Section 4.2 and Equations 12, where the GPU P_i's stage-j SpMM on $A^{i,j}$ requires embeddings from workgroup-j; and "phase" refers to the pipeline execution order according to our scheduling.

The pipeline employs a double-buffer mechanism, with each GPU P_i maintaining two buffers ($recvBuf[i][0]$ and $recvBuf[i][1]$): one for the current-phase *intra-workgroup SpMM* and another for buffering remote embeddings for the next-phase *inter-workgroup embedding distribution*. The pipeline begins with phase-0 *intra-workgroup SpMM* using locally available embeddings, bypassing phase-0 *inter-workgroup embedding distribution* while asynchronously performing phase-1 *inter-workgroup embedding distribution*. Afterwards, the phase-j *intra-workgroup SpMM* is thus overlapped with the phase-$j+1$ *inter-workgroup embedding distribution*.

Algorithm 3 details the pipeline scheduling. For GPU P_i, it first sets $recvBuf[i][0]$ to local $\boldsymbol{B_i}$ and $recvBuf[i][1]$ to empty (Line 3). The main pipeline consists of $\mathcal{W} - 1$ phases (Lines 4-12), each starting with a global synchronization (Line 5) to ensure the availability of embeddings for phase-j *intra-workgroup SpMM*. $g = (\lfloor i/\mathcal{G} \rfloor + j) \bmod \mathcal{W}$ (Line 7) is to determine the sparse data $A_L^{i,g}$ and $A_R^{i,g}$ for

Algorithm 3 Link-aware Distributed SpMM Pipeline

```
1:  function LinkAwarePipeline(i)   ▷ i is GPU P_i's rank
2:      /*Initialize P_i's recvBuf with local B_i*/
3:      recvBuf[i][0] ← B_i, recvBuf[i][1] ← ∅
4:      for j = 0 to W − 2 do
5:          GlobalSync()
6:      /*Phase-j Intra-workgroup SpMM*/
7:          g = (⌊i/G⌋ + j) mod W
8:          DistSpMM(A_L^{i,g}, A_R^{i,g}, recvBuf[i][j%2])
9:      /*Phase-j+1 Inter-workgroup embedding distribution*/
10:         t = (i − G ∗ (j + 1)) mod P
11:         Async.CoalComm(i, t, recvBuf[t][(j + 1)%2])
12:     end for
13:     GlobalSync()
14:     g = (⌊i/G⌋ + W − 1) mod W
15:     DistSpMM(A_L^{i,g}, A_R^{i,g}, recvBuf[i][(W − 1)%2])
16: end function
```

P_i's phase-j *intra-workgroup SpMM*. It then performs phase-j *intra-workgroup SpMM* using Algorithm 1 (Line 8). Following this, according to the local-rank-matching principle mentioned in Section 4.2, it calculates the target GPU rank $t = (i - \mathcal{G} * (j+1)) \bmod \mathcal{P}$ (Line 10) for phase-$j+1$ *inter-workgroup embedding distribution*. Subsequently, it initiates asynchronous request-coalesced communication (Line 11) for phase-$j+1$ *inter-workgroup embedding distribution*. After finishing the main pipeline, the pipeline concludes with a final *intra-workgroup SpMM* phase (Lines 13-15).

5.2 Incorporation with Community-aware Graph Reordering

Graph reordering techniques [13, 17, 20, 23, 24, 27] have been widely used to accelerate graph applications. Since communication in CoLa is aware of the sparsity pattern of a sparse matrix, to further reduce communication volume, we integrate community-aware graph reordering as a preprocessing step to optimize the sparsity pattern of the input sparse matrix (i.e., the adjacency matrix of a graph).

Community-aware graph reordering techniques, such as Rabbit Reordering [6], can efficiently transform a graph's adjacency matrix into an approximate block-diagonal pattern in highly parallel, requiring minimal time. Consequently, the number of nonzeros in **RSC** is reduced, resulting in less remote embedding access for nonempty columns. Simultaneously, nonzeros in **LSC** enjoy better data locality, thereby improving the performance of **LSC** SpMM. Synergizing with CoLa, the community-aware graph reordering is a default preprocessing step.

Table 5: Datasets for evaluation. n is the number of rows, nnz is the number of nonzeros. Density is $\frac{nnz}{n^2}$.

Matrix Name	n	nnz	Density
com-LiveJournal**(CLJ)**	4.00M	69.36M	$4.34e^{-6}$
com-Orkut**(COK)**	3.07M	234.37M	$2.48e^{-5}$
com-Youtube**(CYT)**	1.13M	5.98M	$4.64e^{-6}$
ogbl-citation2**(CIT)**	2.93M	30.39M	$3.54e^{-6}$
ogbn-products**(PRD)**	2.45M	123.72M	$2.06e^{-5}$
soc-LiveJournal**(SLJ)**	4.85M	68.99M	$2.94e^{-6}$
soc-Pokec**(SPK)**	1.63M	30.62M	$1.15e^{-5}$
sx-stackoverflow**(SSO)**	2.60M	36.23M	$5.35e^{-6}$

5.3 Implementation Details

There are several multi-GPU distributed shared memory programming supports, such as the Unified Memory (UM) [4], and NVSHMEM [3], among which NVSHMEM is based on the Partitioned Global Address Space (PGAS) model [44]. NVSHMEM offers both GPU-side fine-grained communication and CPU-side coarse-grained communication primitives. For convenience, all the communication operations in CoLa are implemented based on NVSHMEM. Its GPU-side fine-grained communication primitives are used to implement communication redundancy-free *intra-workgroup SpMM*. Its CPU-side coarse-grained communication primitives are used to implement request-coalesced communication for *inter-workgroup embedding distribution*.

6 Experiment Evaluations

In this section, we evaluate the performance of CoLa, validate the effectiveness of the proposed techniques, and demonstrate CoLa's performance in accelerating GNNs.

6.1 Experimental Setup

6.1.1 Datasets. We collect eight large-scale sparse matrices from SuiteSparse [16] and OGB [21], whose corresponding graphs have million-scale nodes, as listed in Table 5.

6.1.2 Experimental Platforms. We conduct experiments on two platforms with different communication topologies:

- **A800 platform:** A multi-node multi-GPU platform, where each node has 8 NVLink-connected Nvidia A800 GPUs, with different nodes connected with IB. Each A800 GPU has 1381 GB/s global memory bandwidth and a compute capability of 8.6. This platform serves as our main experimental platform.
- **V100 platform:** A single-node multi-GPU platform consists of 2 NUMA domains connected with QPI, with each domain housing 4 Nvidia V100 GPUs linked with PCIe. Each V100 GPU has 725 GB/s global memory bandwidth and a compute capability of 7.0.

Table 6: The comparison of CoLa to baselines on single-node multi-GPU settings.

A800-#nGPU	2				4				8			
Methods	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa
CLJ	27.44 (10.16×)	8.43 (3.12×)	19.30 (7.15×)	2.70	22.21 (12.10×)	5.29 (2.88×)	13.16 (7.17×)	1.84	17.69 (14.20×)	4.91 (3.94×)	9.00 (7.23×)	1.25
COK	52.81 (8.25×)	19.18 (3.00×)	46.43 (7.26×)	6.40	38.72 (8.41×)	10.50 (2.28×)	30.77 (6.68×)	4.60	21.75 (6.69×)	6.42 (1.97×)	21.74 (6.69×)	3.25
CYT	4.62 (9.69×)	1.58 (3.32×)	7.55 (15.83×)	0.48	3.43 (11.83×)	1.67 (5.74×)	8.14 (28.06×)	0.29	2.75 (13.47×)	2.78 (13.60×)	8.77 (42.97×)	0.20
CIT	13.41 (9.73×)	4.68 (3.40×)	9.02 (6.55×)	1.38	8.30 (9.46×)	3.41 (3.88×)	5.46 (6.22×)	0.88	6.80 (11.26×)	3.89 (6.45×)	3.08 (5.10×)	0.60
PRD	28.79 (9.73×)	10.76 (3.63×)	20.35 (6.87×)	2.96	18.14 (10.12×)	6.14 (3.42×)	12.90 (7.19×)	1.79	13.63 (11.49×)	4.37 (3.68×)	7.32 (6.17×)	1.19
SLJ	30.01 (10.70×)	8.98 (3.20×)	20.67 (7.37×)	2.81	24.17 (12.64×)	5.68 (2.97×)	11.69 (6.11×)	1.91	19.57 (14.42×)	5.64 (4.16×)	8.72 (6.42×)	1.36
SPK	11.97 (10.01×)	3.64 (3.05×)	9.10 (7.61×)	1.20	9.47 (10.05×)	2.58 (2.74×)	5.66 (6.01×)	0.94	6.92 (9.25×)	2.86 (3.82×)	3.70 (4.95×)	0.75
SSO	15.08 (9.23×)	4.68 (2.86×)	17.57 (10.75×)	1.63	11.89 (9.52×)	3.23 (2.59×)	14.04 (11.24×)	1.25	9.24 (9.33×)	3.58 (3.61×)	13.01 (13.12×)	0.99
Geo. spd	**9.66×**	**3.19×**	**8.29×**	**1.00×**	**10.42×**	**3.18×**	**8.40×**	**1.00×**	**10.95×**	**4.41×**	**8.50×**	**1.00×**

Table 7: The comparison of CoLa to baselines on multi-node multi-GPU settings.

A800-#nGPU	16				24				32			
Methods	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa
CLJ	23.06 (7.50×)	85.52 (27.84×)	715.59 (232.94×)	3.07	25.31 (7.18×)	96.38 (27.33×)	636.89 (180.63×)	3.53	29.64 (8.05×)	100.62 (27.34×)	940.76 (255.64×)	3.68
COK	25.77 (4.37×)	66.13 (11.22×)	12095.38 (2051.80×)	5.90	25.17 (3.45×)	75.23 (10.31×)	5357.47 (734.20×)	7.30	27.23 (3.46×)	78.81 (10.01×)	6317.69 (802.45×)	7.87
CYT	6.92 (11.19×)	23.57 (38.14×)	420.17 (679.89×)	0.62	9.15 (12.91×)	27.65 (38.99×)	96.13 (135.58×)	0.71	12.57 (14.68×)	30.20 (35.28×)	231.85 (270.85×)	0.86
CIT	17.16 (10.38×)	62.81 (38.02×)	803.36 (486.29×)	1.65	19.96 (10.67×)	70.47 (37.68×)	568.43 (303.97×)	1.87	20.66 (9.90×)	72.84 (34.89×)	636.57 (304.87×)	2.09
PRD	15.50 (7.93×)	52.32 (26.76×)	1042.96 (533.48×)	1.96	17.26 (5.74×)	60.37 (20.08×)	1798.72 (598.38×)	3.01	20.17 (7.36×)	62.40 (22.78×)	766.68 (279.91×)	2.74
SLJ	28.17 (7.45×)	103.09 (27.25×)	7588.86 (2006.04×)	3.78	32.72 (8.68×)	116.13 (30.80×)	734.60 (194.80×)	3.77	34.42 (6.88×)	122.25 (24.42×)	4252.88 (849.39×)	5.01
SPK	9.92 (4.80×)	34.09 (16.48×)	752.78 (364.01×)	2.07	12.14 (5.27×)	39.24 (17.02×)	637.61 (276.62×)	2.31	15.22 (6.19×)	42.07 (17.11×)	452.21 (183.90×)	2.46
SSO	14.43 (5.56×)	56.13 (21.62×)	1476.79 (568.87×)	2.60	17.30 (5.88×)	61.58 (20.93×)	746.64 (253.78×)	2.94	21.17 (6.76×)	66.78 (21.33×)	768.29 (245.46×)	3.13
Geo. spd	**7.04×**	**24.20×**	**657.72×**	**1.00×**	**6.92×**	**23.44×**	**285.97×**	**1.00×**	**7.36×**	**22.61×**	**340.76×**	**1.00×**

6.1.3 Baselines. We evaluate CoLa against state-of-the-art open-source distributed SpMM from two categories:

- **SUMMA-like methods:** CAGNET [33] and MGGCN [7] are based on coarse-grained collective communication as discussed in Section 3.1.1. CAGNET implements a bunch of SUMMA-like distributed SpMM algorithms, including 1D, 1.5D, and 2D, among which we choose the one with the best performance for comparison. MGGCN, originally designed for single-node multi-GPU settings, employs a 1D SUMMA algorithm with pipelined communication-computation overlap. We have tried our best to extend MGGCN to multi-node multi-GPU settings based on the implementation details in their paper, in order to provide a comprehensive comparison with CoLa.
- **Fine-grained communication based:** MGG [38] is based on fine-grained communication adopting row-wise **RSC** SpMM pattern as discussed in Section 3.1.2. Since MGG is based on fine-grained communication and is sparsity-aware, community-aware graph reordering is also applied as a default preprocessing step in its comparison with CoLa. As for the effectiveness of community-aware graph reordering on different distributed SpMM methods, a detailed analysis is provided in Section 6.5.

CAGNET, MGGCN, and MGG accelerate GNN over whole graphs through optimized distributed SpMM kernels. We separately extract their SpMM kernels for comparison as CAG-SpMM, MGGCN-SpMM, and MGG-SpMM.

6.2 Overall Performance

To comprehensively evaluate the performance of CoLa across different communication topologies, we conduct experiments on the A800 platform with both single-node multi-GPU and multi-node multi-GPU settings. Table 6 and Table 7 report the distributed SpMM execution time of CoLa and baselines on single-node multi-GPU settings and the multi-node multi-GPU settings, respectively. The performance is reported as a tuple of execution time (ms) and the speedup achieved by CoLa. The geomean speedups of CoLa over baselines across datasets are also listed.

6.2.1 Performance on Single-node Multi-GPU Settings. CoLa shows remarkable speedups over baselines, consistently showing great performance across all datasets. To further summarize, with the number of GPUs from {2,4,8}, CoLa achieves geomean speedups of **10.33**×, **3.55**×, and **8.39**× over CAG-SpMM, MGG-SpMM, and MGGCN-SpMM, respectively.

With all GPUs connected by NVLink on a single node, the performance gap of baselines over CoLa stems from the redundant communication. The deficiency of MGGCN-SpMM and CAG-SpMM primarily comes from the communication for unnecessary embeddings using coarse-grained collective communication. MGGCN-SpMM partially mitigates this through pipeline optimization, therefore performing better compared to other baselines. Despite MGG-SpMM's use of fine-grained communication to fetch only necessary remote embeddings, some remote embeddings are fetched repeatedly since its application of row-wise **RSC** SpMM. In contrast, CoLa consistently outperforms baselines by eliminating redundant communication using Algorithm 1.

6.2.2 Performance on Multi-node Multi-GPU Settings. CoLa maintains its dominant performance on multi-node multi-GPU settings. Overall, on configurations with 2-node (16 GPUs), 3-node (24 GPUs), and 4-node (32 GPUs), CoLa achieves

geomean speedups of **7.10**×, **23.41**×, and **400.19**× over CAG-SpMM, MGG-SpMM, and MGGCN-SpMM, respectively.

On multi-node settings with heterogeneous links (faster intra-node NVLink vs. slower inter-node IB), CoLa's link-aware optimization techniques provide significant advantages. For the multi-node multi-GPU settings here, CoLa treats each node's 8 GPUs as a workgroup, achieving exceptional performance through its optimization of communication redundancy-free *Intra-workgroup SpMM* and link-aware *inter-workgroup embedding distribution*. Despite being bothered by redundant communication, CAG-SpMM has the best performance among baselines, since its 1.5D algorithm can reduce the communication over inter-node IB. MGGCN-SpMM, without link-aware optimization, suffers from massive inefficient communication over inter-node IB. MGG-SpMM performs the worst mainly due to its notorious application of fine-grained communication over proxy-based IB, where frequent small messages over IB are extremely inefficient, and yet without optimizing the communication volume.

6.3 The Effectiveness of Proposed Techniques

6.3.1 Ablation Studies. We validate the effectiveness of the proposed techniques through ablation studies. With distributed SpMM scaled on 4 nodes with 32 A800 GPUs in total, Table 8 compares the performance of the `basic` method (i.e., applying row-wise SpMM on RSC using fine-grained communication under static 1D partitioning like MGG [38]), `RF` (i.e., communication redundancy-free distributed SpMM in Section 3.2 without link-aware optimization), `RF+CF` (i.e., `RF` with communication fusion introduced in Section 4.2), `RF+CF+RC` (i.e., `RF+CF` with request-coalesced communication introduced in Section 4.3), and `RF+CF+RC+PP` (i.e., `RF+CF+RC` with pipeline optimization).

Table 8 shows that each proposed technique effectively contributes to the performance improvement. `RF` largely boosts the performance compared to `basic`, showing the effectiveness of eliminating redundant communication for remote embeddings. However, `RF` is still slowed down by its application of fine-grained communication over proxy-based IB. `RF+CF` reduces the communication volume over slow IB by communication fusion. `RF+CF+RC` further improves communication efficiency compared to `RF+CF` by applying request-coalesced communication over proxy-based IB. Lastly, `RF+CF+RC+PP` further boosts the performance by applying pipeline optimization.

Table 8: The comparison of `basic`, `RF`, `RF+CF`, `RF+CF+RC`, and `RF+CF+RC+PP` on 4-node with 32 A800 GPUs.

Methods	basic	RF	RF+CF	RF+CF+RC	RF+CF+RC+PP
CLJ	940.76	484.45	19.67	5.48	3.68
COK	6317.69	721.51	16.32	11.07	7.87
CYT	231.85	40.24	5.66	1.20	0.86
CIT	636.57	154.09	14.15	2.88	2.09
PRD	766.68	280.43	12.28	3.91	2.74
SLJ	4252.88	476.86	23.84	6.32	5.01
SPK	452.21	236.66	8.19	3.79	2.46
SSO	768.29	361.71	12.87	4.82	3.13

Table 9: Performance of different methods on V100 platform with 2 QPI-connected NUMA domains, each containing 4 PCIe-connected V100 GPUs.

Methods	CAG-SpMM	MGGCN-SpMM	MGG-SpMM	CoLa_w/o_LA	CoLa
CLJ	68.01(4.85×)	109.32(7.80×)	96.58(6.89×)	27.86 (1.99×)	14.02
COK	61.73(1.86×)	48.34(1.46×)	446.25(13.47×)	59.60 (1.80×)	33.14
CYT	24.31(10.97×)	20.58(9.29×)	21.20(9.57×)	3.44 (1.55×)	2.22
CIT	48.60(7.42×)	44.87(6.85×)	37.25(5.69×)	10.31 (1.57×)	6.55
PRD	44.43(4.42×)	38.06(3.79×)	87.63(8.73×)	10.31 (1.03×)	10.04
SLJ	82.07(4.58×)	73.00(4.08×)	138.12(7.71×)	30.07 (1.68×)	17.91
SPK	27.53(2.87×)	25.65(2.68×)	51.53(5.38×)	18.66 (1.95×)	9.58
SSO	47.61(3.16×)	39.31(2.61×)	109.32(7.25×)	23.72 (1.57×)	15.08
Geo. spd	**4.48×**	**4.09×**	**7.77×**	**1.61×**	**1.00×**

6.3.2 The Effectiveness of Link-aware Optimization on Different Platforms. The effectiveness of link-aware optimization on multi-node multi-GPU settings (i.e., A800 platform) is validated in Section 6.3.1, where link-aware optimization is used to improve communication efficiency over inter-node IB. We further evaluate the effectiveness of link-aware optimization on V100 platforms, where 2 NUMA domains are connected with QPI, with each domain having 4 PCIe-connected V100 GPUs. In this setting, CoLa's link-aware optimization treats 4 GPUs within a single NUMA domain as a workgroup. We use CoLa_w/o_LA to represent CoLa without link-aware optimization. Table 9 presents the performance of CoLa, CoLa_w/o_LA, and baselines. CoLa demonstrates 1.61× geomean speedup over CoLa_w/o_LA by optimizing communication over inter-workgroup QPI. CoLa also maintains remarkable advantages over other baselines, validating its generality of optimization techniques across different GPU platforms.

6.4 Scalibility

6.4.1 Scalability with Increasing the Number of GPUs. Figure 3 presents the execution time of all methods with increasing the number of GPUs from {1,2,4,8} within a single node. The execution time of CoLa efficiently scales down as the number of GPUs is increased across all datasets. This impressive scalability validates CoLa's superior communication efficiency by eliminating redundant communication. It's worth mentioning that only CoLa can successfully scale down from 1 GPU to 2 GPUs across all datasets, while others all degraded due to redundant communication. CAG-SpMM and MGGCN-SpMM under coarse-grained collective communication show unpleasant scalability, where many fetched embeddings are

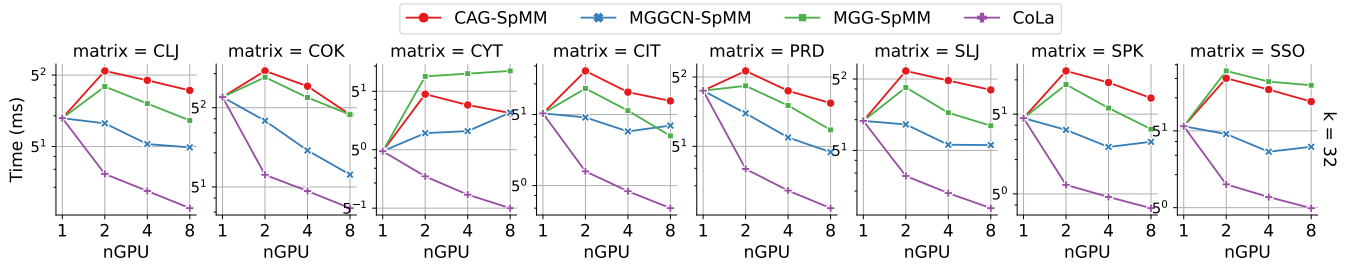

Figure 3: Performance comparison of different methods with increasing GPUs on a single node. When there is one GPU, we use Nvidia cuSPARSE [1] for all methods' single-GPU SpMM.

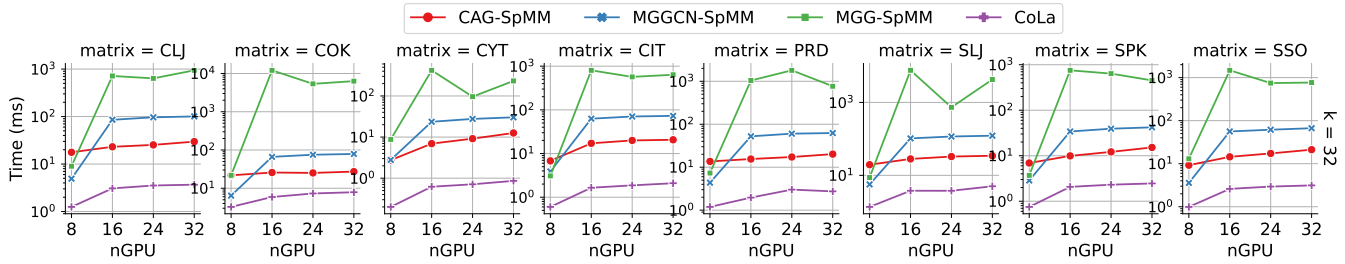

Figure 4: Performance comparison of different methods with increasing GPUs on multi-node multi-GPU settings.

unnecessary for computing. MGG-SpMM also shows poor scalability, due to its excessive redundant communication by loading remote embeddings repeatedly.

Figure 4 presents the execution time of all methods with increasing the number of GPUs from {8,16,24,32} on multi-node settings. As the inter-node IB (18 GB/s) is way slower than intra-node NVLink (300 GB/s), all methods fail to scale down the execution time. All baselines show poor scalability on multi-node settings, without link-aware optimization over IB. CoLa still maintains the best performance among all methods, being the best choice for scaling distributed SpMM to more GPUs.

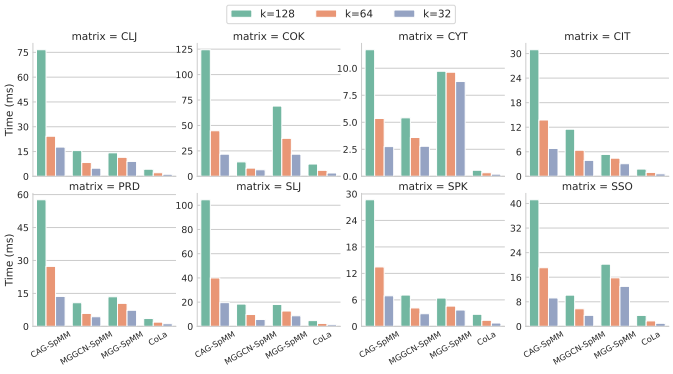

Figure 5: The comparison of different methods with decreasing k from {128,64,32} on 8 GPUs.

6.4.2 Scalability with Decreasing the Embedding Dimension k. Figure 5 presents the execution time of all methods with decreasing k from {128,64,32} on 8 GPUs. As k decreases, the execution time of all methods scales down as expected. Notably, the more steep decline of execution time of a method along with k, the more sensitive its performance is to k. Among all methods, CAG-SpMM is the most sensitive to k, which is not good for its performance with even bigger k.

6.5 The Effectiveness of Community-aware Graph Reordering

To validate the impact of graph reordering, we use '_r' to denote methods with graph reordering, and '_w/o_r' to denote methods without graph reordering. Table 10 demonstrates the geomean speedups of CoLa_r and CoLa_w/o_r over baselines when SpMM is scaled with 8 A800 GPUs. This table reveals that MGGCN and CAGNET do not benefit from graph reordering. MGGCN-SpMM_r even shows much worse performance since it relies on its node permutation preprocessing to balance the workload across GPUs. The performance of MGG is largely boosted by reordering, where CoLa_r shows **12.03**× speedups over MGG-SpMM_w/o_r and **8.50**× over MGG-SpMM_r. This is because sparsity-aware MGG-SpMM_r enjoys fewer nonzeros in **RSC** after reordering, thereby reducing communication according to Equation 6. However, MGG-SpMM_r is still worse compared to CoLa by repeatedly loading remote embeddings. Even without

Table 10: The speedups of CoLa w/o and w/ reordering over baselines on 8 A800 GPUs.

Geo. spd	CAG-SpMM_w/o_r	CAG-SpMM_r	MGGCN-SpMM_w/o_r	MGGCN-SpMM_r	MGG-SpMM_w/o_r	MGG-SpMM_r
CoLa_w/o_r	4.76×	4.11×	1.92×	2.68×	5.23×	3.70×
CoLa_r	10.95×	9.46×	4.41×	6.16×	12.03×	8.50×

graph reordering, CoLa_w/o_r still outperforms all the baselines, which again validates the effectiveness of our proposed techniques. Note that one-time reordering overhead is lightweight and amortizable in iterative applications (e.g., GNNs). More specifically, the reordering accounts for only an average of 2.66% of one epoch of GCN across datasets.

6.6 Application in GNNs

We name the GCN embedded with CoLa as CoLaGCN. We evaluate CoLaGCN and baselines on a 5-layer GCN with an input feature size of 16, an output feature size of 16, and a hidden layer size of 32. Table 11 details the performance comparison of CoLaGCN and baselines when scaled with 8 A800 GPUs. Geomean speedups are also listed in the table, where CoLaGCN significantly outperforms baselines, showing great applicability for GNN tasks.

Table 11: The GCN Performance on 8 A800 GPUs, reported as the average time (s) of one GCN epoch.

Methods	CAGNET	MGGCN	MGG	CoLaGCN
CLJ	7.04(6.30×)	2.31(2.07×)	4.69(4.19×)	1.12
COK	9.87(6.40×)	3.18(2.06×)	10.74(6.97×)	1.54
CYT	7.69(39.44×)	1.52(7.79×)	4.43(22.71×)	0.20
CIT	3.26(6.45×)	1.75(3.46×)	1.68(3.32×)	0.51
PRD	5.51(8.30×)	2.07(3.12×)	3.79(5.72×)	0.66
SLJ	7.99(8.57×)	2.50(2.68×)	4.58(4.91×)	0.93
SPK	3.21(6.18×)	1.66(3.20×)	1.93(3.70×)	0.52
SSO	5.82(8.57×)	1.74(2.56×)	6.61(9.74×)	0.68
Geo. spd	**8.88×**	**3.07×**	**6.22×**	**1.00×**

7 Related Work

In this section, we first review existing distributed SpMM optimization works. Then, we discuss local SpMM optimizations in non-distributed machines. Lastly, we discuss the optimizations on distributed GNNs.

Distributed SpMM. Several existing works [9, 26, 31, 33] devise a family of 1D, 1.5D, 2D, 3D distributed SpMM, and their variants to reduce communication volume using coarse-grained collective communication. MGGCN [7] and MGG [38] accelerate GNNs' embedding aggregation by optimizing distributed SpMM. MGGCN uses coarse-grained asynchronous collective communication to achieve computation-communication overlapping. In contrast, MGG proposes a fine-grained communication pipeline on top of distributed multi-GPU shared memory, causing massive redundant communication from row-wise SpMM on **RSC**. CPU-based Two-Face [10] employs a combination of coarse-grained collective communication and fine-grained asynchronous communication. All these works neglect to discuss communication redundancy and consider the bandwidth gaps among heterogeneous links, which cause inefficient communication.

Local SpMM Optimizations. Some [15, 17, 37, 41] accelerate SpMM by optimizing the workload imbalance among different workers. Some [20, 43, 46] employ hybrid blocking techniques to adaptively exploit the data locality of matrices. Some [12, 29, 39] leverage the powerful tensor core units [14] to accelerate SpMM in sparse DNNs [19] or GNNs. Others [15, 45, 46] employ machine learning to navigate the SpMM optimization strategies.

Distributed GNN. Both WholeGraph[42] and XGNN [32] implement mini-batch GNN training over multi-GPU distributed shared memory by optimizing mini-batch sampling. However, they do not accelerate the aggregation of embeddings with optimized distributed SpMM. DGL [36] relies on Pytorch-Direct [30] to prepare embeddings from the CPU to multiple GPUs, thus forming a data parallelism training. ROC [22] employs Nvidia Legion [8] to manage data transfers. DGCL [11] optimizes the communication planning problem on multi-GPU platforms based on a cost model.

8 Conclusion

In this work, we introduced CoLa, a communication-efficient distributed SpMM framework on multi-GPU platforms. Based on a comprehensive analysis of the different communication volumes under varied communication strategies, we introduced communication redundancy-free distributed SpMM. It eliminates redundant communication by reusing remote embeddings among nonzeros in the same sparse columns. To improve communication efficiency over heterogeneous links, we proposed link-aware optimization techniques, including request-coalesced communication and communication fusion. Request-coalesced communication is to improve communication efficiency over proxy-based links, while communication fusion is to reduce communication over slow links. Pipeline optimizations and community-aware graph reordering are also integrated into CoLa to further improve the performance. Extensive experiments on various matrices and GPU platforms have shown CoLa's impressive performance, scalability, and cross-platform generality over state-of-the-art baselines.

Acknowledgments

This work is supported by the National Science and Technology Major Project (2022ZD0116315), National Natural Science Foundation of China (Nos. 62272054, 62192784), Beijing Nova Program (No. 20230484319), and Xiaomi Young Talents Program.

References

[1] 2024. cuSPARSE. https://docs.nvidia.com/cuda/cusparse/index.html. Accessed: 2024-05-08.

[2] 2024. GPUDirect | NVIDIA Developer. https://developer.nvidia.com/gpudirect. Accessed: 2024-05-13.

[3] 2024. NVSHMEM | NVIDIA Developer. https://developer.nvidia.com/nvshmem. Accessed: 2024-05-08.

[4] 2024. Unified Memory for CUDA Beginners | NVIDIA Technical Blog. https://developer.nvidia.com/blog/unified-memory-cuda-beginners/. Accessed: 2024-05-13.

[5] 2025. Device APIs on Proxy-Based Transport. https://docs.nvidia.com/nvshmem/release-notes-install-guide/best-practice-guide/apis.html. Accessed: 2025-01-13.

[6] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. 2016. Rabbit Order: Just-in-Time Parallel Reordering for Fast Graph Analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 22–31. doi:10.1109/IPDPS.2016.110

[7] Muhammed Fatih Balin, Kaan Sancak, and Umit V. Catalyurek. 2023. MG-GCN: A Scalable multi-GPU GCN Training Framework. In *Proceedings of the 51st International Conference on Parallel Processing* (Bordeaux, France) *(ICPP '22)*. Association for Computing Machinery, New York, NY, USA, Article 79, 11 pages. doi:10.1145/3545008.3545082

[8] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2012. Legion: expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '12)*. IEEE Computer Society Press, Washington, DC, USA, Article 66, 11 pages.

[9] Vivek Bharadwaj, Aydın Buluç, and James Demmel. 2022. Distributed-Memory Sparse Kernels for Machine Learning. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 47–58. doi:10.1109/IPDPS53621.2022.00014

[10] Charles Block, Gerasimos Gerogiannis, Charith Mendis, Ariful Azad, and Josep Torrellas. 2024. 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* (<conf-loc>, <city>La Jolla</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1200–1217. doi:10.1145/3620665.3640427

[11] Zhenkun Cai, Xiao Yan, Yidi Wu, Kaihao Ma, James Cheng, and Fan Yu. 2021. DGCL: an efficient communication library for distributed GNN training. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 130–144. doi:10.1145/3447786.3456233

[12] Zhaodong Chen, Zheng Qu, Liu Liu, Yufei Ding, and Yuan Xie. 2021. Efficient tensor core-based GPU kernels for structured sparsity under reduced precision *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 78, 14 pages. https://doi.org/10.1145/3458817.3476182

[13] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. 2009. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Paris, France) *(KDD '09)*. Association for Computing Machinery, New York, NY, USA, 219–228. doi:10.1145/1557019.1557049

[14] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[15] Guohao Dai, Guyue Huang, Shang Yang, Zhongming Yu, Hengrui Zhang, Yufei Ding, Yuan Xie, Huazhong Yang, and Yu Wang. 2022. Heuristic adaptability to input dynamics for spmm on gpus. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 595–600.

[16] Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (dec 2011), 25 pages. doi:10.1145/2049662.2049663

[17] Ruibo Fan, Wei Wang, and Xiaowen Chu. 2023. Fast Sparse GPU Kernels for Accelerated Training of Graph Neural Networks. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 501–511. doi:10.1109/IPDPS54959.2023.00057

[18] Alberto García-Durán and Mathias Niepert. 2017. Learning graph representations with embedding propagation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5125–5136.

[19] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.* 22, 1, Article 241 (jan 2021), 124 pages.

[20] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. 2019. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming* (Washington, District of Columbia) *(PPoPP '19)*. Association for Computing Machinery, New York, NY, USA, 300–314. doi:10.1145/3293883.3295712

[21] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: datasets for machine learning on graphs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 1855, 16 pages.

[22] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems* 2 (2020), 187–198.

[23] Konstantinos I. Karantasis, Andrew Lenharth, Donald Nguyen, María J. Garzarán, and Keshav Pingali. 2014. Parallelization of reordering algorithms for bandwidth and wavefront reduction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New Orleans, Louisana) *(SC '14)*. IEEE Press, 921–932. doi:10.1109/SC.2014.80

[24] George Karypis and Vipin Kumar. 1997. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1997).

[25] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[26] Penporn Koanantakool, Ariful Azad, Aydin Buluç, Dmitriy Morozov, Sang-Yun Oh, Leonid Oliker, and Katherine Yelick. 2016. Communication-Avoiding Parallel Sparse-Dense Matrix-Matrix Multiplication. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 842–853. doi:10.1109/IPDPS.2016.117

[27] Amy N. Langville and Carl D. Meyer. 2006. A Reordering for the PageRank Problem. *SIAM J. Sci. Comput.* 27, 6 (jan 2006), 2112–2120. doi:10.1137/040607551

[28] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. 2020. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Trans. Parallel Distrib. Syst.* 31, 1 (jan 2020), 94–110. doi:10.1109/TPDS.2019.2928289

[29] Shigang Li, Kazuki Osawa, and Torsten Hoefler. 2022. Efficient quantized sparse matrix operations on tensor cores. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Dallas, Texas) *(SC '22)*. IEEE Press, Article 37, 15 pages.

[30] Seung Won Min, Kun Wu, Sitao Huang, Mert Hidayetoğlu, Jinjun Xiong, Eiman Ebrahimi, Deming Chen, and Wen-mei Hwu. 2021. Large graph convolutional network training with GPU-oriented data communication architecture. *Proc. VLDB Endow.* 14, 11 (jul 2021), 2087–2100. doi:10.14778/3476249.3476264

[31] Oguz Selvitopi, Benjamin Brock, Israt Nisa, Alok Tripathy, Katherine Yelick, and Aydın Buluç. 2021. Distributed-memory parallel algorithms for sparse times tall-skinny-dense matrix multiplication. In *Proceedings of the 35th ACM International Conference on Supercomputing* (<conf-loc>, <city>Virtual Event</city>, <country>USA</country>, </conf-loc>) *(ICS '21)*. Association for Computing Machinery, New York, NY, USA, 431–442. doi:10.1145/3447818.3461472

[32] Dahai Tang, Jiali Wang, Rong Chen, Lei Wang, Wenyuan Yu, Jingren Zhou, and Kenli Li. 2024. XGNN: Boosting Multi-GPU GNN Training via Global GNN Memory Store. *Proc. VLDB Endow.* 17, 5 (may 2024), 1105–1118. doi:10.14778/3641204.3641219

[33] Alok Tripathy, Katherine Yelick, and Aydın Buluç. 2020. Reducing communication in graph neural network training. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–14.

[34] Robert A. van de Geijn and Jerrell Watts. 1995. *SUMMA: Scalable Universal Matrix Multiplication Algorithm*. Technical Report. USA.

[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* 1050, 20 (2017), 10–48550.

[36] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[37] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2021. {GNNAdvisor}: An adaptive and efficient runtime system for {GNN} acceleration on {GPUs}. In *15th USENIX symposium on operating systems design and implementation (OSDI 21)*. 515–531.

[38] Yuke Wang, Boyuan Feng, Zheng Wang, Tong Geng, Kevin Barker, Ang Li, and Yufei Ding. 2023. MGG: Accelerating Graph Neural Networks with Fine-Grained Intra-Kernel Communication-Computation Pipelining on Multi-GPU Platforms. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 779–795. https://www.usenix.org/conference/osdi23/presentation/wang-yuke

[39] Yuke Wang, Boyuan Feng, Zheng Wang, Guyue Huang, and Yufei Ding. 2023. TC-GNN: Bridging Sparse GNN Computation and Dense Tensor Cores on GPUs. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 149–164. https://www.usenix.org/conference/atc23/presentation/wang-yuke

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[41] Carl Yang, Aydın Buluç, and John D. Owens. 2018. Design Principles for Sparse Matrix Multiplication on the GPU. In *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27 - 31, 2018, Proceedings* (Turin, Italy). Springer-Verlag, Berlin, Heidelberg, 672–687.

[42] Dongxu Yang, Junhong Liu, Jiaxing Qi, and Junjie Lai. 2022. WholeGraph: a fast graph neural network training framework with multi-GPU distributed shared memory architecture. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Dallas, Texas) *(SC '22)*. IEEE Press, Article 54, 14 pages.

[43] Xintian Yang, Srinivasan Parthasarathy, and P. Sadayappan. 2011. Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining. *Proc. VLDB Endow.* 4, 4 (jan 2011), 231–242.

[44] Katherine Yelick, Dan Bonachea, Wei-Yu Chen, Phillip Colella, Kaushik Datta, Jason Duell, Susan L. Graham, Paul Hargrove, Paul Hilfinger, Parry Husbands, Costin Iancu, Amir Kamil, Rajesh Nishtala, Jimmy Su, Michael Welcome, and Tong Wen. 2007. Productivity and performance using partitioned global address space languages. In *Proceedings of the 2007 International Workshop on Parallel Symbolic Computation* (London, Ontario, Canada) *(PASCO '07)*. Association for Computing Machinery, New York, NY, USA, 24–32.

[45] Serif Yesil, Azin Heidarshenas, Adam Morrison, and Josep Torrellas. 2023. WISE: Predicting the Performance of Sparse Matrix Vector Multiplication with Machine Learning. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Montreal, QC, Canada) *(PPoPP '23)*. Association for Computing Machinery, New York, NY, USA, 329–341. doi:10.1145/3572848.3577506

[46] Serif Yesil, José E. Moreira, and Josep Torrellas. 2022. Dense dynamic blocks: optimizing SpMM for processors with vector and matrix units using machine learning techniques. In *Proceedings of the 36th ACM International Conference on Supercomputing* (Virtual Event) *(ICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 27, 14 pages. https://doi.org/10.1145/3524059.3532369

[47] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 5171–5181.

[48] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence* (New Orleans, Louisiana, USA) *(AAAI'18/IAAI'18/EAAI'18)*. AAAI Press, Article 544, 8 pages.