Arrow Matrix Decomposition: A Novel Approach for Communication-Efficient Sparse Matrix Multiplication

Lukas Gianinazzi
Department of Computer Science
ETH Zurich

Alexandros Nikolaos Ziogas
Department of Electrical Engineering
ETH Zurich

Langwen Huang
Department of Computer Science
ETH Zurich

Piotr Luczynski
Department of Computer Science
ETH Zurich

Saleh Ashkboos
Department of Computer Science
ETH Zurich

Florian Scheidl
Department of Computer Science
ETH Zurich

Armon Carigiet
Department of Computer Science
ETH Zurich

Chio Ge
Department of Computer Science
ETH Zurich

Nabil Abubaker
Department of Computer Science
ETH Zurich

Maciej Besta
Department of Computer Science
ETH Zurich

Tal Ben-Nun
Department of Computer Science
ETH Zurich

Torsten Hoefler
Department of Computer Science
ETH Zurich

Abstract

We propose a novel approach to iterated sparse matrix dense matrix multiplication, a fundamental computational kernel in scientific computing and graph neural network training. In cases where matrix sizes exceed the memory of a single compute node, data transfer becomes a bottleneck. An approach based on dense matrix multiplication algorithms leads to suboptimal scalability and fails to exploit the sparsity in the problem. To address these challenges, we propose decomposing the sparse matrix into a small number of highly structured matrices called *arrow* matrices, which are connected by permutations. Our approach enables communication-avoiding multiplications, achieving a polynomial reduction in communication volume per iteration for matrices corresponding to planar graphs and other minor-excluded families of graphs. Our evaluation demonstrates that our approach outperforms a state-of-the-art method for sparse matrix multiplication on matrices with hundreds of millions of rows, offering near-linear strong and weak scaling.

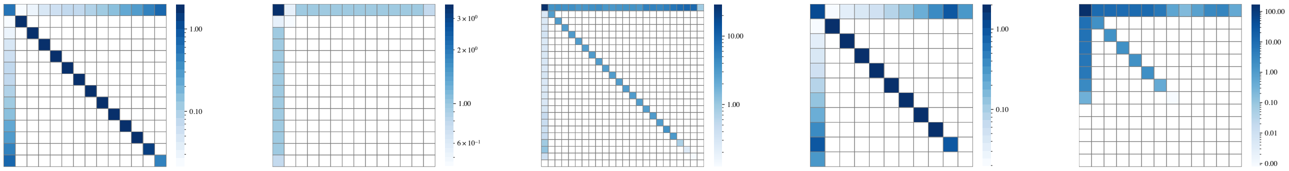
1 Introduction

Iterated sparse-dense matrix multiplications (SpMM) have numerous applications, including the training and inference of graph neural networks [47] and the computation of eigenvectors [24, 32, 38]. As the matrices arising from these problems are often too large to fit into the memory of a single GPU, they need to be *decomposed* and solved on compute clusters [9, 11] or processed in several batches [50]. Data movement becomes the crucial bottleneck for such sparse workloads [11, 45, 50].

Two existing approaches stand out. The first line adapts efficient algorithms designed for dense matrices to the sparse domain [9, 45, 47]. These techniques offer the advantage of low overhead and simplicity, but they encounter limitations due to their origin in dense algorithms. Consequently, their ability to fully harness the available processing power in the sparse matrix regime is limited. This deficiency forces a compromise between latency, bandwidth, and memory.

The second line of work focuses on matrix reorderings [1, 15, 16, 21, 22]. This approach involves permuting the rows and columns of a matrix to enhance computational and communication efficiency. However, these methods often rely on heuristic strategies [16, 23, 48] and are constrained by unfavorable lower bounds and complexity results [21, 22, 35, 39]. Of particular concern is the sensitivity of these bounds to the maximum degree and the diameter of the graph. This drawback is especially pronounced in scale-free graphs and those with skewed degree distributions.

To overcome these limitations, we provide a *matrix decomposition* approach to sparse-matrix times dense-matrix operations. The sparse input matrix $\mathbf{A}$ is *decomposed* into a small number of matrices with bounded *arrow-width* b, meaning that all non-zeros are concentrated in the first b rows, columns, and a band of width b around the diagonal. Formally, $\mathbf{A}$ has arrow-width b if for all $i > b$ and $j > b$ we have that if $A_{ij} \neq 0$, then $|i - j| \leq b$. Arrow-width generalizes the notion of an arrowhead matrix [26], for which $b = 1$. We decompose a matrix $\mathbf{A}$ into a sum of matrices of the form $\mathbf{A} = \sum_{i=1}^{l} \mathbf{P}_{\pi_i} \mathbf{B}_i \mathbf{P}_{\pi_i}^{\top}$, where each matrix $\mathbf{B}_i$ has arrow-width at most b and each matrix $\mathbf{P}_{\pi_i}$ is a permutation matrix. See Figure 1 for an example of the $\mathbf{B}_0$ matrices. Given this decomposition, the computation can be performed on those



(a) GenBank 68M **(b)** MAWI 69M **(c)** Webbase 118M **(d)** OSM Europe 51M **(e)** GAP-twitter 62M

Figure 1. Non-zero structure of the first matrix $\mathbf{B_0}$ in an arrow matrix decomposition for matrices from the SuiteSparse Matrix Collection. The color indicates the number of non-zeros per row; white blocks are empty. Each block has 5 million rows.

regularly-structured matrices in a communication-efficient way and, finally, aggregated.

In contrast to traditional bandwidth-minimization, we overcome the fundamental lower bounds with our decomposition. In particular, while any adjacency matrix of a low-diameter tree has $\Omega(n/\log n)$ bandwidth, we show, in particular, how to decompose the adjacency matrix of such a tree into $O(\log n)$ matrices of bandwidth $O(1)$. We show how to construct such an *arrow matrix decomposition* for several sparsity structures, as characterized by the graphs they represent. The main idea is to use the relationship with *minimum linear arrangement* [14, 20, 41]. Moreover, we prove that the pruning of high-degree vertices enabled by the arrow shape provides a *polynomial* improvement in the communication volume in power law graphs.

Our proposed approach is efficient and can construct the arrow matrix decomposition in polynomial time for a variety of families of graphs, including trees, chordal graphs, planar graphs, and, more generally, K_r-minor free graphs. Additionally, we present a linear-time heuristic based on efficient layouts of random spanning trees that can effectively decompose real-world graphs such as biological and web traffic graphs into a small number of increasingly sparse matrices. In our evaluation, we decompose 13 of the largest matrices in the SuiteSparse matrix collection into just two to four matrices of low arrow width, whereas their maximum bandwidth can exceed 90% of the number of columns.

Our approach provides significant reductions in communication costs of sparse matrix-matrix multiplication compared to traditional approaches. In our experiments, we demonstrate the scalability of our approach by testing it on several sparse matrices with over 50 million rows. On 128 GPUs, our approach reduces the communication volume by $3-5$ times compared to a 1.5D decomposition, a state-of-the art approach for SpMM [45, 47]. Our approach uses less memory per compute node and effortlessly processes sparse matrices with over 200 million vertices and dense right-hand side matrices with a larger number of columns.

Furthermore, our evaluation shows that on a related family of sparse matrices from the same dataset, the runtime of our approach only grows by $2.3-6.2\%$ as we scale from a dataset with 18 million rows to a dataset with over 200 million rows when the ratio of vertices over GPUs remains constant. Overall, our approach shows better scaling both with the number of sparse and the number of dense columns. We demonstrate good strong scaling up to 256 compute nodes on matrices with over 200 million rows where we show speedups of 5.3x-14.3x compared to the 1.5D baseline and 1.7x-58x compared to the 1D hypergraph partitioning baseline.

2 Background

Graphs. Consider an undirected graph G with n vertices $V(G)$ and m edges $E(G)$. The subgraph of G induced by $S \subseteq V$ is $G[S]$. The degree of a vertex v is $\deg(v)$ and the maximum degree of G is $\Delta(G)$, or Δ for short when G is clear from the context. Given a rooted tree, the set of descendants of a vertex v is $v^{\downarrow}$. The diameter of a graph G is $D(G)$. If there is a permutation π of the vertices $V(G)$ such that $\max_{(u,v)\in E(G)} |\pi(u) - \pi(v)| = w$, G has *bandwidth* w [16, 21].

Matrices. We denote the adjacency matrix of G by $\mathbf{A}$, meaning that the number of non-zeros in $\mathbf{A}$ is $nnz(\mathbf{A}) = m$. We consider a dense *tall and skinny feature matrix* $\mathbf{X} = \mathbf{X}_0 \in R^{n \times k}$, with k *features* where $k \ll n$ [45]. Our goal is to compute the matrix iteration $\mathbf{X}_{t+1} = \sigma(\mathbf{A}\mathbf{X_t})$ for some number of steps T. The function σ denotes some application-dependent element-wise function or normalization operation. We will focus our attention on the computation of the product $\mathbf{Y} = \mathbf{A}\mathbf{X}$ in the situation where $T \gg 1$. This means that we can afford to preprocess the problem and amortize the cost over the iterations. A matrix has *bandwidth* w if its nonzero elements are at most w away from the diagonal. That is, the matrix $\mathbf{A}$ has bandwidth w if for all i, j we have that if $A_{ij} \neq 0$, then $|i - j| \leq w$.

The α-β Model of Computation. We consider p processors that can each send and receive one message simultaneously. Sending a message of size s has a latency cost α and a bandwidth cost $\beta \cdot s$ [13, 45]. A message M' depends on another message M if its content, recipient, or existence depends on M. The latency cost of a computation is the largest sum of latency costs along a chain of dependent messages. The bandwidth cost of an algorithm is the largest total bandwidth cost over all processors.

3 Related Work

Based on parallel algorithms for dense matrix multiplications [44], Selvitopi et al. [45] detail several approaches to tile the sparse-times-dense-skinny SpMM, which trade off communication cost with storage.

1.5D** A**-stationary. The 1.5D A-stationary algorithm [45, 47] arranges the processors in a $\frac{p}{c} \times c$ grid, where c is the replication factor. It slices the matrix $\mathbf{A}$ into tiles of size $\frac{nc}{p} \times \frac{n}{c}$ (splitting it both by row and column), with each processor assigned a single tile. It splits the feature matrix $\mathbf{X}$ along the row dimension into tiles of size $\frac{nc}{p} \times k$, meaning that each tile is replicated in the c processors of a grid row. The $\frac{p}{c}$ processor of a grid column compute together a single $\frac{nc}{p} \times k$-sized tile of the output $\mathbf{Y}$, with each processor holding a partial tile of the same size. To do so, they require $\frac{p}{c^2}$ tiles of $\mathbf{X}$. The computation happens in $\frac{p}{c^2}$ rounds, broadcasting one of those tiles along the grid column, so each processor needs to hold only one extra $\mathbf{X}$ tile at any point of the execution. After executing all rounds, each grid column performs an all-reduce operation to compute the full tile. The communication cost is $O\big(\alpha\frac{p}{c^2}\log p + \beta\big(\frac{nk}{c} + \frac{nkc}{p}\big)\big)$ [45]. The total storage cost for all processors is $O\left(m + cnk\right)$. For the special case $c = 1$, the algorithm is equivalent to a 1D version with communication cost $O\big(\alpha p \log p + \beta\big(\frac{nk}{\sqrt{p}} + nk\sqrt{p}\big)\big)$ and total storage cost $O\left(m + nk\right)$. For *full replication*, $c = \sqrt{p}$, the communication cost is $O\big(\alpha \log p + \beta\frac{nk}{\sqrt{p}}\big)$, and the total storage cost is $O\left(m + \sqrt{p}nk\right)$. High values of the replication factor thus reduce the communication cost but increase storage.

2D** A**-stationary. In contrast to the 1.5D algorithm, the feature matrix $\mathbf{X}$ is sliced by columns as well in the 2D $\mathbf{A}$-stationary algorithm. However, this requires computing the result in $\sqrt{p}$ phases. This both reduces the size of the local SpMM operations (which leads to decreased local SpMM performance [45]) and leads to a higher communication cost. Overall, the approach needs to store $O(n/p)$ rows of the feature matrix per processor and has a total communication cost of $O\big(\alpha\sqrt{p}\log p + \beta\frac{nk\log p}{\sqrt{p}}\big)$. Compared to the 1.5D algorithm with $c = \sqrt{p}$, this improves the storage by a factor of $\sqrt{p}$ but increases the latency cost by a factor of $\Theta(\sqrt{p})$ and the bandwidth cost by a factor of $\Theta(\log p)$. Previous work found $2D$ decompositions to scale less favorably compared to 1.5D algorithms in for skinny feature matrices [45, 47].

2D** Y**-stationary. In the case where the matrix $\mathbf{A}$ has fewer columns than rows, an algorithm that keeps the result matrix $\mathbf{Y}$-stationary improves communication costs [45]. As we focus on the case where $\mathbf{A}$ is square (due to being an adjacency matrix of a graph), this approach does not provide any communication cost benefits in our settings.

Square Dense Matrices. For the case where the feature matrix is also square, Koanantakool et al. [30] evaluate several communication avoiding decomposition schemes. In our work, we focus on approaches specifically designed for the case where the feature matrix is skinny, but tall.

Graph Partitioning. Several (hyper-)graph-partitioning approaches have been proposed for sparse matrix-vector [8, 12, 37, 46] and sparse matrix-matrix [6, 19] multiplication. Our matrix decomposition approach is not based on graph partitioning. Instead, our objective function is based on minimum linear arrangement [14]. Our approach avoids communication load imbalance between the partitions as we use the 1.5D A-stationary algorithm, specialized to arrow matrices.

Graph Reordering. If the matrix $\mathbf{A}$ has low bandwidth, one can efficiently compute the sparse matrix-matrix product $\mathbf{AX}$ with a small communication cost and low storage requirements using a 1.5D $\mathbf{A}$-stationary algorithm. The bandwidth of G is at least $\lceil\frac{n-1}{D(G)}\rceil$ and at least $\left\lceil\frac{\Delta(G)}{2}\right\rceil$ [15] meaning that low-diameter networks [3] and power-law networks [2] have high bandwidth. Computing the bandwidth is NP-hard [22], even on bounded degree trees [15].

4 Arrow Matrix Decomposition

Our approach is to *decompose* the graph G into as few graphs as possible, each having low *arrow-width*. Then, we efficiently perform an SpMM on each of the arrow matrices and only need to aggregate the partial results. We demonstrate in Section 5.6 that the arrow shape is necessary to effectively represent graphs with skewed degree distributions.

In terms of matrices, this results in an *arrow matrix decomposition* of the form $\mathbf{A} = \sum_{i=1}^{l} \mathbf{P}_{\pi_i}\mathbf{B}_i\mathbf{P}_{\pi_i}^{\top}$, where each matrix $\mathbf{B}_i$ has arrow-width at most b and each matrix $\mathbf{P}_{\pi_i}$ is a permutation matrix corresponding to a permutation π_i of the vertices of the graph. We call such a decomposition a b-arrow matrix decomposition of order l. Then, we can compute $\mathbf{Y} = \mathbf{AX}$ as

$$\mathbf{AX} = \sum_{i=1}^{l} \mathbf{P}_{\pi_i}\left(\mathbf{B}_i(\mathbf{P}_{\pi_i}^{\top}\mathbf{X})\right) \quad , \tag{1}$$

meaning that we have reduced the computation of the product onto a series of arrow matrix multiplies, permutations, and reductions.

It is desirable for an arrow matrix decomposition that the number of non-zero rows decreases quickly with i, as this reduces storage and communication costs. Note that in this case, we can always collect the non-zeros at the top of the matrix. If the total number of non-zeros in $\mathbf{B}_{i+1}$ is at most $\frac{1}{x}$ times the total number of non-zeros in $\mathbf{B}_i$, then we say the arrow matrix decomposition is x*-compacting*. For $x > 1$, an x-compacting arrow decomposition has order $O(1 + \log_x n)$. In our experiments, we will construct order $1 - 3$ decompositions.

4.1 Distributed SpMM Algorithm

We present a distributed algorithm for SpMM using an arrow matrix decomposition. In Section 6, we analyze the data

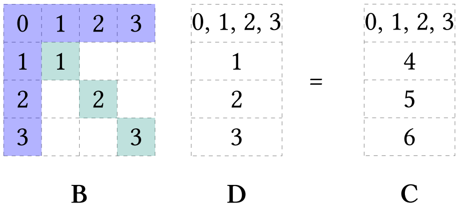


Figure 2. In a distribution of an arrow matrix **B**, each tile of **B** is $b \times b$ and each tile of **D** and **C** is $b \times k$. The numbers indicate the process ranks holding or contributing to the tile.

movement and storage requirements of this algorithm in the $\alpha - \beta$ model. In particular, it improves bandwidth cost and storage requirements by a factor of $\Theta(\sqrt{p})$ at a similar latency cost compared to a fully replicated 1.5D decomposition.

We present a distributed algorithm for SpMM using an arrow matrix decomposition. In Section 6, we analyze the data movement and storage requirements of this algorithm in the $\alpha - \beta$ model. This method notably improves bandwidth cost and storage efficiency by a factor of $\Theta(\sqrt{p})$, while maintaining comparable latency costs to a fully replicated 1.5D decomposition.

Algorithm 1: Arrow Matrix Multiply

```
Data: $\mathbf{B}$, $\mathbf{D}$, rank $r$
Result: $\mathbf{C}=\mathbf{BD}$
Broadcast($\mathbf{D}^{(0)}$, root=0)
$\mathbf{C}^{(0)} = \mathbf{B}^{(0,i)}\mathbf{D}^{(i)}$
Reduce($\mathbf{C}^{(0)}$, root=0)
if $r > 0$ then
    $\mathbf{C}^{(r)} = \mathbf{B}^{(r,0)}\mathbf{D}^{(0)} + \mathbf{B}^{(r,r)}\mathbf{D}^{(r)}$
return $\mathbf{C}^{(r)}$
```

Arrow Matrix SpMM. Let us begin with how to compute the product **BD** = **C** when **B** has arrow width b. The arrow matrix's non-zeros appear in three bands, leading a 1.5D decomposition to result in most tiles of **B** being zero, thus yielding a communication-efficient algorithm. To further enhance efficiency, we consider a block-diagonal band.

We tile the $n \times n$ matrix **B**, with arrow width b, into $b \times b$ tiles, indexed as $\mathbf{B}^{(i,j)}$. Due to the arrow structure, the non-zeros occur in three types of tiles: $\lceil \frac{n}{b} \rceil$ row tiles ($\mathbf{B}^{(0,j)}$), $\lceil \frac{n}{b} \rceil$ column tiles ($\mathbf{B}^{(i,0)}$ for $i > 0$), and $\lceil \frac{n}{b} \rceil$ diagonal tiles ($\mathbf{B}^{(i,i)}$). The matrices **D** and **C** are sliced into $b \times k$ tiles, indexed as $\mathbf{D}^{(i)}$ and $\mathbf{C}^{(i)}$. Each rank i initially holds three tiles of **B** and one slice of **D**, as depicted in Figure 2. The multiplication using an arrow matrix is detailed in Algorithm 1 and involves two collective communication operations.

SpMM Algorithm. Next, we describe how to multiply with a matrix given its arrow decomposition. Each rank is assigned to one of the matrices of the decomposition. Each matrix is distributed as in Figure 2. Initially, the first matrix alone contains the input matrix **X**, with each of its ranks r holding a distinct block $\mathbf{X}^{(r)}$. This block is then sent to the subsequent matrix in the sequence, propagating through each matrix. This propagation utilizes a specific permutation, $\pi_{j+1} \circ \pi_j^{-1}$, to shuffle the rows when transmitting from matrix j to matrix $j+1$. Each matrix then computes its local product as described in Algorithm 1, resulting in a partial output $\mathbf{Y_j}$, with the segment $\mathbf{Y_j^{(j)}}$ stored by the rank r assigned to matrix j. Finally, the partial results $\mathbf{Y_j}$ are aggregated in reverse order, following the opposite pattern of the input matrix distribution. For a detailed explanation of this procedure, refer to Algorithm 2.

Algorithm 2: Arrow Decomposition Multiply

```
Data: Arrow Decomposition $\mathbf{A} = \sum_{i=1}^{l} \mathbf{P}_{\pi_i}\mathbf{B}_i\mathbf{P}_{\pi_i}^{\top}$, $\mathbf{X}$,
      rank $r$, where rank $r$ belongs to the $j$-th arrow matrix.
Result: Y=AX
for $k \leftarrow 1$ to $l$ do
    if $k == j$ then
        Send $\mathbf{X}^{(r)}$ to matrix $j+1$
    if $k+1 == j$ then
        Receive $\mathbf{X}^{(r)} = (\mathbf{P}_{\pi_j}^{\top}\mathbf{X})^{(r)}$ from matrix $j-1$
$\mathbf{Y_j^{(r)}} = (\mathbf{B}_j(\mathbf{P}_{\pi_j}^{\top}\mathbf{X}))^{(i)}$
for $k \leftarrow l$ to 1 do
    if $k == j$ then
        Send $\mathbf{Y}^{(r)}$ to matrix $j-1$
    if $k-1 == j$ then
        Receive $\hat{\mathbf{Y}}^{(r)} = (\sum_{i=r+1}^{l} \mathbf{P}_{\pi_i}\mathbf{Y}_i)^{(r)}$ from $j+1$
        $\mathbf{Y}^{(r)}$ += $\hat{\mathbf{Y}}^{(r)}$
```

5 Constructing the Decomposition

When we construct an arrow matrix decomposition, there is a trade-off between the time to compute the decomposition and its compactness. We frame the decomposition problem in a graph-theoretic language, which allows us to obtain algorithms that are polynomial time and provide strong bounds on certain sparsity structures.

The high-level idea is to consider the matrix as a graph and find a permutation of its vertices, a so-called *linear arrangement*, such that many edges connect vertices that are close in the order of the permutation. We minimize a cost function that sums over the distances of the edges in the linear arrangement. Because high-degree vertices add high costs, we collect those at the beginning of the order. Then, we construct a remainder graph consisting of the edges that are much further apart than the average cost of the solution and proceed recursively.

In addition to provable polynomial time bounds on several families of graphs, we present a near-linear time heuristic

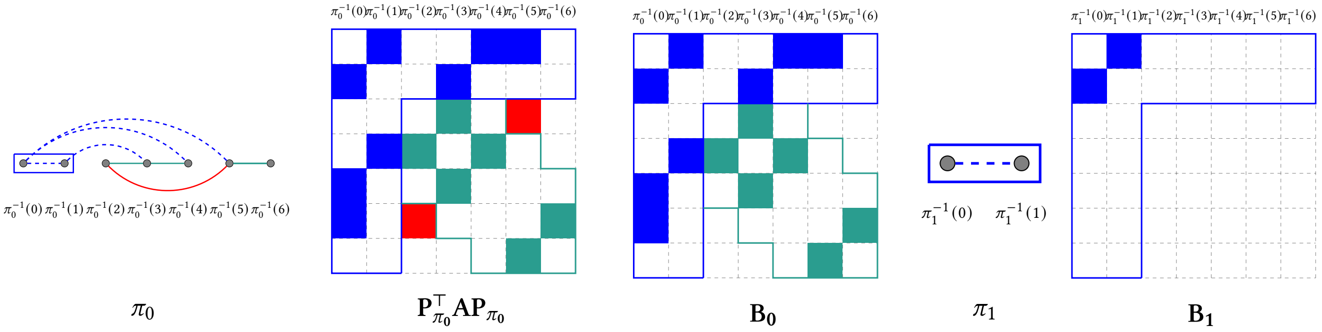

$\pi_0^{-1}(0)\,\pi_0^{-1}(1)\,\pi_0^{-1}(2)\,\pi_0^{-1}(3)\,\pi_0^{-1}(4)\,\pi_0^{-1}(5)\,\pi_0^{-1}(6)$
π_0
$\pi_0^{-1}(0)\pi_0^{-1}(1)\pi_0^{-1}(2)\pi_0^{-1}(3)\pi_0^{-1}(4)\pi_0^{-1}(5)\pi_0^{-1}(6)$
$\mathbf{P}_{\pi_0}^\top \mathbf{A} \mathbf{P}_{\pi_0}$
$\pi_0^{-1}(0)\pi_0^{-1}(1)\pi_0^{-1}(2)\pi_0^{-1}(3)\pi_0^{-1}(4)\pi_0^{-1}(5)\pi_0^{-1}(6)$
$\mathbf{B}_0$
$\pi_1^{-1}(0)$ $\pi_1^{-1}(1)$
π_1
$\pi_1^{-1}(0)\pi_1^{-1}(1)\pi_1^{-1}(2)\pi_1^{-1}(3)\pi_1^{-1}(4)\pi_1^{-1}(5)\pi_1^{-1}(6)$
$\mathbf{B}_1$

Figure 3. LA-Decompose produces a linear arrangement π_0 of the vertices of the graph that corresponds to the sparsity structure of the input matrix. This creates three parts in the matrix (1) A flipped 'L' shape that contains the highest degree vertices (in blue), (2) a band around the diagonal (in green), and (3) the remainder (in red). The first two parts form the first matrix $\mathbf{B}_0$ of the decomposition. The rest of the decomposition proceeds recursively on the remainder.

based on high-quality linear arrangements of random spanning trees. Because of its scalability to hundreds of millions of nodes, we use this random spanning forests approach to compute our decompositions in our evaluation.

5.1 LA-Decompose

Any reordering of the matrices and rows of a square matrix can be viewed as a permutation of the vertices of its graph. Such a permutation is called a *linear arrangement*. Our goal is to find a permutation that leads to most of the non-zeros being close to the diagonal. Hence, we consider the cost function $\lambda_\pi(G)$ of the linear arrangement π as $\lambda_\pi(G) = \sum_{(u,v)\in E(G)} |\pi(u) - \pi(v)|$. If the graph G is clear from the context, we omit G from the notation. A linear arrangement of G with the smallest cost is a *minimum linear arrangement* (MLA) [14, 20, 41]. Computing a minimum linear arrangement is NP-hard, however, it can be approximated in polynomial time within a $O(\sqrt{\log n}\log\log n)$ factor [14] and solved exactly in polynomial time on trees [4] and chordal graphs [42]. Note that a graph with bandwidth b has a linear arrangement of cost at most mb or nb^2. In contrast, we show there are graphs with a linear arrangement of cost $O(n\log n)$ but bandwidth $\Omega(n/\log n)$.

The idea of our algorithm is that a linear arrangement π of cost $\lambda_\pi(G)$ concentrates a constant fraction of the non-zeros along a $O(\lambda_\pi(G)/m)$-wide band along the diagonal. Removing this portion and repeating the process until no edges are left leads to a compact arrow decomposition. We present a framework for computing an arrow matrix decomposition using a linear arrangement, called LA-Decompose($\mathbf{A}$, b):

We are given a matrix $\mathbf{A}$ and a desired arrow-width $b \geq 2$. Set $\mathbf{A}_0 = \mathbf{A}$ and $i = 0$. Until the number of non-zeros in $\mathbf{A}_i$ is at most $2b$, repeat the following steps:

1. Place the b highest degree vertices V_i^h at the beginning of the linear arrangement π_i.
2. Compute a linear arrangement π_i' of the induced subgraph $G_i' = G_i[V_i \backslash V_i^h]$ of $\mathbf{A}_i$ and concatenate it to π_i.
3. Set $\mathbf{B}_i$ equal to the submatrix of $\mathbf{P}_{\pi_i}^\top \mathbf{A}_i \mathbf{P}_{\pi_i}$ consisting of the first b rows and columns and a symmetric b-wide band around the diagonal.
4. Set $\mathbf{A}_{i+1} = \mathbf{A}_i - \mathbf{P}_{\pi_i}\mathbf{B}_i\mathbf{P}_{\pi_i}^\top$ and increment i.

Observe that the matrices $\mathbf{A}_i$ do not need to be constructed explicitly and one can instead work on the corresponding graphs. See Figure 3 for an illustration of the approach.

Lemma 1. *For $x = \frac{bm}{\max_i \lambda_{\pi_i'}(G_i')}$, LA-Decompose($\mathbf{A}$, b) computes an x-compacting b-arrow matrix decomposition.*

Proof. The arrow-width of any of the matrices $\mathbf{B}_i$ is b by construction. Moreover, in step i the average distance from the diagonal is at most $\frac{\lambda_{\pi_i'}(G_i')}{m}$. No more than a $\frac{1}{x}$ fraction of the entries can be more than x-times the average away from the diagonal. Hence, in each iteration at most a $\frac{1}{x}$ fraction of the edges remain, i.e., the decomposition is x-compacting. □

Lemma 1 means that the number of nonzeros decreases geometrically for the matrices in the decomposition as long as b is larger than the average cost of an edge in the linear arrangements π_i'. In our experiments, this will always be the case for our choices of b.

Next, we will show efficient algorithms for linear arrangements and prove lower bounds for certain families of graphs. The bounds on the cost of the linear arrangement will depend necessarily on the maximum degree Δ of the graph. This is why we removed the highest degree vertices before computing the linear arrangement in LA-Decompose. In Section 5.6, we show how the pruning of high-degree vertices improves the arrow decomposition in graphs with a power law degree distribution, which occur in real-word graphs [2].

Table 1. Bounds on the cost of a linear arrangement.

Graph family	Linear arrangement cost
K_r-minor free (§5.2, [28])	$O(n\Delta\sqrt{n}r)$
Planar (§5.2, [33])	$O(n\Delta\sqrt{n})$
Treewidth τ (§5.2 [10, 43])	$O(n\tau \log n)$
Series-Parallel (§5.2)	$O(n \log n)$
Trees (§5.4)	$n\Delta$

5.2 Linear Arrangement using Separators

We show how to construct linear arrangements efficiently by recursively partitioning the graph. This allows us to obtain bounds on the cost of a linear arrangement for several families of graphs that can be separated efficiently.

A set of vertices S whose removal leaves the graph with connected components of size at most $\frac{2}{3}n$ is a $\frac{2}{3}$-separator. For any positive integer $k \leq n$, let $s_k(G)$ be the smallest number such that all subgraphs of G containing at most k vertices have a $\frac{2}{3}$-separator of size $s_k(G)$. The *separation number* $s(G)$ of G is $s(G) = s_n(G)$. Small separators can be constructed, in particular, for clique-minor free graphs [28] and bounded treewidth graphs [10, 43].

Separator-LA(G) constructs a linear arrangement using separators recursively:

1. Compute a $\frac{2}{3}$-separator S of the current subgraph G.
2. Place the vertices of S at the beginning of the linear order.
3. Then, place the connected components of $G[V(G)\backslash S]$ that remain after removing S in increasing order after S. Within each connected component, place vertices recursively using Separator-LA.

Lemma 2. *Separator-LA(G) produces a linear arrangement of cost $O(n\Delta s(G) \log n)$. If $s_n(G) \in \Theta(n^\epsilon)$ for some constant $\epsilon > 0$, the linear arrangement has cost $O(n\Delta s_n(G))$.*

Proof. The cost at a particular level of recursion is at most $n\Delta|S| \leq n\Delta s_n(G)$. Let $n_0, \ldots, n_i$ be the sizes of the connected components of the graph $G[V(G)\backslash S]$ after removing S. The cost $\lambda(n)$ of the linear arrangement on n vertices is at most:

$$\lambda(n) \leq O(n\Delta s_n(G)) + \sum_i \lambda(n_i) \ ,$$

which solves to $O(\Delta n s_n(G) \log n)$ because the depth of the recursion is $O(\log n)$. Note that if $s_n(G) \in \Theta(n^\epsilon)$, $\sum_i s_{n_i}(G) < s_n(G)/2$. □

See Table 1 for a summary of the bounds obtained by Separator-LA on various families of graphs.

5.3 Linear Arrangements using Random MSTs

For datasets with hundreds of millions of nodes, it is crucial to have an algorithm that uses *near-linear* time. Computing separators in K_r-minor-free graphs takes $\Omega(m\sqrt{n})$ time using state-of-the-art algorithms [40], which becomes prohibitive for graphs with hundreds of millions of vertices. We propose a linear arrangement scheme using a *random spanning forest* of the input graph:

1. Construct a weighted graph G' by drawing edge weights independently from the standard uniform distribution.
2. Compute a minimum spanning forest F of G'.
3. Compute a linear arrangement of each tree in the forest F in decreasing order of size and concatenate them.

In our experiments in Section 7, we evaluate the linear arrangement using random forests and demonstrate its efficacy on real-world datasets. As trees have separation number 2, we directly get a linear arrangement of the spanning trees of cost $O(n\Delta \log n)$ using Separator-LA. However, improving the quality of the linear arrangement of trees is possible.

5.4 Linear Arrangement of Trees

In this section, we show an improvement in the cost of a linear arrangement over Separator-LA by a factor of $\Theta(\log n)$ for trees. We can get a tighter bound on the arrow width of an arrow decomposition of a tree with the following layout, which we use in our experiments: Place the root at the first position. Then, sort the children subtrees by size and arrange these subtrees one after the other in this order. Arrange each subtree recursively. This arrangement π is called *smallest-first order.* Instead of arguing about the cost of the linear arrangement and then using that most edges are close to the average, we directly argue about how many edges are within a $x\Delta$ wide band around the diagonal. For every edge (u, v) in the $x\Delta$-wide band around the diagonal, we have that $|\pi(u) - \pi(v)| \leq x\Delta$.

Lemma 3. *In smallest-first order π of a tree T, at least*

$$\min\left(n - 1, \lceil \frac{(x-1)\cdot(n-1)}{x} \rceil + 1\right)$$

edges are within an $x\Delta$-wide band around the diagonal.

Proof. Observe that the vertices of every subtree are listed consectively in the linear order π. Hence, we can use strong induction on the number of edges in the tree T. We root the trees at an arbitrary vertex. For a vertex v, let $E(v)$ be the set of edges in the subtree rooted at v. Let $E_x(v)$ be the set of edges in $E(v)$ within the $x\Delta$ band around the diagonal. Let $P(v)$ be the predicate

$$P(v) \equiv \text{If } |E(v)| \geq x \text{, then at least } \left\lceil \frac{x-1}{x} |E(v)| \right\rceil + 1 \text{ edges}$$
$$\text{in } E(v) \text{ are within a } x\Delta \text{ band around the diagonal} \ .$$

Note thay if the tree has less than x edges, then all its edges are within an $x\Delta$ band around the diagonal. We prove inductively that $P(v)$ holds for all trees. As a base case, consider an arbitrary tree rooted at v with $x \leq E(v) \leq x\Delta$ edges. For such a tree, every edge is within a $x\Delta$ band around the diagonal and $|E(v)| \geq \lceil \frac{x-1}{x} \cdot |E(v)| \rceil + 1$. For the inductive step, consider consider a tree rooted at v where $|E(v)| > x\Delta$. By induction, for each $v' \neq v$ in the subtree rooted at v we may assume that $P(v')$ holds. We proceed by case distinction on the degree of v.

Case $\deg(\mathbf{v}) = \mathbf{1}$ Let u be the child of v. By definition of *smallest-first* order, the distance between u and v in the linear arrangement π is 1 and therefore the $\{v, u\} \in E_x(v)$. Notice that $E_x(v) = \{v, u\} \cup E_x(u)$. Because $|E(u)| = |E(v)| - 1 \geq x$, we conclude by $P(u)$ that

$$\begin{aligned} E_x(v) &= 1 + |E_x(u)| \\ &\geq 1 + \left\lceil \frac{x-1}{x} \cdot |E(u)| \right\rceil + 1 \\ &\geq \left\lceil \frac{x-1}{x} \cdot |E(v)| \right\rceil + 1 \ . \end{aligned}$$

Case $\deg(\mathbf{v}) \geq \mathbf{2}$. Let $C(v) = u_1, ..., u_{\deg(v)}$ be the list of children of v, sorted in increasing order by the size of their subtree. Let i be the largest index such that $|\pi(v) - \pi(c_i)| \leq x\Delta$, i.e., $\{v, u_i\}$ is in the $x\Delta$ band. Notice that $\forall j \leq i \ |\pi(v) - \pi(u_j)| \leq x\Delta$, by the definition of *smallest-first* order. Because v's subtree is of size greater than $x\Delta$, we have $i < \deg(v)$. It now follows that:

$$\begin{aligned} &|E_x(v)| \\ &= i + \sum_{w=1}^{\deg(v)} |E_x(u_w)| \\ &= \left(\sum_{w=1}^{i} |E_x(u_w)| + 1 \right) + \sum_{w=i+1}^{\deg(v)} |E_x(u_w)| \\ &\geq \left(\sum_{w=1}^{i} \left\lceil \frac{x-1}{x} |E(u_w)| \right\rceil + 1 \right) + \sum_{w=i+1}^{\deg(v)} \left\lceil \frac{x-1}{x} |E(u_w)| \right\rceil + 1 \\ &\geq \sum_{w=1}^{\deg(v)} \left\lceil \frac{x-1}{x} |E(u_w)| \right\rceil + 1 \\ &\geq \deg(v) + \sum_{w=1}^{\deg(v)} \left\lceil \frac{x-1}{x} |E(v)| \right\rceil \\ &\geq \deg(v) + \frac{x-1}{x} |E(u_v)| \\ &\geq \left\lceil \frac{x-1}{x} |E(v)| \right\rceil + 1 \ . \end{aligned}$$

We now explain the first inequality. First, we look at vertices in $\sum_{w=1}^{i} (|E_x(u_w)| + 1)$. If $|E(u_w)| \geq x$, we have by induction hypothesis that $|E_x(u_w)| + 1 \geq \lceil \frac{x-1}{x} |E(u_w)| \rceil + 1$. If u_w has less than x edges in its subtree, we cannot use the induction hypothesis, but we still have $|E_x(u_w)| + 1 = |E(u_w)| + 1 \geq \lceil \frac{x-1}{x} |E(u_w)| \rceil + 1$. Next, observe that $\sum_{w=1}^{i} |E(u_w)| \geq x\Delta$ because otherwise $\{v, w_{i+1}\}$ would be in the $x\Delta$ band. This means that at least one child u_w with $w \leq i$ has to have at least x edges in its subtree. Because the subtrees are sorted by size, it follows that all children u_w with $w > i$ satisfy $|E(u_w)| \geq x$ and we can use the induction hypothesis. . □

We immediately get a more efficient x-compacting arrow matrix decomposition for trees using LA-Decompose:

Corollary 1. *A tree has an x-compacting $x\Delta$-arrow decomposition that can be computed in $O(n)$ work.*

Proof. Follows from Lemma 1 and Lemma 3. □

Note how this result contrasts with the bounds on the bandwidth of a tree graph: The bandwidth of a balanced binary tree is $\Omega(n/\log n)$, whereas, we can construct a decomposition into $O(\log n)$ matrices of bandwidth $O(1)$.

5.5 Lower Bounds

The linear dependence on the maximum degree Δ is necessary for any linear arrangement of the graph families listed in Table 1. We prove the lower bound for trees first, which then implies the other lower bounds:

Lemma 4. *For every $\Delta > 3$, there are trees with a minimum linear arrangement of cost $\Omega(n\Delta)$.*

Proof. First, consider a star graph of $\Delta - 1$ nodes. Any linear arrangement costs at least $\Omega(\Delta^2)$, as, at least a quarter of the nodes are at least $\Delta/4$ away from the central node (no matter where it is placed). Moreover, observe that inserting additional nodes into the graph can only increase the cost incurred by the edges in the star.

Now, consider a complete $(\Delta - 1)$-ary tree. The parents of the leaf nodes together with their descendants constitute $\Omega(n/\Delta)$ disjoint star graphs with degree $\Delta - 1$. Their layout costs $\Omega(\Delta^2)$ each, which implies the result. □

The $\Omega(n\Delta)$ lower bound on the cost of a linear arrangement applies to all families in Table 1 and is tight for trees, as shown in Section 5.4. The linear dependence on the maximum degree is undesirable, as many sparse real-world graphs exhibit a large maximum degree [2]. Next, we show the arrow decomposition overcomes this dependence on the maximum degree by pruning the highest-degree vertices.

5.6 Pruning in Power Law Graphs

Many real-world graphs, such as the web graph, social networks, and protein interaction networks, exhibit a power law degree distribution [2]. This means that while the average degree is small, the maximum degree can be a significant fraction of the number of vertices. On these graphs, the first step of LA-Decompose (pruning the highest degree vertices) provides a *polynomial* improvement in the arrow width. We proceed to bound the improvement analytically as a function of the power law.

There are various probability distributions that generate a power law [5, 7, 31, 51]. To model the vertex degrees, it is appropriate to choose a discrete distribution that is bounded to the interval of the number of vertices. Hence, we model the degree distribution of a vertex as a *discrete truncated Zipf distribution* [7], truncated between 1 and n with shape parameter α. Note that for simplicity of notation, we are considering $n+1$ vertices here giving degrees between 1 and n. We exclude the possibility of singleton vertices, as they do not contribute to the arrow width.

The probability mass function $p(x)$ of a discrete truncated Zipf distribution is given by

$$p(x) = \frac{x^{-\alpha}}{\sum_{j=1}^{n} j^{-\alpha}} \quad . \tag{2}$$

The term in the denominator is the generalized harmonic number $H_{n,\alpha}$. Note that as n goes to infinity, the generalized harmonic numbers approach the Riemann zeta function $\zeta(\alpha) = \sum_{j=1}^{\infty} j^{-\alpha}$. For an integer $x \geq 0$, the survival probability $S(x)$ is given by

$$S(x) = \frac{H_{n,\alpha} - H_{x,\alpha}}{H_{n,\alpha}} \quad . \tag{3}$$

The expected number of vertices with degrees larger than some given x is at most $nS(x)$. This tells us how many vertices we need to prune (in expectation) to be left with a graph with maximum degree x. To derive a bound on this expectation $nS(x)$, we derive a closed-form approximation to the survival function:

Theorem 1. *For all x larger than some constant, the survival function $S(x)$ of the truncated Zipf distribution with shape $\alpha > 1$ truncated between 1 and n is bounded by $S(x) \leq \frac{x^{1-\alpha}}{(\alpha-1)\zeta(\alpha)}$.*

Proof. We lower bound the cumulative distribution function $F(x) = 1 - S(x) = \frac{H_{x,\alpha}}{H_{n,\alpha}}$, which gives us an upper bound on $S(x)$. The main technical challenge is to obtain a suitable closed-form approximation to the generalized harmonic numbers. We employ the Euler-Maclaurian summation formula [25] to bound $\zeta(\alpha) - H_{n,\alpha}$, which implies that for any constant $\alpha > 1$

$$H_{n,\alpha} = \zeta(\alpha) + \frac{n^{1-\alpha}}{1-\alpha} + \frac{n^{1-\alpha}}{2n} - \frac{\alpha n^{1-\alpha}}{12n^2} + O\left(\frac{\alpha n^{1-\alpha}}{n^3}\right) \quad .$$

For large enough x and $x+1 \geq \alpha > 1$, the first two terms dominate:

$$H_{x,\alpha} \geq \zeta(\alpha) + \frac{1}{1-\alpha} x^{1-\alpha} \quad ,$$
$$H_{x,\alpha} \leq \zeta(\alpha) + \frac{1}{2(1-\alpha)} x^{1-\alpha} \quad .$$

Using these inequalities we can proceed:

$$\begin{aligned} F(x) &\geq \frac{\zeta(\alpha) + \frac{1}{1-\alpha} x^{1-\alpha}}{\zeta(\alpha) + \frac{1}{2(1-\alpha)} n^{1-\alpha}} \\ &= \frac{2(\alpha-1)n^{\alpha-1}\zeta(\alpha) - 2\frac{n^{\alpha-1}}{x^{\alpha-1}}}{2(\alpha-1)n^{\alpha-1}\zeta(\alpha) - 1} \\ &\geq 1 - \frac{x^{1-\alpha}}{(\alpha-1)\zeta(\alpha)} \quad , \end{aligned}$$

which implies the result. □

Note that this implies that the survival function itself takes on the shape of a power law. The larger α, the quicker the survival function diminishes. We are now ready to bound the number of high-degree vertices in a power law graph.

Lemma 5. *Consider a graph G whose degree distribution follows a truncated Zipf distribution with shape parameter $\alpha > 1$. For any $b \geq \Omega(1)$ and $\Delta_0 \geq \Omega(1)$, the probability that G has more than b vertices of degree larger or equal to Δ_0 is at most $\frac{n\Delta_0^{1-\alpha}}{b(\alpha-1)\zeta(\alpha)}$.*

Proof. The expected number of vertices with degrees larger than Δ_0 is at most $nS(\Delta_0)$. The result follows from Theorem 1 and Markvov's inequality. □

Let us see what this implies for the question of pruning high-degree vertices. If we set $\Delta_0 = n^{\delta}$ for some constant $\delta > 0$, we get that after pruning the $b \in \omega(n^{(1-\alpha)\delta+1})$ vertices of largest degree, the maximum degree of the remaining subgraph is at most Δ_0 with probability $1 - o(1)$. As our bounds on the cost of a linear arrangement depend linearly on the maximum degree of the subgraph that remains after pruning the high-degree graphs, we would like to balance the number of pruned vertices with the remaining maximum degree. The parameter δ that achieves this balance is $\delta = \frac{1}{\alpha}$.

We conclude with the implication of this result for the arrow decomposition of trees and note that we can derive similar statements for the other considered graph families:

Corollary 2. *Consider a tree whose degree distribution follows a truncated Zipf distribution with shape $\alpha > 1$. LA-Decompose with parameter $b = \omega(n^{\frac{1}{\alpha}})$ produces an x-compacting xb-arrow matrix decomposition with probability $1 - o(1)$.*

Proof. Follows from Lemma 5 and Corollary 1. □

Observe that this bound is now independent of the maximum degree in the original graph, which would have been $\Omega(n)$ in expectation. Hence, pruning the high-degree vertices provides a polynomial improvement in the arrow width of power law graphs.

6 Data Movement Analysis

Sparse matrix multiplication is a typical memory-bound operation when the dense matrix is tall and skinny, as the number of arithmetic operations is of a similar order of magnitude to the number of memory accesses. Hence, minimizing data movement is paramount to achieving the best performance and scalability. Note that for sparse datasets, the size of the feature matrix $\mathbf{X}$ dominates the storage, i.e., $m \ll nk$. Our algorithm falls into the class of $\mathbf{A}$-stationary algorithms, where the sparse matrix remains local and only the dense feature matrix and the result of the SpMM are communicated. We show that at the cost of a slightly increased latency, a c-compacting arrow decomposition enables a $\Theta(\sqrt{p})$ reduction in bandwidth requirements compared to a direct 1.5D decomposition and a $\Theta(\sqrt{p})$ storage improvement in the setting where the feature matrix dominates the storage.

6.1 Data Movement

Multiplying with an Arrow Matrix. We now analyze the communication cost incurred by our approach from Section 3.1 in the $\alpha - \beta$ model of computation. We focus first on the multiplication of an arrow matrix $\mathbf{B}$ with a tall-skinny matrix $\mathbf{X}$.

Lemma 6. *Consider a matrix $\mathbf{B} \in R^{n \times n}$ with arrow-width b and a matrix $\mathbf{X} \in R^{n \times k}$. If $p = \lceil n/b \rceil$, computing $Y = \mathbf{BX}$ has a communication cost of $O(\alpha \log p + \beta\ bk \log p)$.*

Proof. Recall that we have $\lceil \frac{n}{b} \rceil$ row tiles $\mathbf{B_{0,j}}$, $\lceil \frac{n}{b} \rceil$ column tiles $\mathbf{B_{i,0}}$ and $\lceil \frac{n}{b} \rceil$ diagonal tiles $\mathbf{B_{i,i}}$. Let $\mathbf{X}$ also be split row-wise into $\lceil \frac{n}{b} \rceil$ blocks of size $b \times k$. We distribute the calculation of $\mathbf{BX}$ as follows: For each row tile $\mathbf{B_{0,j}}$, there is a processor responsible for calculating $\mathbf{B_{0,j}X_j}$. The intermediate results are reduced and summed at one node. For each column tile $\mathbf{B_{i,0}}$ with $i > 0$, we have a processor responsible for calculating $\mathbf{B_{i,0}X_0} + \mathbf{B_{i,i}X_i}$. Due to the arrow shape, we only have two non-zero tiles per row when $i > 1$. Hence, we can do the calculation of the entire row on one processor.

Overall, we have $\lceil \frac{2n}{b} \rceil - 1$ computation tasks which we assign to our p processors. We assume that the tiles of $\mathbf{B}$ are already correctly distributed as they remain fixed throughout the iterations. Note that half of the computation tasks will require a copy of $\mathbf{X_0}$, i.e. we will need one broadcast of $\mathbf{X_0}$ to half of the processors which incurs a communication cost of $O(\alpha \log p + \beta\ bk \log p)$. The reduce-operation for the row tiles incurs the same cost (the reduce-operation also involves $\frac{p}{2}$ processors since the row tiles make up half of the computation tasks). Lastly, we will need to send the diagonal blocks of $\mathbf{X}$ to the right processors using pairwise communication. Since each processor only needs a single block from $\mathbf{X}$, this only incurs a cost of $O(\alpha + \beta\ bk)$. □

Multiplying with an Arrow Decomposition. Next, we analyze the communication cost of combining the intermediate products $\mathbf{B}_i(\mathbf{P}_{\pi_i}^\top \mathbf{X})$ of our matrix decomposition to finally arrive at $\mathbf{Y} = \mathbf{AX}$. To improve the efficiency of multiplying repeatedly with the same matrix $\mathbf{A}$, we leave the rows of $\mathbf{Y}$ permuted in the order of the first matrix in the decomposition. In the end, we might need to permute back to the original order of rows depending on the application. If the result is required in the original order of rows, the communication cost of this permutation is fully amortized after at most $\log^2 p$ iterations of multiplying with $\mathbf{A}$. The key insight is that when the number of nonzeros decreases quickly, we can implement the permutations for the aggregation more efficiently than doing a naive all-to-all.

Theorem 2. *For a matrix $\mathbf{A} \in R^{n \times n}$ with an x-compacting b-arrow decomposition and a feature matrix $\mathbf{X} \in R^{n \times k}$, if $x \geq \Omega(\log^2 p)$ and $p = \Theta(\frac{n}{b})$, computing $\mathbf{P}_{\pi_0}^\top \mathbf{Y} = \mathbf{P}_{\pi_0}^\top \mathbf{AX}$ has a communication cost of $O(\alpha \log^2 p + \beta \frac{nk}{p})$.*

Proof. Assuming we calculated $\mathbf{Y}_i = \mathbf{B}_i(\mathbf{P}_{\pi_i}^\top \mathbf{X})$ for each $\mathbf{B_i}$ matrix in our decomposition using Lemma 6, it remains to aggregate the results. We need to sum the $\mathbf{Y_i}$ matrices, however, each of them has its rows permuted differently. Let $\mathbf{Y}_i$ and $\mathbf{Y}_{i+1}$ be any two of these matrices that we want to sum and let P_i be the set of processors among which the blocks of $\mathbf{Y}_i$ are distributed.

If we were to send all the rows of $\mathbf{Y_{i+1}}$ to their corresponding processor in P_i naively, in the worst case we would have to perform one scatter operation for each processor in P_{i+1} to all the processors in P_i. To alleviate this, we will first sort the rows of $\mathbf{Y_{i+1}}$ by their destination processor in P_i. Then, for any $p_j^{i+1} \in P_{i+1}$, it holds that if it stores rows of $\mathbf{Y_{i+1}}$ that need to be sent to processors $p_t^i ... p_{t+k}^i$ then for any processor p_l^{i+1} with $l > j$, it holds that it only stores rows of $\mathbf{Y_{i+1}}$ that need to be sent to $p_m^i ... p_{m+k'}^i$ where $m \geq t + k$.

Note that we can determine the destination processor for each row in advance since the permutation matrices are fixed in the pre-processing. We can also pre-determine the destination processor range of each processor in P_{i+1} this way. After sorting, we can schedule the scatter operations much more efficiently in parallel: Observe that each processor logically receives a message from at most two scatter operations and that these come from neighboring processors in P_{i+1}. Hence, we can perform the scatter operations in two phases involving first all evenly-indexed processors in P_{i+1} and then oddly-indexed processors in P_{i+1}. This scattering then incurs a communication cost of $O(\alpha \log p + \beta\ \frac{nk}{c} \log p)$.

As for the sorting itself, we can employ a sorting network and place each processor of P_{i+1} on a wire. When two processors p_k^{i+1} and p_l^{i+1} with $k < l$ are connected, p_k^{i+1} will send p_l^{i+1} all its rows that are above its range and p_l^{i+1} will send p_k^{i+1} all its rows that are below. For a bitonic sorting network with depth $O(\log^2 p)$, this leads to a communication cost of $O(\alpha \log^2 p + \beta\ \frac{nk}{c} \log^2 p)$. Sorting networks with a smaller depth exist but are unwieldy in practice.

Overall, we can thus perform the sorting and the subsequent scattering with a communication cost of $O(\alpha \log^2 p + \beta\ \frac{nk}{c} \log^2 p)$. Assuming we have a c-compacting decomposition, recall that it holds that the total number of non-zeros of Y_{i+1} is at most $\frac{1}{c}$ times that of Y_i. Thus, if we perform the summation of all l parts in decreasing order, the total communication can be upper-bounded by the summation of Y_0 and Y_1, i.e., $O(\alpha \log^2 p + \beta\ \frac{nk}{c} \log^2 p)$. For $c \geq \Omega(\log^2 p)$, we arrive at a communication cost of $O(\alpha \log^2 p + \beta \frac{nk}{p})$ for one matrix iteration, as desired.

To prepare the next iteration, the accumulated result is distributed in the reverse communication pattern (replacing scatters with gather). This concludes the calculation of $\mathbf{P}_{\pi_0}^\top \mathbf{AX}$. □

This result shows that given an adequate arrow decomposition, we obtain a communication cost that improves on a fully replicated 1.5D decomposition by a factor $\sqrt{p}$ in terms of bandwidth at the cost of a $\log p$ factor in terms of latency. Compared to a 1D decomposition, the latency cost is a factor $\Omega(\frac{p}{\log p})$ smaller. Compared to an **A**-stationary 2D decomposition, the bandwidth cost is a factor $\Theta(\sqrt{p})$ smaller and the latency cost is a factor $\Theta\left(\frac{\sqrt{p}}{\log n}\right)$ smaller.

6.2 Storage Requirements

We propose to store the matrix **A** in a sparse format, such as compressed sparse row (CSR), and store the dense matrices in row-major. Because an arrow matrix has fewer blocks compared to a matrix that has been 1.5D decomposed directly, we can afford to have smaller blocks with the same number of processors. This removes the main downside of a 1.5D decomposition with full replication, namely its high storage requirements.

Lemma 7. *Consider a matrix* $\mathbf{A} \in R^{n\times n}$ *with an* x*-compacting* b*-arrow decomposition and* $\mathbf{X} \in R^{n\times k}$*. If* $x \geq 1+\epsilon$ *for a constant* ϵ *and the blocks of* **A** *are stored in CSR, the total storage cost is* $m + O(nk)$*.*

Proof. The number of nonzero rows in the matrices of the decomposition decreases geometrically because the decomposition is x-compacting. Hence, the cost to store **X** is $O(nk)$ and the cost to store row and index pointers for **A** is $O(n)$. Since each edge occurs in exactly one matrix of the decomposition, the value arrays take up m space. □

The storage matches that of a 2D decomposition. Compared to a 1.5D decomposition with replication factor c, the storage used by the dense matrices is a factor $\Theta(c)$ smaller. For full replication, this constitutes a factor $\Theta(\sqrt{p})$ saving. In conclusion, the arrow matrix decomposition shares the favorable space requirements of 1D and 2D decompositions while improving on the bandwidth cost and memory requirements of a fully replicated 1.5D decomposition.

7 Evaluation

We benchmark the scalability of sparse matrix multiplication with our decomposition compared to a 1.5D decomposition and a 1D hypergraph partitioning decomposition.

7.1 Setup

System. We ran our SpMM experiments on the Piz Daint supercomputer on the Cray XC50 nodes with Aries routing and a dragonfly network topology. Each node has a 12-core HT-enabled Intel Xeon E5-2690 v3 CPU with 64GB of RAM and one NVIDIA Tesla P100 GPU with 16GB of memory. For decomposing the graphs, we ran our algorithm on single nodes with 4 Intel(R) E7-4830 v4 CPUs and 2TB of memory, and 2 AMD EPYC 7742 CPUs with 512GB of memory.

Table 2. Summary of the datasets' density properties.

Dataset	Vertices n	$\frac{\mathrm{nnz}(A)}{n}$	Max. Degree Δ
MAWI 226M	226,196,185	2.12	210,795,477
MAWI 69M	68,863,315	2.08	63,040,326
GenBank 214M	214,005,017	2.17	8
GenBank 68M	67,716,231	2.05	35
WebBase 118M	118,142,155	8.63	816,127
OSM Europe	50,912,018	2.12	13
GAP-twitter 62M	61,578,415	23.85	770,155
sk-2005 51M	50,636,154	38.50	8,563,808

Datasets. We evaluate our approach on sparse graph datasets with 18-226 million rows and up to a 1.9 billion nonzeros from the SuiteSparse Matrix Collection [18]. See Table 2 for a summary of the dataset's characteristics. We considered all matrices with at least 50M rows and fewer than 100 nonzeros per row on average. The denser graphs cause out-of-memory issues and timeouts with both the baselines and our approach. For the feature matrices, we use 128, 64, and 32 columns.

Implementation. We implement all SpMM algorithms as a Python module, using *numpy* v1.24.2 [27], *scipy* v1.10.1 [49], *cupy* v11.6 [36], and *mpi4py* v3.1.4 [17]. The single-block intra-node GPU SpMM operations are implemented with the *CSR*-times-dense matrix multiplication (CSRMM) kernels found in the NVIDIA cuSPARSE (v11.0). Our experiments use Cray-MPICH v7.7.18. The code is available at: https://github.com/spcl/arrow-matrix

Our approach utilizes the linear arrangement framework (Section 5.1) with pruning (Section 5.6). We construct the linear arrangements using the random spanning MSTs algorithm (Section 5.3). For each tree, we employ the algorithm from Section 5.4. The decomposition uses Julia v.1.9.3. Our implementation may leave a few ranks unused when the block size does not evenly divide the matrix size.

1.5D Baseline. We compare our approach based on the 1.5D decomposition of an arrow matrix decomposition against the direct 1D and 1.5D decomposition schemes. We use the same libraries and kernels to ensure a comparison that focuses on the merits of the decomposition schemes. Similarly as Tripathy at et al. [47] we include a parameter c that interpolates between the $1D$ decomposition ($c = 1$) and the single-round 1.5D decomposition ($c = \sqrt{p}$). As shown in Figure 4, a larger c leads to lower runtimes, as expected from the theory. Hence, we use $c = \lfloor\sqrt{p}\rfloor$ in our experiments, resulting in one or two computational rounds. For larger feature sizes, the whole data sometimes do not fit in GPU memory. In this case, we further tile the single-round computation into smaller blocks. This approach is significantly faster than lowering c since host-to-device memory transfers are faster than the network.

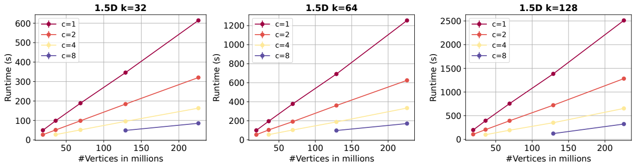


Figure 4. Weak scaling of the 1D / 1.5D baseline for varying replication factors c on the MAWI datasets.

Hypergraph Partitioning Baseline. Additionally, we compare our implementation against a 1D hypergraph partitioning scheme (HP-1D). We adapt the PETSc-style variant from previous work on SpMV [29] to the SpMM setting. The matrix is permuted according to a hypergraph partitioning. The hypergraph contains a vertex for each row i and a net for each column j that connects all vertices i for which the matrix has a nonzero entry in row i and column j. To partition the hypergraphs, we use HYPE [34]. After permuting, the matrices are split row-wise in 1D. The computation consists of two parts; a local SpMM that requires no communication and a non-local SpMM for which features of other processors are needed. The local SpMM can overlap with the message exchange, hiding computational costs. We implement this overlap using MPI nonblocking send and receive.

Measurements. We run each SpMM for at least 7 iterations and drop the first iteration, as it includes GPU and library initialization costs. We report the mean over the iterations of the maximum runtime of any participating rank. Additionally, we show the minimum and maximum over the iterations when they deviate more than 5% from the mean.

7.2 Decomposition Results

We executed our random forests algorithm on each dataset, varying the arrow width as $b \in 0.5 \times 10^6, 1 \times 10^6, \ldots, 5 \times 10^6$. This approach yielded, at most, 4 matrices in the decomposition for all datasets. Additionally, the second matrix in the decomposition contained between a few hundred and 25 million nonzero rows, constituting less than $0.1\% - 13\%$ of the rows. This aligns with our assumptions in Theorem 2, affirming our algorithm's generation of highly compact decompositions for these sparse datasets.

Refer to Figure 1 for a depiction of the first matrix in the decomposition (with analogous outcomes for all MAWI and GenBank matrices). Notably, the datasets exhibit distinct characteristics: In the MAWI data, most nonzeros cluster in the 'pruned' part near the matrix's top and left corners. Conversely, for the GenBank and OSM Europe data, the majority of nonzeros appear in the diagonal band. The Webbase and GAP-twitter datasets showcase a notable number of nonzeros in the pruned segment, yet they are less skewed than the MAWI datasets. These patterns correspond with the datasets' maximum degrees. Specifically, the MAWI datasets feature a nearly equal maximum degree and vertex count due to the prevalence of large star subgraphs. In contrast, the Webbase and GAP-twitter datasets have a maximum degree of $8 - 13\%$ of the vertex count. The GenBank k-mer and OSM Europe datasets, on the other hand, display a maximum degree at most 20 times the average. Our approach handles these diverse matrix behaviors robustly.

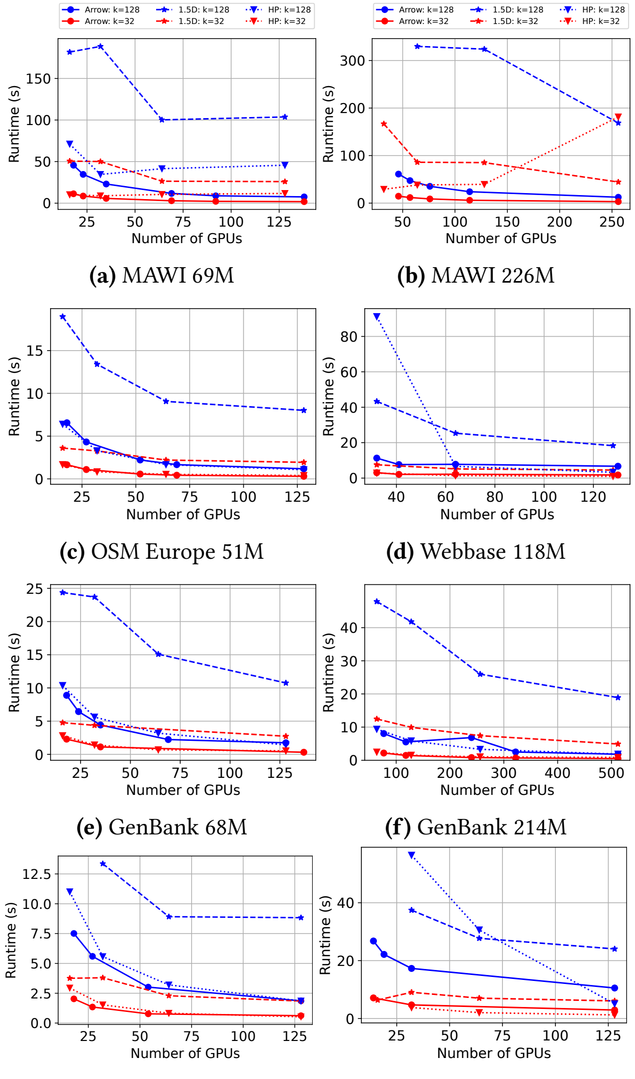
Comparison with 1.5D. We contrast the count of nonzero blocks in our arrow decomposition matrices with those in a 1.5D decomposition employing equally sized blocks. As matrix rows increase, our method uses notably fewer nonzero blocks. For the two largest datasets, our decomposition results in 15-20 times fewer nonzero blocks at $b = 5 \times 10^6$, and over 100 times fewer at $b = 10^6$. Our SpMM experiments demonstrate this improves the scalability of our approach.

Comparison with Hypergraph partitioning. HYPE was able to effectively partition the graphs with low maximum degree. On the GAP-twitter and sk-2005 graphs, the results are mixed. Especially for smaller number of partitions (≤ 32) we observe a high partition cost. We attribute this to the power-law distribution of those graphs, which makes them challenging to partition in a balanced way. On the MAWI series of graphs, the partitioning is completely ineffective as it leads to one partition being connected to an overwhelming majority of the other vertices. This is because fundamentally, the partitioning cost is lower bounded by the maximum degree, which we overcome using pruning.

7.3 SpMM Experiments

Strong Scaling. Figure 5 summarizes our strong scaling evaluation for varying GPU counts and feature matrix column sizes ($k \in 32, 128$). Our approach outperforms the 1.5D baseline by significant margins in all instances, except for GAP-twitter on 16 ranks and k=32. In the other cases, the speedup is between 1.7x and 14x. For the MAWI graphs, the hypergraph partitioning baseline does not scale and is up to 58 times slower than our approach. On sk-2005 and GAP-twitter our approach is up to 1.4 and 2 times faster, respectively. On the other graphs, our approaches has a similar runtime with HP-1D. Generally, the more features, the greater the runtime improvement over both baselines. The more skewed the degree distribution is, the larger our improvement over the hypergraph partitioning baseline.

We noticed significant load imbalance in GPU kernels on the MAWI graphs, reaching up to 8x. This imbalance renders both baselines less effective in reducing SpMM kernel runtime with increasing ranks. Moreover, when $c^2 \leq p$, more communication rounds are needed for the 1.5D baseline. Hence, sometimes increasing the number of ranks increases the runtime on MAWI. We observe that the MAWI datasets contain very large star subgraphs, leading to a high maximum degree. Moreover, they cause load imbalance issues on the GPU, which we think responsible for the comparatively long execution times on the MAWI instances.



(a) MAWI 69M

(b) MAWI 226M

(c) OSM Europe 51M

(d) Webbase 118M

(e) GenBank 68M

(f) GenBank 214M

(g) sk-2005 51M

(h) GAP-twitter 62M

Figure 5. Strong scaling results for varying features sizes.

Weak Scaling. We applied our decomposition to all MAWI datasets, maintaining a consistent arrow width of 3 million for a constant computational load per rank. As depicted in Figure 6, as the dataset size grows from 19 million to 226 million vertices, the runtime only increases marginally by $2.4 - 6.2\%$. In contrast, the 1.5D baseline decelerates by a factor of $3 - 3.18$ when going from the smallest to the largest dataset, along with poorer absolute runtimes. The Hypergraph partitioning baseline's runtime grows near-linearly with the number of rows. This is because of the inherent limitations of a 1D decomposition on matrices with highly skewed degree distributions.

8 Conclusion

In conclusion, we introduce a novel matrix decomposition, investigating its properties both theoretically and experimentally. The resulting matrices exhibit a distinctive 'arrow' structure that facilitates distributed computation. We demonstrate its efficacy in scenarios of extreme sparsity, where graphs possess an average of 2 to 38 nonzeros per row, even in cases of severely skewed degree distribution.

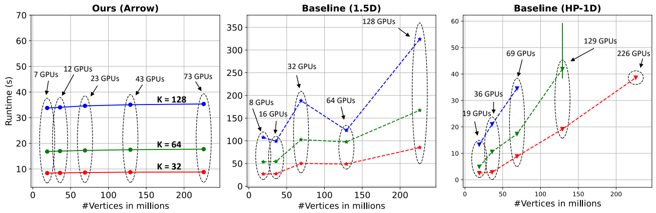


Figure 6. Weak scaling on the MAWI datasets. Missing datapoints represent out-of-memory events.

Building upon this matrix decomposition, we propose a distributed approach for sparse times tall-skinny-dense matrix multiplication. This method, grounded in a 1.5D decomposition of the arrow matrices, addresses the memory constraints inherent in direct 1D and 1.5D decompositions. Remarkably, it has low communication costs, all the while avoiding the need to partition the computation into numerous steps. We substantiate this claim through both theoretical analysis and empirical evaluation.

Our comparison against the 1.5D decomposition and a 1D hypergraph partitioning demonstrate our method's scalability and efficiency. Strong scaling experiments highlighted its superiority in handling larger matrices and features, while weak scaling tests showcased its stable runtime with growing datasets.

Acknowledgments

This project received funding from the European Research Council under the European Union's Horizon 2020 programme (Project MAELSTROM, No. 95513) and the Horizon Europe programme (Project GLACIATION, No. 101070141). This work was also supported by the PASC DaCeMI project, as well as UrbanTwin, a joint initiative of the Board of the Swiss Federal Institutes of Technology, in the strategic areas of Energy, Climate and Environmental Sustainability and Engagement and Dialogue with Society. A.N.Z. and T.B.N. are funded by the Swiss National Science Foundation ("QuaTrEx" Project No. 20935 and Ambizione Project No. 185778, respectively). The authors wish to acknowledge the Swiss National Supercomputing Center (CSCS) for providing computing infrastructure.

References

[1] Acer, S., Selvitopi, R. O., and Aykanat, C. Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems. *Parallel Comput. 59* (2016), 71–96.

[2] Albert, R., and Barabási, A.-L. Statistical mechanics of complex networks. *Rev. Mod. Phys. 74* (Jan 2002), 47–97.

[3] Albert, R., Jeong, H., and Barabási, A. Diameter of the world-wide web. 130–131.

[4] Alemany-Puig, L., Esteban, J. L., and Ferrer-i-Cancho, R. Minimum

projective linearizations of trees in linear time. *Inf. Process. Lett. 174* (2022), 106204.

[5] Balakrishnan, N., and Nevzorov, V. B. *A primer on statistical distributions.* John Wiley & Sons, 2004.

[6] Ballard, G., Druinsky, A., Knight, N., and Schwartz, O. Hypergraph partitioning for sparse matrix-matrix multiplication. *ACM Trans. Parallel Comput. 3*, 3 (2016), 18:1–18:34.

[7] Boamah-Addo, K., Kozubowski, T., and Panorska, A. A discrete truncated zipf distribution. *Statistica Neerlandica* (09 2022).

[8] Boman, E. G., Devine, K. D., and Rajamanickam, S. Scalable matrix computations on large scale-free graphs using 2d graph partitioning. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA - November 17 - 21, 2013* (2013), W. Gropp and S. Matsuoka, Eds., ACM, pp. 50:1–50:12.

[9] Borstnik, U., VandeVondele, J., Weber, V., and Hutter, J. Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Comput. 40*, 5-6 (2014), 47–58.

[10] Böttcher, J., Pruessmann, K. P., Taraz, A., and Würfl, A. Bandwidth, expansion, treewidth, separators and universality for bounded-degree graphs. *Eur. J. Comb. 31*, 5 (2010), 1217–1227.

[11] Buono, D., Petrini, F., Checconi, F., Liu, X., Que, X., Long, C., and Tuan, T. Optimizing sparse matrix-vector multiplication for large-scale data analytics. In *Proceedings of the 2016 International Conference on Supercomputing, ICS 2016, Istanbul, Turkey, June 1-3, 2016* (2016), O. Ozturk, K. Ebcioglu, M. T. Kandemir, and O. Mutlu, Eds., ACM, pp. 37:1–37:12.

[12] Çatalyürek, Ü. V., and Aykanat, C. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distributed Syst. 10*, 7 (1999), 673–693.

[13] Chan, E., Heimlich, M., Purkayastha, A., and van de Geijn, R. A. Collective communication: theory, practice, and experience. *Concurr. Comput. Pract. Exp. 19*, 13 (2007), 1749–1783.

[14] Charikar, M., Hajiaghayi, M. T., Karloff, H. J., and Rao, S. $l_2{}^2$ spreading metrics for vertex ordering problems. *Algorithmica 56*, 4 (2010), 577–604.

[15] Chinn, P. Z., Chvatalova, J., Dewdney, A. K., and Gibbs, N. E. The bandwidth problem for graphs and matrices - a survey. *J. Graph Theory 6*, 3 (1982), 223–254.

[16] Cuthill, E. H., and McKee, J. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th national conference, ACM 1969, USA, 1969* (1969), S. L. Pollack, T. R. Dines, W. C. Sangren, N. R. Nielsen, W. G. Gerkin, A. E. Corduan, L. Nowak, J. L. Mueller, J. H. III, P. S. T. Yuen, J. Stein, and M. M. Mueller, Eds., ACM, pp. 157–172.

[17] Dalcín, L., Paz, R., and Storti, M. Mpi for python. *Journal of Parallel and Distributed Computing 65*, 9 (2005), 1108–1115.

[18] Davis, T. A., and Hu, Y. The university of florida sparse matrix collection. *ACM Trans. Math. Softw. 38*, 1 (dec 2011).

[19] Devine, K. D., Boman, E. G., Heaphy, R. T., Bisseling, R. H., and Çatalyürek, Ü. V. Parallel hypergraph partitioning for scientific computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece* (2006), IEEE.

[20] Eikel, M., Scheideler, C., and Setzer, A. Minimum linear arrangement of series-parallel graphs. In *Approximation and Online Algorithms - 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers* (2014), E. Bampis and O. Svensson, Eds., vol. 8952 of *Lecture Notes in Computer Science*, Springer, pp. 168–180.

[21] Feige, U. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci. 60*, 3 (2000), 510–539.

[22] Feige, U. Coping with the np-hardness of the graph bandwidth problem. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings* (2000), M. M. Halldórsson, Ed., vol. 1851 of *Lecture Notes in Computer Science*, Springer, pp. 10–19.

[23] Geng, T., Wu, C., Zhang, Y., Tan, C., Xie, C., You, H., Herbordt, M. C., Lin, Y., and Li, A. I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021* (2021), ACM, pp. 1051–1063.

[24] Golub, G. H., and Van Loan, C. F. *Matrix computations.* JHU press, 2013.

[25] Graham, R. L., Knuth, D. E., and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*, second ed. Addison-Wesley, Reading, MA, 1994.

[26] Gravvanis, G. A. An approximate inverse matrix technique for arrowhead matrices. *Int. J. Comput. Math. 70*, 1 (1998), 35–45.

[27] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[28] Kawarabayashi, K., and Reed, B. A. A separator theorem in minor-closed classes. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA* (2010), IEEE Computer Society, pp. 153–162.

[29] Kaya, K., Uçar, B., and Çatalyürek, Ü. V. Analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication. In *Parallel Processing and Applied Mathematics - 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part II* (2013), R. Wyrzykowski, J. J. Dongarra, K. Karczewski, and J. Wasniewski, Eds., vol. 8385 of *Lecture Notes in Computer Science*, Springer, pp. 174–184.

[30] Koanantakool, P., Azad, A., Buluç, A., Morozov, D., Oh, S., Oliker, L., and Yelick, K. A. Communication-avoiding parallel sparse-dense matrix-matrix multiplication. In *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016* (2016), IEEE Computer Society, pp. 842–853.

[31] Kozubowski, T. J., Panorska, A. K., and Forister, M. L. A discrete truncated pareto distribution. *Statistical Methodology 26* (2015), 135–150.

[32] Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards 45*, 4 (Oct. 1950).

[33] Lipton, R. J., and Tarjan, R. E. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics 36*, 2 (1979), 177–189.

[34] Mayer, C., Mayer, R., Bhowmik, S., Epple, L., and Rothermel, K. HYPE: massive hypergraph partitioning with neighborhood expansion. In *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018* (2018), N. Abe, H. Liu, C. Pu, X. Hu, N. K. Ahmed, M. Qiao, Y. Song, D. Kossmann, B. Liu, K. Lee, J. Tang, J. He, and J. S. Saltz, Eds., IEEE, pp. 458–467.

[35] Muradian, D. The bandwidth minimization problem for cyclic caterpillars with hair length 1 is np-complete. *Theor. Comput. Sci. 307*, 3 (2003), 567–572.

[36] Okuta, R., Unno, Y., Nishino, D., Hido, S., and Loomis, C. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)* (2017).

[37] Page, B. A., and Kogge, P. M. Scalability of hybrid spmv with hypergraph partitioning and vertex delegation for communication avoidance. In *International Conference on High Performance Computing & Simulation (HPCS 2020)* (2021).

[38] PAIGE, C. C. Computational Variants of the Lanczos Method for the

Eigenproblem. *IMA Journal of Applied Mathematics 10*, 3 (12 1972), 373–381.

[39] Papadimitriou, C. H. The np-completeness of the bandwidth minimization problem. *Computing 16*, 3 (1976), 263–270.

[40] Plotkin, S. A., Rao, S., and Smith, W. D. Shallow excluded minors and improved graph decompositions. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA* (1994), D. D. Sleator, Ed., ACM/SIAM, pp. 462–470.

[41] Rao, S., and Richa, A. W. New approximation techniques for some linear ordering problems. *SIAM J. Comput. 34*, 2 (2004), 388–404.

[42] Raoufi, P., Rostami, H., and Bagherinezhad, H. An optimal time algorithm for minimum linear arrangement of chord graphs. *Inf. Sci. 238* (2013), 212–220.

[43] Robertson, N., and Seymour, P. Graph minors. vi. disjoint paths across a disc. *Journal of Combinatorial Theory, Series B 41*, 1 (1986), 115–138.

[44] Schatz, M. D., van de Geijn, R. A., and Poulson, J. Parallel matrix multiplication: A systematic journey. *SIAM J. Sci. Comput. 38*, 6 (2016).

[45] Selvitopi, O., Brock, B., Nisa, I., Tripathy, A., Yelick, K. A., and Buluç, A. Distributed-memory parallel algorithms for sparse times tall-skinny-dense matrix multiplication. In *ICS '21: 2021 International Conference on Supercomputing, Virtual Event, USA, June 14-17, 2021* (2021), H. Zhou, J. Moreira, F. Mueller, and Y. Etsion, Eds., ACM, pp. 431–442.

[46] Sun, J., Vandierendonck, H., and Nikolopoulos, D. S. Graphgrind: addressing load imbalance of graph partitioning. In *Proceedings of the International Conference on Supercomputing, ICS 2017, Chicago, IL, USA, June 14-16, 2017* (2017), W. D. Gropp, P. Beckman, Z. Li, and F. J. Cazorla, Eds., ACM, pp. 16:1–16:10.

[47] Tripathy, A., Yelick, K. A., and Buluç, A. Reducing communication in graph neural network training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020* (2020), C. Cuicchi, I. Qualters, and W. T. Kramer, Eds., IEEE/ACM, p. 70.

[48] Turner, J. S. On the probable performance of heuristics for bandwidth minimization. *SIAM J. Comput. 15*, 2 (1986), 561–580.

[49] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272.

[50] Zheng, D., Mhembere, D., Lyzinski, V., Vogelstein, J. T., Priebe, C. E., and Burns, R. C. Semi-external memory sparse matrix multiplication for billion-node graphs. *IEEE Trans. Parallel Distributed Syst. 28*, 5 (2017), 1470–1483.

[51] Zipf, G. K. *Human behavior and the principle of least effort: An introduction to human ecology.* Addison-Wesley, Cambridge, Massachusetts, 1949.