

Preserving Sparsity and Privacy in Straggler-Resilient Distributed Matrix Computations

Anindya Bijoy Das*, Aditya Ramamoorthy†, David J. Love*, Christopher G. Brinton*

*School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA

†Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010 USA

das207@purdue.edu, adityar@iastate.edu, djlove@purdue.edu, cgb@purdue.edu

Abstract—Existing approaches to distributed matrix computations involve allocating coded combinations of submatrices to worker nodes, to build resilience to stragglers and/or enhance privacy. In this study, we consider the challenge of preserving input sparsity in such approaches to retain the associated computational efficiency enhancements. First, we find a lower bound on the weight of coding, i.e., the number of submatrices to be combined to obtain coded submatrices to provide the resilience to the maximum possible number of stragglers (for given number of nodes and their storage constraints). Next we propose a distributed matrix computation scheme which meets this exact lower bound on the weight of the coding. Further, we develop controllable trade-off between worker computation time and the privacy constraint for sparse input matrices in settings where the worker nodes are honest but curious. Numerical experiments conducted in Amazon Web Services (AWS) validate our assertions regarding straggler mitigation and computation speed for sparse matrices.

Index Terms—Distributed computing, MDS Codes, Stragglers, Sparsity, Privacy.

I. INTRODUCTION

Computing platforms are constantly stressed to meet the growing demands of end users for data processing. The increasing complexity of data tasks, such as deep neural network AI/ML models, and the sheer volumes of data to be processed, continue to hinder scalability.

Matrix computations serve as the fundamental building blocks for many data processing tasks in AI/ML and optimization. As data sizes increase, these computations involve high-dimensional matrices, requiring larger runtimes with all else constant. The underlying concept behind distributed computation is to break down the entire operation into smaller tasks and distribute them across multiple worker nodes. However, in these distributed systems, the overall execution time of a job can be significantly affected by slower or failed worker nodes, commonly known as “stragglers” [1].

Recently, a number of coding theory techniques [2]–[13] have been proposed to mitigate the effect of stragglers. A simple example is presented in [2] to illustrate a technique for computing $\mathbf{A}^T \mathbf{x}$ using three workers. The technique involves partitioning the matrix \mathbf{A} into two block-columns, denoted as $\mathbf{A} = [\mathbf{A}_0 | \mathbf{A}_1]$. The workers are then assigned specific tasks: one computes $\mathbf{A}_0^T \mathbf{x}$, another computes $\mathbf{A}_1^T \mathbf{x}$, and the third computes $(\mathbf{A}_0 + \mathbf{A}_1)^T \mathbf{x}$. Each worker then handles only half of the computational load, the system can recover $\mathbf{A}^T \mathbf{x}$ if

any two out of the three workers return their results. This means that the system is resilient to the failure or delay of one straggler. In general, the recovery threshold is an important metric defined as the minimum number of workers (τ) required to complete their tasks, enabling the recovery of $\mathbf{A}^T \mathbf{x}$ from any subset of τ worker nodes.

While there are several works that achieve the optimal recovery threshold [3], [5], [12], [13] for given number of nodes and storage constraints, they possess certain limitations. Real-world datasets, utilized in various domains such as optimization, deep learning, power systems, computational fluid dynamics etc. often consist of sparse matrices. An efficient exploitation of this sparsity can significantly decrease the overall time required for matrix computations [14]. However, techniques based on MDS codes [3], [5], [12], [13] construct dense linear combinations of submatrices; this eliminates the inherent sparsity in the matrix structure. As a consequence, the computation speed of worker nodes can be severely reduced. In this work, one of our objectives is to develop approaches that combine a relatively small number of submatrices while maintaining an optimal recovery threshold.

Another significant issue in distributed computation is the information leakage of the associated “input” matrix [15]–[18]. The assumption is that the input matrix \mathbf{A} is known to the central node, but the assigned smaller tasks should involve a protection against information leakage at the worker nodes. Several works [15]–[17] propose adding random matrices to the linear combinations of submatrices introduced by MDS codes with the goal of reducing the mutual information between the assigned encoded submatrices and the original matrix \mathbf{A} . This is again problematic for sparse matrices since the addition of dense random matrices can destroy the sparsity. Thus, we also aim to develop codes that optimize the trade-off between privacy and efficiency.

In this work, first we formulate the problem (Sec. II) and find a lower bound on the number of submatrices to be combined (Sec. III) for coded submatrices that will provide resilience to the maximum number of stragglers in a given system. Next, we develop a novel approach for distributed matrix-vector multiplication (Sec. IV) which meets that lower bound, maximizing sparsity preservation while providing resilience to the maximum number of stragglers. Our proposed approach involves a computationally efficient process to find a “good”

set of random coefficients that make the system numerically stable. Our approach also addresses the privacy issue through a controllable trade-off between privacy leakage and worker computation time for sparse input matrices (Sec. IV-B). Finally, we carry out experiments on an Amazon Web Services (AWS) which verify the effectiveness of our proposed methodology compared with baseline approaches in terms of different time, stability, and privacy metrics (Sec. V).

II. PROBLEM FORMULATION

In this work, we examine a distributed system comprising n worker nodes. The primary objective of this system is to calculate the product $\mathbf{A}^T \mathbf{x}$, where $\mathbf{A} \in \mathbb{R}^{t \times r}$ represents a sparse matrix and $\mathbf{x} \in \mathbb{R}^t$ denotes a vector. It is assumed that the workers are identical in terms of their memory capacity and computational speed. Specifically, each worker can store $\gamma_A = \frac{1}{k_A}$ fraction of the whole matrix \mathbf{A} , and also, the entire vector \mathbf{x} . In practical situations, stragglers may arise due to variations in computational speed or failures experienced by certain assigned workers at specific times [3].

In line with previous approaches, our initial step involves partitioning matrix \mathbf{A} into k_A distinct block-columns. Subsequently, we will distribute to each worker node a random linear combination of certain block-columns from \mathbf{A} along with the vector \mathbf{x} . Nevertheless, as discussed in Sec. I, assigning dense linear combinations could lead to the loss of inherent sparsity in the corresponding matrices. To avoid this issue, our goal is to allocate linear combinations involving a smaller number of submatrices [9], [19]. In order to quantify this approach, we introduce the concept of “weight” for the encoded submatrices. This measure serves as a crucial metric when dealing with sparse matrices in distributed computations.

Definition 1. We define the “weight” (ω_A) of the submatrix encoding procedure as the number of submatrices that are linearly combined to obtain each encoded submatrix. We assume homogeneous weights of the encoded submatrices across the worker nodes, i.e., every node will be assigned linear combinations of the same number of uncoded submatrices.

Thus, our goal is to obtain the optimal recovery threshold ($\tau = k_A$) while maintaining ω_A (for the assigned encoded submatrices) as low as possible. We also consider the privacy implications of our approach assuming that the worker nodes are honest but curious.

III. MINIMUM WEIGHT OF CODING

We consider a coded matrix-vector multiplication scheme with homogeneous weight, ω_A , where matrix \mathbf{A} is partitioned into k_A disjoint block-columns, $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_{k_A-1}$. Now we state the following proposition which provides a lower bound on ω_A for any coded matrix-vector multiplication scheme with resilience to $s = n - k_A$ stragglers.

Proposition 1. Consider a coded matrix-vector multiplication scheme aiming at resilience to $s = n - k_A$ stragglers out of n total nodes each of which can store $1/k_A$ fraction of

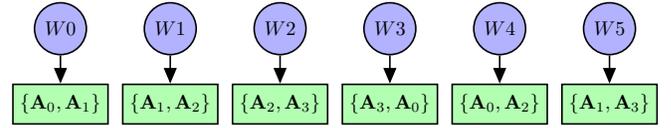


Fig. 1: Submatrix allocation for a system with $n = 6$, $s = 2$ and $\gamma_A = \frac{1}{4}$ according to Alg. 1. Here, the weight of every coded submatrix is $\omega_A = \lceil \frac{k_A(s+1)}{k_A+s} \rceil = 2$. Any $\{\mathbf{A}_i, \mathbf{A}_j\}$ indicates a random linear combination of \mathbf{A}_i and \mathbf{A}_j .

matrix \mathbf{A} . Any scheme that partitions \mathbf{A} into k_A disjoint block-columns has to maintain a minimum homogeneous weight $\lceil \frac{(n-s)(s+1)}{n} \rceil$.

Proof. Since the scheme aims at resilience to *any* s stragglers, any scheme needs to ensure the presence of any \mathbf{A}_i (where $i = 0, 1, \dots, k_A - 1$) in at least $s + 1$ different nodes. In other words, \mathbf{A}_i has to participate within the encoded submatrices in at least $s + 1$ different nodes. Now, we assume homogeneous weight ω_A , i. e., each of these n nodes is assigned a linear combination of ω_A uncoded submatrices from \mathbf{A} . Thus, we can say $n \omega_A \geq k_A(s + 1)$, hence,

$$\omega_A \geq \frac{(n-s)(s+1)}{n}.$$

Thus, the minimum homogeneous weight, $\hat{\omega}_A = \lceil \frac{(n-s)(s+1)}{n} \rceil$. ■

Now we state the following corollary (of Proposition 1) which considers different values of k_A in terms of s , and provides the corresponding optimal weights for coded sparse matrix-vector multiplication.

Corollary 1. Consider the same setting as Prop. 1 for coded matrix-vector multiplication. Now,

- (i) if $k_A > s^2$, then $\hat{\omega}_A = s + 1$.
- (ii) if $s \leq k_A \leq s^2$, then $\lceil \frac{s+1}{2} \rceil \leq \hat{\omega}_A \leq s$.

Proof. Since $n = k_A + s$, from Prop. 1, we have

$$\hat{\omega}_A = \lceil \frac{k_A(s+1)}{k_A+s} \rceil = \lceil \frac{1+s}{1+\frac{s}{k_A}} \rceil; \quad (1)$$

hence, $\hat{\omega}_A$ is a non-decreasing function of k_A for fixed s .

Part (i): When $k_A > s^2$, we have $\frac{s}{k_A} < \frac{1}{s}$, and $\frac{1+s}{1+\frac{s}{k_A}} > \frac{1+s}{1+\frac{1}{s}} = s$. Thus, from (1), $\hat{\omega}_A > s$. In addition, from (1), for any $s \geq 0$, we have $\hat{\omega}_A \leq s + 1$. Thus, we have $\hat{\omega}_A = s + 1$.

Part (ii): If $k_A = s^2$, from (1), we have $\hat{\omega}_A = s$. Similarly, if $k_A = s$, from (1), we have $\hat{\omega}_A = \lceil \frac{s+1}{2} \rceil$. Thus, the non-decreasing property of $\hat{\omega}_A$ in terms of k_A concludes the proof. ■

Now we describe a motivating example below where the encoding scheme meets the lower bound mentioned in Prop. 1.

Example 1. Consider a toy system with $n = 6$ worker nodes each of which can store $1/4$ fraction of matrix \mathbf{A} . We partition matrix \mathbf{A} into $k_A = 4$ disjoint block-columns,

$\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$. According to Prop. 1, the optimal weight ω_A can be as low as $\left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil = 2$. Now, we observe that the way the jobs are assigned in Fig. 1 meets that lower bound, where random linear combinations of $\omega_A = 2$ submatrices are assigned to the nodes. It can be verified that this system has a recovery threshold $\tau = k_A = 4$, and thus, it is resilient to any $s = 2$ stragglers.

IV. PROPOSED APPROACH

In this section, we detail our overall approach for distributed matrix-vector multiplication which is outlined in Alg. 1. We partition matrix \mathbf{A} into k_A block columns, $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_{k_A-1}$, and assign a random linear combination of ω_A (weight) submatrices of \mathbf{A} to every worker node. We show that for given n and k_A , our proposed approach provides resilience to maximum number of stragglers, $s = n - k_A$. In addition, our coding scheme maintains the minimum weight of coding as mentioned in Prop. 1.

Formally, we set $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil$, and assign a linear combination of $\mathbf{A}_i, \mathbf{A}_{i+1}, \mathbf{A}_{i+2}, \dots, \mathbf{A}_{i+\omega_A-1}$ (indices modulo k_A) to worker node W_i , for $i = 0, 1, 2, \dots, k_A - 1$, where the linear coefficients are chosen randomly from a continuous distribution. Next, we assign a random linear combination of $\mathbf{A}_{i\omega_A}, \mathbf{A}_{i\omega_A+1}, \mathbf{A}_{i\omega_A+2}, \dots, \mathbf{A}_{(i+1)\omega_A-1}$ (indices modulo k_A) to worker node W_i , for $i = k_A, k_A + 1, \dots, n - 1$. Note that every worker node also receives the vector \mathbf{x} . Once the fastest $\tau = k_A$ worker nodes finish and return their computation results, the central node decodes $\mathbf{A}^T \mathbf{x}$. Note that we assume $k_A \geq s$, i.e., at most *half* of the nodes may be stragglers.

A. Straggler Resilience Guarantee

Next we state the following lemma which would assist us to prove Theorem 1 which discusses straggler resilience of our proposed scheme.

Lemma 1. Choose any $m \leq k_A$ worker nodes out of all n nodes in the distributed system. Now, if we assign the jobs to the worker nodes according to Alg. 1, the total number of participating uncoded \mathbf{A} submatrices within those m worker nodes is lower bounded by m .

Proof. First we partition all n worker nodes into *two* sets where the first set, \mathcal{W}_0 includes the first k_A nodes and the second set, \mathcal{W}_1 , includes the next s worker nodes, i.e., we have

$$\begin{aligned} \mathcal{W}_0 &= \{W_0, W_1, W_2, \dots, W_{k_A-1}\}; \\ \text{and } \mathcal{W}_1 &= \{W_{k_A}, W_{k_A+1}, \dots, W_{n-1}\}. \end{aligned}$$

Thus, we have $|\mathcal{W}_0| = k_A$ and $|\mathcal{W}_1| = s \leq k_A$. Now, we choose any $m \leq k_A$ worker nodes, where we choose m_0 nodes from \mathcal{W}_0 and m_1 nodes from \mathcal{W}_1 , so that $m = m_0 + m_1$. We denote set of the participating uncoded \mathbf{A} submatrices within those nodes as \mathcal{A}_0 and \mathcal{A}_1 , respectively. Hence, to prove the lemma, we need to show $|\mathcal{A}_0 \cup \mathcal{A}_1| \geq m$, for any $m \leq k_A$.

Algorithm 1: Proposed scheme for distributed matrix-vector multiplication

Input : Matrix \mathbf{A} , vector \mathbf{x} , n -number of workers, s -number of stragglers, storage fraction $\gamma_A = \frac{1}{k_A}$, such that $k_A \geq s$.

- 1 Partition \mathbf{A} into k_A disjoint block-columns;
- 2 Set weight $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil$;
- 3 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
- 4 **if** $i < k_A$ **then**
- 5 Define $T = \{i, i + 1, \dots, i + \omega_A - 1\}$ (reduced modulo k_A);
- 6 **else**
- 7 Define $T = \{i\omega_A, i\omega_A + 1, \dots, (i + 1)\omega_A - 1\}$ (reduced modulo k_A);
- 8 **end**
- 9 Create a random vector \mathbf{r} of length k_A with entries $r_m, 0 \leq m \leq k_A - 1$;
- 10 Create a random linear combination of \mathbf{A}_q 's where $q \in T$, thus $\tilde{\mathbf{A}}_i = \sum_{q \in T} r_q \mathbf{A}_q$;
- 11 Assign encoded submatrix $\tilde{\mathbf{A}}_i$ and the vector \mathbf{x} to worker node W_i ;
- 12 **end**

Output : The central node recovers $\mathbf{A}^T \mathbf{x}$ from the returned results by the fastest k_A nodes.

First, according to Alg. 1, we assign a random linear combination of $\mathbf{A}_i, \mathbf{A}_{i+1}, \mathbf{A}_{i+2}, \dots, \mathbf{A}_{i+\omega_A-1}$ (indices modulo k_A) to worker node $W_i \in \mathcal{W}_0$. Thus, the participating submatrices are assigned in a cyclic fashion [20], and the total number of participating submatrices within any m_0 nodes of \mathcal{W}_0 is

$$|\mathcal{A}_0| \geq \min(m_0 + \omega_A - 1, k_A). \quad (2)$$

Next, we state the following claim for the number of participating submatrices in \mathcal{W}_1 , with the proof in Appendix A.

Claim 1. Choose any $m_1 \geq \omega_A$ nodes from \mathcal{W}_1 . The number of participating submatrices within these nodes, $|\mathcal{A}_1| = k_A$.

Now, if $m_1 \leq \omega_A - 1$, from (2) we have

$$\begin{aligned} |\mathcal{A}_0 \cup \mathcal{A}_1| &\geq |\mathcal{A}_0| = \min(m_0 + \omega_A - 1, k_A) \\ &\geq \min(m_0 + m_1, k_A) \geq m, \end{aligned}$$

since $m = m_0 + m_1 \leq k_A$. And, if $m_1 \geq \omega_A$, from Claim 1 we can say,

$$|\mathcal{A}_0 \cup \mathcal{A}_1| \geq |\mathcal{A}_1| = k_A \geq m,$$

which concludes the proof of the lemma. \blacksquare

Example 2. Consider the same scenario in Example 1, where $k_A = 4$ and $s = 2$, therefore, $\mathcal{W}_0 = \{W_0, W_1, W_2, W_3\}$ and $\mathcal{W}_1 = \{W_4, W_5\}$. Now, choose $m = 3$ nodes, W_0, W_1 and W_4 . Thus, $m_0 = 2$ and $m_1 = 1$. Now, from the figure, we have $\mathcal{A}_0 = \{\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2\}$ and $\mathcal{A}_1 = \{\mathbf{A}_0, \mathbf{A}_1\}$. Hence,

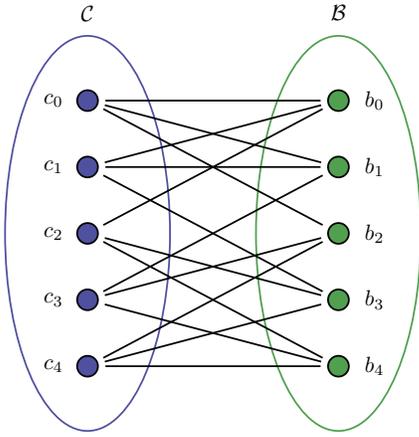


Fig. 2: A bipartite graph $\mathcal{G} = \mathcal{C} \cup \mathcal{B}$ with $|\mathcal{C}| = |\mathcal{B}| = 5$ where the set of equations is \mathcal{C} and the set of unknowns is \mathcal{B} . Here, $\omega_A = 3$.

$|\mathcal{A}_0 \cup \mathcal{A}_1| = 3 \geq m$. Similar properties can be shown for any choice $m \leq k_A = 4$ different nodes.

Now we state the following theorem which provides the guarantee of resilience to maximum number of stragglers for given storage constraints.

Theorem 1. Assume that a system has n worker nodes each of which can store $1/k_A$ fraction of matrix \mathbf{A} and the whole vector \mathbf{x} for the distributed matrix-vector multiplication $\mathbf{A}^T \mathbf{x}$. If we assign the jobs according to Alg. 1, we achieve resilience to $s = n - k_A$ stragglers.

Proof. According to Alg. 1, first we partition matrix \mathbf{A} into k_A disjoint block-columns. Thus, to recover the matrix-vector product, $\mathbf{A}^T \mathbf{x}$, we need to decode all k_A vector unknowns, $\mathbf{A}_0^T \mathbf{x}, \mathbf{A}_1^T \mathbf{x}, \mathbf{A}_2^T \mathbf{x}, \dots, \mathbf{A}_{k_A-1}^T \mathbf{x}$. We denote the set of these k_A unknowns as \mathcal{B} . Now we choose an arbitrary set of k_A worker nodes each of which corresponds to an equation in terms of ω_A of those k_A unknowns. Denoting the set of k_A equations as \mathcal{C} , we can say, $|\mathcal{B}| = |\mathcal{C}| = k_A$.

Now we consider a bipartite graph $\mathcal{G} = \mathcal{C} \cup \mathcal{B}$, where any vertex (equation) in \mathcal{C} is connected to some vertices (unknowns) in \mathcal{B} which participate in the corresponding equation. Thus, each vertex in \mathcal{C} has a neighborhood of cardinality ω_A in \mathcal{B} . An example with $k_A = 5$ and $\omega_A = 3$ is shown in Fig. 2.

Our goal is to show that there exists a perfect matching among the vertices of \mathcal{C} and \mathcal{B} . To do so, we consider $\bar{\mathcal{C}} \subseteq \mathcal{C}$, where $|\bar{\mathcal{C}}| = m \leq k_A$. Now, we denote the neighbourhood of $\bar{\mathcal{C}}$ as $\mathcal{N}(\bar{\mathcal{C}}) \subseteq \mathcal{B}$. Thus, according to Lemma 1, for any $m \leq k_A$, we can say that $|\mathcal{N}(\bar{\mathcal{C}})| \geq m$. So, according to Hall's marriage theorem [21], we can say that there exists a perfect matching among the vertices of \mathcal{C} and \mathcal{B} .

Next we consider the largest matching where the vertex $c_i \in \mathcal{C}$ is matched to the vertex $b_j \in \mathcal{B}$, which indicates that b_j participates in the equation corresponding to c_i . Now, considering k_A equations and k_A unknowns, we construct the $k_A \times k_A$ coding (or decoding) matrix \mathbf{H} where row i corresponds to the equation associated to c_i where b_j participates.

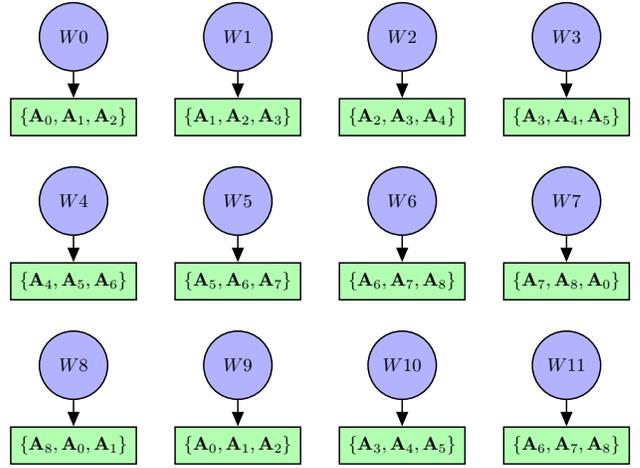


Fig. 3: Submatrix allocation for $n = 12$ workers and $s = 3$ stragglers, with $\gamma_A = \frac{1}{9}$ according to Alg. 1. Here, the weight of every submatrix is $\omega_A = \lfloor \frac{k_A(s+1)}{k_A+s} \rfloor = 3$. Any $\{\mathbf{A}_i, \mathbf{A}_j, \mathbf{A}_k\}$ indicates a random linear combination of the corresponding submatrices where the coefficients are chosen i.i.d. at random from a continuous distribution.

We replace row i of \mathbf{H} by \mathbf{e}_j where \mathbf{e}_j is a unit row-vector of length k_A with the j -th entry being 1, and 0 otherwise. Thus we have a $k_A \times k_A$ matrix where each row has only one non-zero entry which is 1. In addition, since we have a perfect matching, \mathbf{H} will have only one non-zero entry in every column. Thus, \mathbf{H} is a permutation of the identity matrix, and therefore, \mathbf{H} is full rank. Since the matrix is full rank for a choice of definite values, according to Schwartz-Zippel lemma [22], the matrix continues to be full rank for random choices of non-zero entries. Thus, the central node can recover all k_A unknowns from any set of k_A worker nodes. ■

Example 3. Consider a system with $n = 12$ nodes each of which can store $1/9$ -th fraction of matrix \mathbf{A} . We partition \mathbf{A} as $\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_8$. According to Alg. 1, we set the weight $\omega_A = \lfloor \frac{k_A(s+1)}{k_A+s} \rfloor = 3$, and assign random linear combinations of ω_A submatrices to each node as shown in Fig. 3. It can be verified that $\mathbf{A}^T \mathbf{x}$ can be recovered from any $\tau = k_A = 9$ nodes, therefore, the scheme is resilient to any $s = 3$ stragglers.

Remark 1. While our proposed approach meets the lower bound on the weight as mentioned in Prop. 1, the approach in [11] assigns a weight $\min(s+1, k_A)$ which can often be higher than ours (e.g., Examples 1 and 3), and thus, may lead to reduction in worker computation speed.

1) *Computational Complexity for a Worker Node:* In this work, we assume that the “input” matrix, $\mathbf{A} \in \mathbb{R}^{t \times r}$, is sparse, i.e., most of the entries of \mathbf{A} are zero. Let us assume that the probability for any entry of \mathbf{A} to be non-zero is μ , where $\mu > 0$ is very small. According to Alg. 1, we combine ω_A submatrices (of size $t \times r/k_A$) to obtain the coded submatrices and assign them to the worker nodes. Hence, the probability for any entry of any coded submatrix to be non-zero is $1 -$

$(1 - \mu)^{\omega_A}$ which can be approximated by $\omega_A \mu$. Thus, in our approach, the per worker node computational complexity is $\mathcal{O}\left(\omega_A \mu \times \frac{rt}{k_A}\right)$ where $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil$.

On the other hand, the dense coded approaches [5], [12], [13] combine k_A submatrices for encoding, hence, their per worker node computational complexity is $\mathcal{O}\left(k_A \mu \times \frac{rt}{k_A}\right) = \mathcal{O}(\mu \times rt)$ which is $\frac{k_A}{\omega_A} \approx \frac{s+k_A}{s+1}$ times higher than that of ours. Moreover, the recent sparse matrix computations approach in [11] combines $s+1$ submatrices for encoding (when $s < k_A$). Thus, its corresponding computational complexity is $\mathcal{O}\left((s+1)\mu \times \frac{rt}{k_A}\right)$; approximately $(1+s/k)$ times higher than that of ours. We clarify this with the following example.

Example 4. Consider the same setting in Example 3 where $n = 12$, $k_A = 9$ and $s = 3$. In this scenario, the recent work [11] assigns random linear combinations of $\min(s+1, k_A) = 4$ submatrices to each node. Thus, our proposed approach enjoys a 25% decrease in computational complexity, which could significantly enhance the overall computational speed.

2) Numerical Stability and Coefficient Determination Time:

In this section, we discuss the numerical stability of our proposed distributed matrix computations scheme. The condition number is widely regarded as a significant measure of numerical stability for such a system [3], [12], [13]. In the context of a system consisting of n workers and s stragglers, the worst-case condition number (κ_{worst}) is defined as the highest condition number among the decoding matrices when considering all possible combinations of s stragglers. In methods involving random coding like ours, the idea is to generate random coefficients multiple (e.g., 20) times and selecting the set of coefficients that results in the lowest κ_{worst} among those trials.

In our proposed method, we partition matrix \mathbf{A} into k_A disjoint block-columns, which underscores the necessity to recover k_A vector unknowns. Consequently, in each attempt, we must determine the condition numbers of $\binom{n}{k_A}$ decoding matrices, each of size $k_A \times k_A$. This whole process has a total complexity of $\mathcal{O}\left(\binom{n}{k_A} k_A^3\right)$. On the other hand, the recent sparse matrix computation techniques, such as sparsely coded straggler (SCS) optimal scheme discussed in [20] or the class-based scheme discussed in [9] partition matrix \mathbf{A} into $\Delta_A = \text{LCM}(n, k_A)$ block-columns. Thus, in each attempt, they need to ascertain the condition numbers of $\binom{n}{k_A}$ matrices, each of which has a size $\Delta_A \times \Delta_A$, resulting in a total complexity of $\mathcal{O}\left(\binom{n}{k_A} \Delta_A^3\right)$. Since Δ_A can be considerably larger than k_A , those methods involve significantly more complexity compared to our proposed scheme. For instance, if we consider a scenario where n and k_A are co-prime, then $\Delta_A = nk_A$, and thus the complexity of the approaches presented in [9], [20] is approximately $\mathcal{O}(n^3)$ times higher than our method.

B. Private Matrix-vector Multiplication

Now, we discuss how we can modify Alg. 1 to add protection against information leakage of the “input” matrix \mathbf{A} in

Algorithm 2: Proposed scheme for Private distributed matrix-vector multiplication for non-colluding nodes

Input : Matrix $\mathbf{A} \in \mathbb{F}^{t \times r}$, vector $\mathbf{x} \in \mathbb{F}^{t \times 1}$, n -number of nodes, storage fraction $\frac{1}{k_A}$, where $n > k_A$.

- 1 Create a sparse random matrix $\mathbf{S} \in \mathbb{F}^{t \times r/k_A}$, where the probability of any entry to be non-zero is μ ;
- 2 Create a random vector \mathbf{r} of length n ;
- 3 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
- 4 Create encoded submatrix $\tilde{\mathbf{A}}_i$ according to Alg. 1;
- 5 Assign submatrix $\bar{\mathbf{A}}_i = \tilde{\mathbf{A}}_i + \mathbf{r}_i \mathbf{S}$ to worker W_i ;
- 6 Assign vector \mathbf{x} to worker node W_i ;
- 7 **end**

Output : The central node recovers $\mathbf{A}^T \mathbf{x}$ from the returned results by the fastest $k_A + 1$ nodes.

the worker nodes, which we assume are honest but curious. The traditional idea developed in several private distributed computations approaches [15], [16] is to add dense random matrices to the submatrices of the “input” matrix. While this can provide protection against information leakage up to certain levels, it substantially increases the number of non-zero entries in the encoded submatrices of an originally sparse input matrix, which can reduce the overall computation speed.

In our scheme, we propose that the central node will generate a sparse matrix $\mathbf{S} \in \mathbb{F}^{t \times r/k_A}$ where the probability of any entry being non-zero is μ . Next, the central node will add \mathbf{S} to all the encoded submatrices to be assigned to the worker nodes according to Alg. 1. In other words, if the central node was supposed to send the encoded submatrix $\tilde{\mathbf{A}}_i$ to worker node W_i according to Alg. 1, then for private sparse matrix computations, the central node will send $\bar{\mathbf{A}}_i = \tilde{\mathbf{A}}_i + \mathbf{S}$ to worker node W_i . The upcoming corollary proves that the central node can recover the final result, $\mathbf{A}^T \mathbf{x}$ from any $k_A + 1$ nodes (in a similar process as in Sec. IV). Note that the central node sends the vector \mathbf{x} to all n nodes. The overall procedure for private matrix-vector multiplication is outlined in Alg. 2.

Corollary 2. Assume that a system has n worker nodes each of which can store $1/k_A$ fraction of matrix \mathbf{A} for conducting private matrix-vector multiplication $\mathbf{A}^T \mathbf{x}$. If we assign the jobs according to Alg. 2 to achieve our desired level of protection against information leakage of \mathbf{A} , we achieve resilience to $s = n - (k_A + 1)$ stragglers.

Proof. We prove the corollary in a similar fashion as we have proved Theorem 1. Instead of k_A vector unknowns, $\mathbf{A}_0^T \mathbf{x}, \mathbf{A}_1^T \mathbf{x}, \mathbf{A}_2^T \mathbf{x}, \dots, \mathbf{A}_{k_A-1}^T \mathbf{x}$, to recover $\mathbf{A}^T \mathbf{x}$, we have one more unknown, $\mathbf{S}^T \mathbf{x}$ involved in this process. Similar to the proof of Theorem 1, we denote the set of these $k_A + 1$ unknowns as \mathcal{B} , and choose an arbitrary set of $k_A + 1$ worker nodes each of which corresponds to an equation in terms of $\omega_A + 1$ of those $k_A + 1$ unknowns. Denoting the set of $k_A + 1$ equations as \mathcal{C} , we can say, $|\mathcal{B}| = |\mathcal{C}| = k_A + 1$.

We again consider a bipartite graph $\mathcal{G} = \mathcal{C} \cup \mathcal{B}$, and claim that a perfect matching exists between the vertices in \mathcal{C} and \mathcal{B} .

The reason is that the new unknown $\mathbf{S}^T \mathbf{x}$ participates in every equation, hence, the size of the neighborhood of $\bar{C} \in \mathcal{C}$ will always increase by 1 (as compared to Theorem 1) when $|\bar{C}| = m \leq k_A$. Thus, for any \bar{C} , when $|\bar{C}| = m \leq k_A + 1$, the size of the neighborhood $|\mathcal{N}(\bar{C})| \geq m$. This proves the perfect matching, and then, similar to the proof of Theorem 1, using Schwartz-Zippel lemma [22], we can prove the corollary. ■

We consider a system of non-colluding worker nodes which are honest but curious. In this setting, in order to be private from an information-theoretic standpoint, the encoded matrices $\bar{\mathbf{A}}_i$ should not leak any information about the data matrix \mathbf{A} . In this regard, denote the mutual information of two random variables X and Y as $\mathcal{I}(X, Y)$. A perfectly private scheme in our setting must satisfy the information-theoretic constraint, $\mathcal{I}(\bar{\mathbf{A}}_i, \mathbf{A}) = 0$, for $i = 0, 1, \dots, n-1$. Denoting $\mathcal{H}(X, Y)$ as the joint entropy of two random variables X and Y , for our scheme we can write

$$\begin{aligned} \mathcal{I}(\bar{\mathbf{A}}_i; \mathbf{A}) &= \mathcal{I}(\tilde{\mathbf{A}}_i + \mathbf{S}; \mathbf{A}) \\ &= \mathcal{H}(\tilde{\mathbf{A}}_i + \mathbf{S}) - \mathcal{H}(\tilde{\mathbf{A}}_i + \mathbf{S} | \mathbf{A}) = \mathcal{H}(\tilde{\mathbf{A}}_i + \mathbf{S}) - \mathcal{H}(\mathbf{S} | \mathbf{A}) \end{aligned}$$

Now, for small η , the number of non-zero entries in any $\tilde{\mathbf{A}}_i$ is approximately $\omega_A \eta \times \frac{rt}{k_A}$. Thus, we have

$$\begin{aligned} \mathcal{I}(\bar{\mathbf{A}}_i; \mathbf{A}) &\approx (\omega_A \eta + \mu - \omega_A \eta \mu) \frac{rt}{k_A} \log|\mathbb{F}| - \mu \frac{rt}{k_A} \log|\mathbb{F}| \\ &= \omega_A \eta (1 - \mu) \frac{rt}{k_A} \log|\mathbb{F}| \end{aligned} \quad (3)$$

Thus, $\mathcal{I}(\bar{\mathbf{A}}_i, \mathbf{A})$ decreases with the increase of μ ; if the central node uses a denser \mathbf{S} , the system will have more protection, at the expense of longer computation times due to sparsity being destroyed. The system will be fully protected if $\mu = 1$, in other words, when \mathbf{S} is fully dense.

Remark 2. A recent work [23] also studied this privacy issue in sparse matrix computations for a different setting of distributed computation. In that setting, the worker nodes are partitioned into two non-communicating clusters, the untrusted cluster and the partly trusted cluster, and different number of tasks are assigned to different nodes. This objective is different than our focus on being resilient to the maximum number of stragglers.

V. NUMERICAL EXPERIMENTS

In this section, we evaluate the effectiveness of our proposed approach by conducting numerical experiments and comparing its performance with various competing methods [5], [9], [11]–[13], [20]. Note that there are several other works specifically developed for sparse matrix computations. Among them, the approach in [14] does not provide resilience to maximum number of stragglers for given storage constraints. The approach in [23] partitions the worker nodes into untrusted and partly trusted cluster, which is not aligned to our assumption. The approach in [24] assigns some jobs to the central node to reduce the probability of rank-deficiency in the decoding, which

is also not in line of our assumptions. So, in the numerical experiment section, we do not consider these approaches.

We explore two different distributed systems: the first one consists of $n = 30$ worker nodes with $s = 5$ stragglers and the other consists of $n = 36$ nodes with $s = 8$ stragglers. We focus on a sparse input matrix \mathbf{A} sized $40,000 \times 31,500$ and a dense vector \mathbf{x} of length 40,000. We consider two distinct scenarios in which the sparsity of \mathbf{A} is 98%, and 99%, respectively. This implies that randomly selected 98% and 99% entries, respectively, in the matrix \mathbf{A} are zero. It is worth noting that there exist numerous practical instances where data matrices demonstrate such (or, even more) levels of sparsity (refer to [25] for specific examples). The experiments are carried out on an AWS (Amazon Web Services) cluster, utilizing a `c5.18xlarge` machine as the central node and `t2.small` machines as the worker nodes.

Worker computation time: Table I presents a comparison among different methods based on the computation time required by worker nodes to complete their respective tasks. In these scenarios, where $k_A = 25$ or 28, the approaches described in [5], [12], [13] allocate linear combinations of k_A submatrices to the worker nodes. Consequently, the original sparsity of matrix \mathbf{A} is lost within the encoded submatrices. As a result, the worker nodes experience a significantly increased processing time for their tasks compared to our proposed approach or the methods outlined in [9], [11], [20], which are specifically designed for sparse matrices and involve smaller weights.

To discuss the effectiveness of our approach in more details, we compare the weight of the coding of our approach against the approach in [11]. In the first scenario, when $n = 30$ and $s = 5$, our approach sets the weight $\left\lceil \frac{(n-s)(s+1)}{n} \right\rceil = \left\lceil \frac{25 \times 6}{30} \right\rceil = 5$, whereas the approach in [11] uses a weight $\min(s+1, k_A) = \min(6, 25) = 6$. Thus, our approach involves around 17% less computational complexity per worker node, which is supported by the results in Table I. Similarly, when $n = 36$ and $s = 8$, our proposed approach involves a weight $\left\lceil \frac{28 \times 9}{36} \right\rceil = 7$, which is smaller than the corresponding weight, $s + 1 = 9$, used by the approach in [11].

Communication delay: Table I also illustrates the delay incurred during the transmission of encoded submatrices from the central node to the worker node. The approaches presented in [5], [12], and [13] employ dense linear combinations of submatrices, resulting in a significant increase in the number of non-zero entries within the encoded submatrices. Consequently, transmitting these large number of non-zero entries leads to a substantial communication delay within the system. In contrast, our proposed scheme mitigates this issue by utilizing encoded submatrices formed through linear combinations of only a limited number of uncoded submatrices which significantly reduces the corresponding communication delay.

For example, consider the scenario when $n = 36$, $s = 8$ and \mathbf{A} is 99% sparse. In this scenario, the approach in [5] needs to

TABLE I: Comparison of worker computation time and communication delay (matrix transmission time) for matrix-vector multiplication for $n = 30, s = 5$, and $n = 36, s = 8$, when randomly chosen 98% and 99% entries of matrix \mathbf{A} are zero.

| METHODS | $n = 30$ AND $s = 5$ | | | | $n = 36$ AND $s = 8$ | | | |
|------------------------|----------------------|-------------|--------------------|-------------|----------------------|-------------|--------------------|-------------|
| | COMP. TIME (IN MS) | | COMM. DELAY (IN S) | | COMP. TIME (IN MS) | | COMM. DELAY (IN S) | |
| | 99% | 98% | 99% | 98% | 99% | 98% | 99% | 98% |
| POLY. CODE [5] | 61.4 | 62.3 | 0.67 | 1.14 | 55.7 | 56.3 | 0.52 | 0.95 |
| ORTHO POLY [12] | 62.2 | 61.7 | 0.69 | 1.17 | 56.2 | 56.4 | 0.49 | 0.91 |
| RKRP CODE [13] | 60.3 | 61.1 | 0.65 | 1.11 | 56.8 | 57.4 | 0.51 | 0.93 |
| SCS OPT. SCH. [20] | 24.1 | 38.3 | 0.24 | 0.37 | 28.1 | 41.3 | 0.28 | 0.42 |
| CLASS-BASED [9] | 17.3 | 28.2 | 0.20 | 0.31 | 22.1 | 33.7 | 0.24 | 0.35 |
| CYCLIC CODE [11] | 19.5 | 33.4 | 0.23 | 0.35 | 26.7 | 37.6 | 0.27 | 0.39 |
| Proposed Scheme | 16.7 | 27.7 | 0.19 | 0.32 | 21.8 | 33.9 | 0.24 | 0.34 |

TABLE II: Comparison among different approaches in terms of worst case condition number (κ_{worst}) and the corresponding required time for 10 trials to find a good set of random coefficients

| METHODS | κ_{worst} FOR $n = 30, s = 5$ | REQ. TIME FOR 10 TRIALS (IN S) |
|---------------------|---|-----------------------------------|
| POLY. CODE [5] | 1.47×10^{13} | 0 |
| ORTHO-POLY [12] | 1.40×10^8 | 0 |
| RKRP CODE [13] | 1.76×10^6 | 81.84 |
| SCS OPT. SCH. [20] | 4.68×10^7 | 1138.6 |
| CLASS BASED [9] | 7.16×10^6 | 1479.3 |
| CYCLIC CODE [11] | 1.06×10^7 | 78.38 |
| PROP. SCHEME | 8.21×10^6 | 77.41 |

transmit up to $0.01 \times 28 \times \frac{40,000 \times 31,500}{28} = 1.26 \times 10^7$ number of non-zero entries to each node. The corresponding number for the approach in [11], [19] is $0.01 \times (s+1) \times \frac{40,000 \times 31,500}{28} = 4.05 \times 10^6$. On the other hand, the corresponding number for our proposed method is $0.01 \times \left\lceil \frac{k_A(s+1)}{n} \right\rceil \times \frac{40,000 \times 31,500}{28} = 3.15 \times 10^6$, which is smaller than the previous ones, and clarifies the reduction of communication delay as mentioned in Table I.

Numerical stability: Next, we assess the numerical stability of distributed systems using different coded matrix computation techniques. We examine the condition numbers of the decoding matrices for various combinations of n workers and s stragglers. By comparing the worst-case condition number (κ_{worst}) across different methods, we present the κ_{worst} values in Table II. The polynomial code approach [5] involves ill-conditioned Vandermonde matrices and demonstrates significant numerical instability, as evidenced by its notably high value of κ_{worst} . Our proposed approach, among the numerically stable methods, exhibits smaller κ_{worst} value compared to the method in [12] where the condition numbers increases exponentially in terms of $s = n - k_A$. Note that the approach in [13] provides slightly smaller κ_{worst} value than ours; however, as mentioned in Table I, the worker computation time and the communication delay are significantly higher in that case, since they assign dense linear combinations to the worker nodes.

Coefficient determination time: Next, Table II shows a comparative analysis of various methods with respect to the time required for performing 20 trials to obtain a ‘‘good’’

set of random coefficients that ensures numerical stability of the system. As explained in Section IV-A2, the techniques proposed in [20] and [9] involve partitioning matrix \mathbf{A} into $\Delta_A = \text{LCM}(n, k_A)$ block-columns. For instance, when $n = 30$ and $s = 5$, $\Delta_A = 150$ is significantly larger than $k_A = 25$, which denotes the partition level in our approach. Consequently, when dealing with higher-sized matrices to determine the condition number, the methods proposed in [20] and [9] necessitate considerably more time compared to our approach.

Trade-off between privacy and worker computation time: Next, we compare the trade-off between protection against information leakage and the worker node computation time. Consider a 99% sparse matrix \mathbf{A} of size $40,000 \times 31,500$, i.e., 99% entries of \mathbf{A} are zero. We assume the nodes to be honest but curious. Now, according to the discussion in Sec. IV-B, we add matrix \mathbf{S} to the encoded submatrices of \mathbf{A} . Fig. 4 shows the trade-off between the privacy (in terms of μ) and the worker computation time for two different scenarios of n and s . The extreme case $\mu = 0$ indicates that the worker node receives only the coded submatrices as outlined by Alg. 1, and in that case, the computation speed is very high. On the other extreme, as clarified in (3), when $\mu = 1$, i.e., dense noise is added to the assigned submatrices, then $\mathcal{I}(\hat{\mathbf{A}}_i; \mathbf{A}) = 0$, which indicates the full protection against information leakage from the honest but curious worker nodes. However, that comes with a sacrifice in the worker node computation speed. In this experiment, we see that the worker computation time is most sensitive at small values of μ , i.e., when less than 20% non-zero entries are being added. After this point, privacy can be improved with little downside to computational time. Note that the approaches in [9], [20], while being specifically suited to sparse matrices, do not address the privacy issue.

VI. CONCLUSION

In this study, we devised a distributed scheme for multiplying large matrices by vectors, specifically designed for sparse input matrices. First we found a lower bound on the weight for the encoding of any scheme for the resilience to the maximum number of stragglers for given storage constraints. Our proposed straggler-optimal approach meets the lower bound and maintains the inherent sparsity of the input matrix \mathbf{A} up to a certain extent. As a result, it substantially reduces

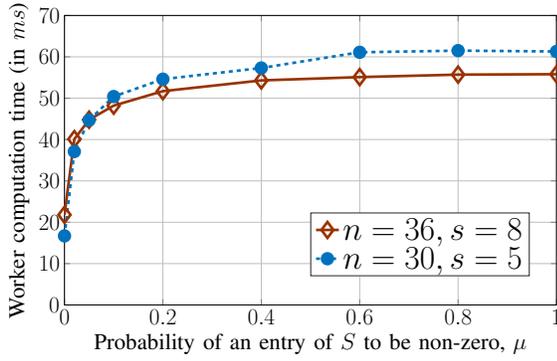


Fig. 4: Trade-off between the protection against information leakage and the worker computation time. A larger μ enhances the protection, but reduces the computation speed.

both computation and communication delays compared to dense coded methods. We also explored the privacy aspect of sparse matrix computations when the nodes are honest but curious. We achieved a controllable balance between the preserved sparsity level and information leakage. Our claims were corroborated through numerical experiments conducted on an AWS cluster.

A future direction can include developing schemes for sparse distributed matrix-matrix multiplication which meets the lower bound on the weight. Another direction may include developing sparsely coded schemes with protection against information leakage when the worker nodes can collude among them.

APPENDIX

A. Proof of Claim 1

Proof. Consider the worker nodes in \mathcal{W}_1 . According to Alg. 1, we assign a linear combination of $\mathbf{A}_{i\omega_A}, \mathbf{A}_{i\omega_A+1}, \mathbf{A}_{i\omega_A+2}, \dots, \mathbf{A}_{(i+1)\omega_A-1}$ (indices modulo k_A) to worker node W_i , for $i = k_A, k_A + 1, \dots, n - 1$. Thus, the participating submatrices in worker node W_{k_A} are $\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_{\omega_A-1}$ (indices reduced modulo k_A). Similarly, the participating submatrices in W_{k_A+1} are $\mathbf{A}_{\omega_A}, \mathbf{A}_{\omega_A+1}, \dots, \mathbf{A}_{2\omega_A-1}$ (indices reduced modulo k_A). In a consequence, ω_A number of submatrices participate in each of those s worker nodes sequentially in an increasing order in terms of their indices (reduced modulo k_A).

Now, denote the number of appearances of any submatrix \mathbf{A}_i within the nodes in \mathcal{W}_1 by $\mathbf{v}_i \geq 0$. Thus, for any $0 \leq j, k \leq k_A - 1$, we have $|\mathbf{v}_j - \mathbf{v}_k| \leq 1$, where $\sum_{i=0}^{k_A-1} \mathbf{v}_i = s\omega_A$. Thus, the average of these \mathbf{v}_i 's is $\rho = \frac{s\omega_A}{k_A}$. If ρ is an integer, then $\mathbf{v}_i = \lfloor \rho \rfloor = \rho$ for $i = 0, 1, 2, \dots, k_A - 1$, since for every pair of j, k , we have $|\mathbf{v}_j - \mathbf{v}_k| \leq 1$. Similarly, if ρ is not an integer, then $\mathbf{v}_i \geq \lfloor \rho \rfloor$. Thus, within all s nodes of \mathcal{W}_1 , every submatrix participates in at least $\lfloor \rho \rfloor$ times over $\lfloor \rho \rfloor$ distinct nodes. In other words, any submatrix may not participate in at most $s - \lfloor \rho \rfloor$ nodes within the nodes of \mathcal{W}_1 .

First, consider the case, $k_A = s$. Here, every submatrix participates in $\lfloor \rho \rfloor = \omega_A$ nodes, therefore, any submatrix does

not participate in $s - \omega_A$ nodes. But, we choose any $m_1 \geq \omega_A$ nodes in \mathcal{W}_1 , where $\omega_A = \lceil \frac{s+1}{2} \rceil$, since $k_A = s$. Thus,

$$2\omega_A \geq s + 1 > s \text{ which indicates that, } \omega_A > s - \omega_A.$$

In addition, since $m_1 \geq \omega_A$, we claim that $m_1 > s - \omega_A$. Thus, every submatrix will participate at least once within those chosen m_1 nodes, hence $|\mathcal{A}_1| = k_A$.

Next, consider the other case when $k_A > s$. Again, since we choose any arbitrary $m_1 \geq \omega_A$ nodes in \mathcal{W}_1 , we are leaving $s - m_1$ nodes in \mathcal{W}_1 . But

$$s - m_1 \leq s - \omega_A < s - \lfloor \rho \rfloor.$$

The second inequality holds since $s < k_A$. Thus, every submatrix will participate at least once within those $m_1 \geq \omega_A$ nodes, hence $|\mathcal{A}_1| = k_A$. ■

REFERENCES

- [1] A. Ramamoorthy, A. B. Das, and L. Tang, "Straggler-resistant distributed matrix computation via coding theory: Removing a bottleneck in large-scale data processing," *IEEE Sig. Proc. Mag.*, vol. 37, no. 3, pp. 136–145, 2020.
- [2] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Info. Th.*, vol. 64, no. 3, pp. 1514–1529, 2018.
- [3] A. B. Das, A. Ramamoorthy, and N. Vaswani, "Efficient and robust distributed matrix computations via convolutional coding," *IEEE Trans. Info. Th.*, vol. 67, no. 9, pp. 6266–6282, 2021.
- [4] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. of Adv. in Neur. Inf. Proc. Syst. (NeurIPS)*, 2016, pp. 2100–2108.
- [5] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Proc. of Adv. in Neur. Inf. Proc. Syst. (NeurIPS)*, 2017, pp. 4403–4413.
- [6] A. B. Das, L. Tang, and A. Ramamoorthy, " C^3LES : Codes for coded computation that leverage stragglers," in *Proc. of IEEE Info. Th. Workshop*, 2018, pp. 1–5.
- [7] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Trans. Info. Th.*, vol. 66, no. 3, pp. 1920–1933, 2020.
- [8] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. of Intl. Conf. on Mach. Learn. (ICML)*, 2017, pp. 3368–3376.
- [9] A. B. Das and A. Ramamoorthy, "A unified treatment of partial stragglers and sparse matrices in coded matrix computation," *IEEE Jour. on Sel. Area. in Info. Th.*, vol. 3, no. 2, pp. 241–256, 2022.
- [10] A. K. Pradhan, A. Heidarzadeh, and K. R. Narayanan, "Factored LT and factored raptor codes for large-scale distributed matrix multiplication," *IEEE Jour. Sel. Area. Info. Th.*, vol. 2, no. 3, pp. 893–906, 2021.
- [11] A. B. Das, A. Ramamoorthy, D. J. Love, and C. G. Brinton, "Coded matrix computations for D2D-enabled linearized federated learning," in *Proc. of IEEE Intl. Conf. on Acoustics, Speech and Sig. Proc. (ICASSP)*, 2023, pp. 1–5.
- [12] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," *IEEE Trans. Info. Th.*, vol. 67, no. 5, pp. 2758–2785, 2021.
- [13] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Random Khatri-Rao-product codes for numerically-stable distributed matrix multiplication," in *Proc. of Annu. Allerton Conf. Commun. Control Comput.*, Sep. 2019, pp. 253–259.
- [14] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *Proc. of Intl. Conf. on Mach. Learn. (ICML)*, 2018, pp. 5152–5160.
- [15] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," in *Proc. of IEEE Glob. Comm. Conf. (GLOBECOM)*, 2018, pp. 1–6.
- [16] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2722–2734, 2020.
- [17] J. Li and C. Hollanti, "Private and secure distributed matrix multiplication schemes for replicated or mds-coded servers," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 659–669, 2022.

- [18] Q. Yu and A. S. Avestimehr, "Coded computing for resilient, secure, and privacy-preserving distributed matrix multiplication," *IEEE Trans. on Comm.*, vol. 69, no. 1, pp. 59–72, 2021.
- [19] A. B. Das, A. Ramamoorthy, D. J. Love, and C. G. Brinton, "Distributed matrix computations with low-weight encodings," in *Proc. of IEEE Intl. Symp. on Info. Th.*, 2023.
- [20] A. B. Das and A. Ramamoorthy, "Coded sparse matrix computation schemes that leverage partial stragglers," *IEEE Trans. Info. Th.*, vol. 68, no. 6, pp. 4156–4181, 2022.
- [21] J. Marshall. Hall, *Combinatorial theory*. Wiley, 1986.
- [22] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Jour. of the ACM*, vol. 27, no. 4, pp. 701–717, 1980.
- [23] M. Xhemrishi, R. Bitar, and A. Wachter-Zeh, "Distributed matrix-vector multiplication with sparsity and privacy guarantees," in *Proc. of IEEE Intl. Symp. on Info. Th.*, 2022, pp. 1028–1033.
- [24] R. Ji, A. Heidarzadeh, and K. R. Narayanan, "Sparse random khatri-rao product codes for distributed matrix multiplication," in *Proc. of IEEE Info. Th. Workshop*, 2022, pp. 416–421.
- [25] SuiteSparse Matrix Collection. [Online]. Available: <https://sparse.tamu.edu/>