Coded Sparse Matrix Multiplication

Sinong Wang [1] **Jiashang Liu** [1] **Ness Shroff** [2]

Abstract

In a large-scale and distributed matrix multiplication problem $C = A^\intercal B$, where $C \in \mathbb{R}^{r\times t}$, the coded computation plays an important role to effectively deal with "stragglers" (distributed computations that may get delayed due to few slow or faulty processors). However, existing coded schemes could destroy the significant sparsity that exists in large-scale machine learning problems, and could result in much higher computation overhead, i.e., $O(rt)$ decoding time. In this paper, we develop a new coded computation strategy, we call *sparse code*, which achieves near *optimal recovery threshold*, *low computation overhead*, and *linear decoding time* $O(nnz(C))$. We implement our scheme and demonstrate the advantage of the approach over both uncoded and current fastest coded strategies.

1. Introduction

In this paper, we consider a distributed matrix multiplication problem, where we aim to compute $C = A^\intercal B$ from input matrices $A \in \mathbb{R}^{s\times r}$ and $B \in \mathbb{R}^{s\times t}$ for some integers r, s, t. This problem is the key building block in machine learning and signal processing problems, and has been used in a large variety of application areas including classification, regression, clustering and feature selection problems. Many such applications have large-scale datasets and massive computation tasks, which forces practitioners to adopt distributed computing frameworks such as Hadoop (Dean & Ghemawat, 2008) and Spark (Zaharia et al., 2010) to increase the learning speed.

Classical approaches of distributed matrix multiplication rely on dividing the input matrices equally among all available worker nodes. Each worker computes a partial result, and the master node has to collect the results from all workers to output matrix C. As a result, a major performance bottleneck is the latency in waiting for a few slow or faulty processors – called "stragglers" to finish their tasks (Dean & Barroso, 2013). To alleviate this problem, current frameworks such as Hadoop deploy various straggler detection techniques and usually *replicate* the straggling task on another available node.

Recently, forward error correction or coding techniques provide a more effective way to deal with the "straggler" in the distributed tasks (Dutta et al., 2016; Lee et al., 2017a; Tandon et al., 2017; Yu et al., 2017; Li et al., 2018; Wang et al., 2018). It creates and exploits coding redundancy in local computation to enable the matrix C recoverable from the results of partial finished workers, and can therefore alleviate some straggling workers. For example, consider a distributed system with 3 worker nodes, the coding scheme first splits the matrix A into two submatrices, i.e., $A = [A_1, A_2]$. Then each worker computes $A_1^\intercal B$, $A_2^\intercal B$ and $(A_1 + A_2)^\intercal B$. The master node can compute $A^\intercal B$ as soon as **any** 2 out of the 3 workers finish, and can overcome one straggler.

In a general setting with N workers, each input matrix A, B is divided into m, n submatrices, respectively. The *recovery threshold* is defined as the minimum number of workers that the master needs to wait for in order to compute C. The above MDS coded scheme is shown to achieve a recovery threshold $\Theta(N)$. An improved scheme proposed in (Lee et al., 2017c) referred to as the product code, can offer a recovery threshold of $\Theta(mn)$ with high probability. More recently, the work (Yu et al., 2017) designs a type of *polynomial code*. It achieves the recovery threshold of mn, which exactly matches the information theoretical lower bound. However, many problems in machine learning exhibit both extremely **large-scale** targeting data and a **sparse** structure, i.e., $nnz(A) \ll rs$, $nnz(B) \ll st$ and $nnz(C) \ll rt$. The key question that arises in this scenario is: *is coding really an efficient way to mitigate the straggler in the distributed sparse matrix multiplication problem?*

1.1. Motivation: Coding Straggler

To answer the aforementioned question, we first briefly introduce the current coded matrix multiplication schemes. In these schemes, each local worker calculates a coded version

[1]Department of ECE, The Ohio State University, Columbus, USA [2]Departments of ECE and CSE, The Ohio State University, Columbus, USA. Correspondence to: Sinong Wang <wang.7691@osu.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

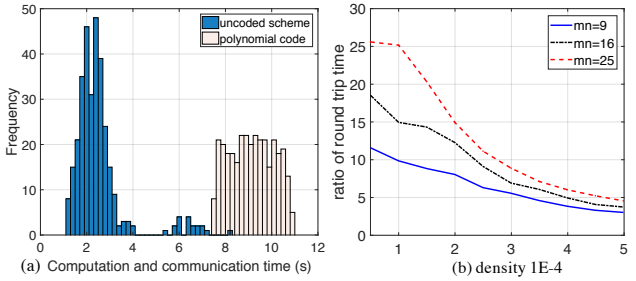

Figure 1. Measured local computation and communication time. of submatrix multiplication. For example, kth worker of the polynomial code (Yu et al., 2017) essentially calculates

$$\underbrace{\sum_{i=1}^{m} A_i^\intercal x_k^i}_{\tilde{A}_k^\intercal} \underbrace{\sum_{j=1}^{n} B_j x_k^{jm}}_{\tilde{B}_k}, \tag{1}$$

where A_i, B_j are the corresponding submatrics of the input matrices A and B, respectively, and x_k is a given integer. One can observe that, if A and B are sparse, due to the matrices additions, the density of the coded matrices $\tilde{A}_k$ and $\tilde{B}_k$ will increase at most m and n times, respectively. Therefore, the time of matrix multiplication $\tilde{A}_k^\intercal \tilde{B}_k$ will increase roughly $O(mn)$ times of the simple uncoded one $A_i^\intercal B_j$.

In the Figure 1(a), we experiment large and sparse matrix multiplication from two random Bernoulli square matrices with dimension roughly equal to 1.5×10^5 and the number of nonzero elements equal to 6×10^5. We show measurements on local computation and communication time required for $N = 16$ workers to operate the polynomial code and uncoded scheme. Our finding is that the final job completion time of the polynomial code is significantly increased compared to that of the uncoded scheme. The main reason is that the increased density of input matrix leads to the increased computation time, which further incurs the even larger data transmission and higher I/O contention. In the Figure 1(b), we generate two 10^5 random square Bernoulli matrices with different densities p. We plot the ratio of the average local computation time between the polynomial code and the uncoded scheme versus the matrix density p. It can be observed that the coded scheme requires 5 or more computation time vs the uncoded scheme, and this ratio is particularly large, i.e., $O(mn)$, when the matrix is sparse.

In certain suboptimal coded computation schemes such as MDS code, product code (Lee et al., 2017a;c) and short dot (Dutta et al., 2016), the generator matrices are generally dense, which also implies a heavy workload for each local worker. Moreover, the decoding algorithm of polynomial code and MDS type of codes is based on the fast polynomial interpolation algorithm in the finite field. Although it leads to nearly linear decoding time, i.e., $O(rt\ln^2(mn\ln(mn)))$, it still incurs extremely high cost when dealing with current large-scale and sparse data. Therefore, inspired by this phenomenon, we are interested in the following key problem:

can we find a coded matrix multiplication scheme that has small recovery threshold, low computation overhead and decoding complexity only dependent on $nnz(C)$?

1.2. Main Contribution

In this paper, we answer this question positively by designing a novel coded computation strategy, we call *sparse code*. It achieves near optimal recovery threshold $\Theta(mn)$ by exploiting the coding advantage in local computation. Moreover, such a coding scheme can exploit the sparsity of both input and output matrices, which leads to low computation load, i.e., $O(\ln(mn))$ times of uncoded scheme and, nearly linear decoding time $O(nnz(C)\ln(mn))$.

The basic idea in *sparse code* is: each worker chooses a random number of input submatrices based on a given degree distribution P; then computes a weighted linear combination $\sum_{ij} w_{ij} A_i^\intercal B_j$, where the weights w_{ij} are randomly drawn from a finite set S. When the master node receives a bunch of finished tasks such that the coefficient matrix formed by weights w_{ij} is full rank, it starts to operate a hybrid decoding algorithm between peeling decoding and Gaussian elimination to recover the resultant matrix C.

We prove the optimality of the *sparse code* by carefully designing the degree distribution P and the algebraic structure of set S. The recovery threshold of the sparse code is mainly determined by how many tasks are required such that the coefficient matrix is full rank and the hybrid decoding algorithm recovers all the results. We design a type of Wave Soliton distribution (definition is given in Section 4), and show that, under such a distribution, when $\Theta(mn)$ tasks are finished, the hybrid decoding algorithm will successfully decode all the results with decoding time $O(nnz(C)\ln(mn))$.

Moreover, we reduce the full rank analysis of the coefficient matrix to the determinant analysis of a random matrix in $\mathbb{R}^{mn\times mn}$. The state-of-the-art in this field is limited to the Bernoulli case (Tao & Vu, 2007; Bourgain et al., 2010), in which each element is identically and independently distributed random variable. However, in our proposed sparse code, the matrix is generated from a degree distribution, which leads to dependencies among the elements in the same row. To overcome this difficulty, we find a different technical path: we first utilize the Schwartz-Zeppel Lemma (Schwartz, 1980) to reduce the determinant analysis problem to the analysis of the probability that a random bipartite graph contains a perfect matching. Then we combine the combinatoric graph theory and the probabilistic method to show that when number of mn tasks are collected, the coefficient matrix is full rank with high probability.

We further utilize the above analysis to formulate an optimization problem to determine the optimal degree distribution P when mn is small. We finally implement and bench-

mark the sparse code at Ohio Supercomputer Center (Center, 1987), and empirically demonstrate its performance gain compared with the existing strategies.

2. Preliminary

We are interested in a matrix multiplication problem with two input matrices $A \in \mathbb{R}^{s\times r}$, $B \in \mathbb{R}^{s\times t}$ for some integers r, s, t. Each input matrix A and B is evenly divided along the column side into m and n submatrices, respectively.

$$A = [A_1, A_2, \ldots, A_m] \text{ and } B = [B_1, B_2, \ldots, B_n]. \quad (2)$$

Then computing the matrix C is equivalent to computing mn blocks $C_{ij} = A_i^\intercal B_j$. Let the set $W = \{C_{ij} = A_i^\intercal B_j | 1 \le i \le m, 1 \le j \le n\}$ denote these components. Given this notation, the *coded distributed matrix multiplication problem* can be described as follows: define N coded computation functions, denoted by

$$\boldsymbol{f} = (f_1, f_2, \ldots, f_N).$$

Each local function f_i is used by worker i to compute a submatrix $\tilde{C}_i \in \mathbb{R}^{\frac{r}{m}\times\frac{t}{n}} = f_i(W)$ and return it to the master node. The master node waits only for the results of the partial workers $\{\tilde{C}_i | i \in I \subseteq \{1, \ldots, N\}\}$ to recover the final output C using certain decoding functions. For any integer k, the recovery threshold $k(\boldsymbol{f})$ of a coded computation strategy $\boldsymbol{f}$ is defined as the minimum integer k such that the master node can recover matrix C from results of the any k workers. The framework is illustrated in Figure 2.

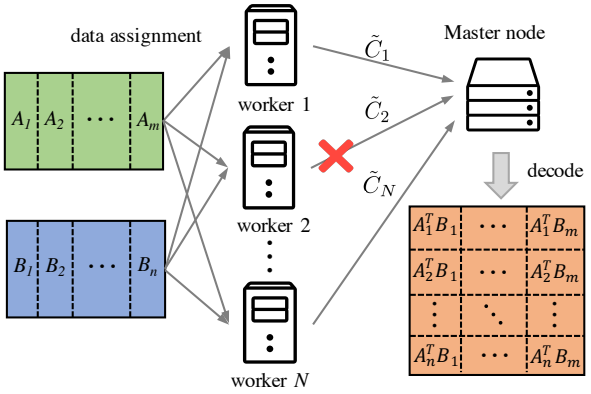

Figure 2. Framework of coded distributed matrix multiplication.

2.1. Main Results

The main result of this paper is the design of a new coded computation scheme, we call the *sparse code*, that has the following performance.

Theorem 1. *The sparse code achieves a recovery threshold $\Theta(mn)$ with high probability, while allowing nearly linear decoding time $O(nnz(C)\ln(mn))$ at the master node.*

As shown in TABLE 1, compared to the state of the art, the sparse code provides order-wise improvement in terms of the recovery threshold, computation overhead and decoding complexity. Specifically, the decoding time of MDS code (Lee et al., 2017a), product code (Lee et al., 2017c), LDPC code and polynomial code (Yu et al., 2017) is $O(rt\ln^2(mn\ln(mn)))$, which is dependent on the dimension of the output matrix. Instead, the proposed sparse code actually exhibits a decoding complexity that is nearly linear time in number of nonzero elements of the output matrix C, which is extremely less than the product of dimension. Although the decoding complexity of sparse MDS code is linear to $nnz(C)$, it is also dependent on the mn. To the best of our knowledge, this is the first coded distributed matrix multiplication scheme with complexity independent of the dimension.

Table 1. Comparison of Existing Coding Schemes

scheme	recovery threshold	computation overhead[1]	decoding time
MDS	$\Theta(N)$	$\Theta(mn)$	$\tilde{O}(rt)$[2]
sparse MDS	$\Theta^*(mn)$[2]	$\Theta(\ln(mn))$	$\tilde{O}(mn \cdot nnz(C))$
product code	$\Theta^*(mn)$	$\Theta(mn)$	$\tilde{O}(rt)$
LDPC code	$\Theta^*(mn)$	$\Theta(\ln(mn))$	$\tilde{O}(rt)$
polynomial	mn	mn	$\tilde{O}(rt)$
our scheme	$\Theta^*(mn)$	$\Theta(\ln(mn))$	$\tilde{O}(nnz(C))$

[1] Computation overhead is the time of local computation over uncoded scheme.
[2] $\tilde{O}(\cdot)$ omits the logarithmic terms and $O^*(\cdot)$ refers the high probability result.

Regarding the recovery threshold, the existing work (Yu et al., 2017) has applied a cut-set type argument to show that the minimum recovery threshold of any scheme is

$$K^* = \min_{\boldsymbol{f}} k(\boldsymbol{f}) = mn. \quad (3)$$

The proposed sparse code matches this lower bound with a constant gap and high probability.

3. Sparse Codes

In this section, we first demonstrate the main idea of the sparse code through a motivating example. We then formally describe the construction of the general sparse code and its decoding algorithm.

3.1. Motivating Example

Consider a distributed matrix multiplication task $C = A^\intercal B$ using $N = 6$ workers. Let $m = 2$ and $n = 2$ and each input matrix A and B be evenly divided as

$$A = [A_1, A_2] \quad \text{and} \quad B = [B_1, B_2].$$

Then computing the matrix C is equivalent to computing following 4 blocks.

$$C = A^\intercal B = \begin{bmatrix} A_1^\intercal B_1 & A_1^\intercal B_2 \\ A_2^\intercal B_1 & A_2^\intercal B_2 \end{bmatrix}$$

We design a coded computation strategy via the following procedure: each worker i locally computes a weighted sum

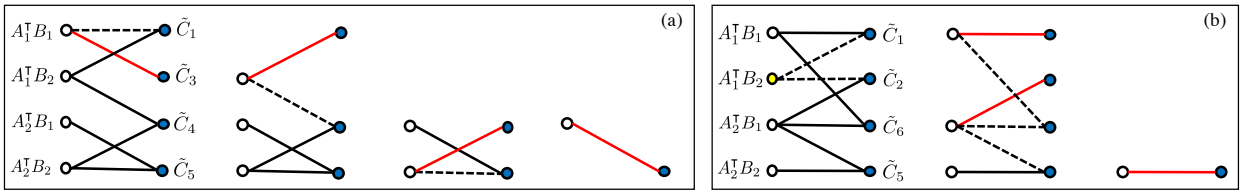

Figure 3. Example of the hybrid peeling and Gaussian decoding process of sparse code.

of four components in matrix C.

$$\tilde{C}_i = w_1^i A_1^\intercal B_1 + w_2^i A_1^\intercal B_2 + w_3^i A_2^\intercal B_1 + w_4^i A_2^\intercal B_2,$$

Each weight w_j^i is independently and identically distributed Bernoulli random variable with parameter p. For example, Let $p = 1/3$, then on average, $2/3$ of these weights are equal to 0. We randomly generate the following $N = 6$ local computation tasks.

$$\tilde{C}_1 = A_1^\intercal B_1 + A_1^\intercal B_2, \quad \tilde{C}_2 = A_1^\intercal B_2 + A_2^\intercal B_1$$
$$\tilde{C}_3 = A_1^\intercal B_1, \quad \tilde{C}_4 = A_1^\intercal B_2 + A_2^\intercal B_2$$
$$\tilde{C}_5 = A_2^\intercal B_1 + A_2^\intercal B_2, \quad \tilde{C}_6 = A_1^\intercal B_1 + A_2^\intercal B_1$$

Suppose that both the 2rd and 6th workers are stragglers and the master node has collected the results from nodes $\{1, 3, 4, 5\}$. According to the designed computation strategy, we have following group of linear systems.

$$\begin{bmatrix} \tilde{C}_1 \\ \tilde{C}_3 \\ \tilde{C}_4 \\ \tilde{C}_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} A_1^\intercal B_1 \\ A_1^\intercal B_2 \\ A_2^\intercal B_1 \\ A_2^\intercal B_2 \end{bmatrix}$$

One can easily check that the above coefficient matrix is full rank. Therefore, one straightforward way to recover C is to solve $rt/4$ linear systems, which proves decodability. However, the complexity of this decoding algorithm is expensive, i.e., $O(rt)$ in this case.

Interestingly, we can use a type of *peeling algorithm* to recover the matrix C with only three sparse matrix additions: first, we can straightforwardly recover the block $A_1^\intercal B_1$ from worker 3. Then we can use the result of worker 1 to recover block $A_1^\intercal B_2 = \tilde{C}_1 - A_1^\intercal B_1$. Further, we can use the results of worker 4 to recover block $A_2^\intercal B_2 = \tilde{C}_4 - A_1^\intercal B_2$ and use the results of worker 5 to obtain block $A_2^\intercal B_1 = \tilde{C}_5 - A_2^\intercal B_2$. Actually, the above peeling decoding algorithm can be viewed as an edge-removal process in a bipartite graph. We construct a bipartite graph with one partition being the original blocks W and the other partition being the finished coded computation tasks $\{\tilde{C}_i\}$. Two nodes are connected if such computation task contains that block. As shown in the Figure 3(a), in each iteration, we find a *ripple* (degree one node) in the right that can be used to recover one node of left. We remove the adjacent edges of that left node, which might produce some new ripples in the right. Then we iterate this process until we decode all blocks.

Based on the above graphical illustration, the key point of successful decoding is the existence of the ripple during the edge removal process. Clearly, this is not always true from the design of our coding scheme and the uncertainty in the cloud. For example, if both the 3rd and 4th workers are stragglers and the master node has collected the results from node $\{1, 2, 5, 6\}$, even though the coefficient matrix is full rank, there exists no ripple in the graph. To avoid this problem, we can randomly pick one block and recover it through a linear combination of the collected results, then use this block to continue the decoding process. This particular linear combination can be determined by solving a linear system. Suppose that we choose to recover $A_1^\intercal B_2$, then we can recover it via the following linear combination.

$$A_1^\intercal B_2 = \frac{1}{2}\tilde{C}_1 + \frac{1}{2}\tilde{C}_2 - \frac{1}{2}\tilde{C}_6.$$

As illustrated in the Figure 3(b), we can recover the rest of the blocks using the same peeling decoding process. The above decoding algorithm only involves simple matrix additions and the total decoding time is $O(nnz(C))$.

3.2. General Sparse Code

Now we present the construction and decoding of the sparse code in a general setting. We first evenly divide the input matrices A and B along the column side into m and n submatrices, as defined in (2). Then we define a set S that contains m^2n^2 distinct elements except zero element. One simplest example of S is $[m^2n^2] \triangleq \{1, 2, \ldots, m^2n^2\}$. Under this setting, we define the following class of coded computation strategies.

Definition 1. (Sparse Code) *Given the parameter $P \in \mathbb{R}^{mn}$ and set S, we define the $(P, S)-$sparse code as: for each worker $k \in [N]$, compute*

$$\tilde{C}_k = f_k(W) = \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}^k A_i^\intercal B_j. \tag{4}$$

Here the parameter $P = [p_1, p_2, \ldots, p_{mn}]$ is the degree distribution, where the p_l is the probability that there exists number of l nonzero weights w_{ij}^k in each worker k. The value of each nonzero weight w_{ij}^k is picked from set S independently and uniformly at random.

Without loss of generality, suppose that the master node collects results from the first K workers with $K \leq N$.

Given the above coding scheme, we have

$$\begin{bmatrix}\tilde{C}_1\\ \tilde{C}_2\\ \vdots\\ \tilde{C}_K\end{bmatrix} = \begin{bmatrix}w_{11}^1 & w_{12}^1 & \cdots & w_{mn}^1\\ w_{11}^2 & w_{12}^2 & \cdots & w_{mn}^2\\ \vdots & \vdots & \ddots & \vdots\\ w_{11}^K & w_{12}^K & \cdots & w_{mn}^K\end{bmatrix} \cdot \begin{bmatrix}A_1^\intercal B_1\\ A_1^\intercal B_2\\ \vdots\\ A_m^\intercal B_n\end{bmatrix}.$$

We use $M \in \mathbb{R}^{K\times mn}$ to represent the above coefficient matrix. To guarantee decodability, the master node should collect results from enough number of workers such that the coefficient matrix M is of column full rank. Then the master node goes through a **peeling decoding** process: it first finds a ripple worker to recover one block. Then for each collected results, it subtracts this block if the computation task contains this block. If there exists no ripple in our peeling decoding process, we go to **rooting step**: randomly pick a particular block $A_i^\intercal B_j$. The following lemma shows that we can recover this block via a linear combination of the results $\{\tilde{C}_k\}_{k=1}^K$.

Lemma 1. (rooting step) *If rank$(M) = mn$, for any $k_0 \in \{1, 2, \ldots, mn\}$, we can recover a particular block $A_i^\intercal B_j$ with column index k_0 in matrix M via the following linear combination.*

$$A_i^\intercal B_j = \sum\nolimits_{k=1}^{K} u_k \tilde{C}_k. \tag{5}$$

The vector $u = [u_1, \ldots, u_K]$ can be determined by solving $M^T u = e_{k_0}$, where $e_{k_0} \in \mathbb{R}^K$ is a unit vector with unique 1 locating at the index k_0.

The basic intuition is to find a linear combination of row vectors of matrix M such that the row vectors eliminate all other blocks except the particular block $A_i^\intercal B_j$. The whole procedure is listed in Algorithm 1.

Here we conduct some analysis of the complexity of Algorithm 1. During each iteration, the complexity of operation $\tilde{C}_k = \tilde{C}_k - M_{kk_0} A_i^\intercal B_j$ is $O(nnz(A_i^\intercal B_j))$. Suppose that the number of average nonzero elements in each row of coefficient matrix M is α. Then each block $A_i^\intercal B_j$ will be used $O(\alpha K/mn)$ times in average. Further, suppose that there exists number of c blocks requiring the rooting step (5) to recover, the complexity in each step is $O(\sum_k nnz(\tilde{C}_k))$. On average, each coding block $\tilde{C}_k$ is equal to the sum of $O(\alpha)$ original blocks. Therefore, the complexity of Algorithm 1 is

$$\begin{aligned}&O\left(\frac{\alpha K}{mn}\sum\nolimits_{i,j} nnz(A_i^\intercal B_j)\right) + O\left(c\sum\nolimits_k nnz(\tilde{C}_k)\right)\\ =&O\left((c+1)\alpha K/mn \cdot nnz(C)\right).\end{aligned} \tag{6}$$

We can observe that the decoding time is linear in the density of matrix M, the recovery threshold K and the number of rooting steps (5). In the next section, we will show that, under a good choice of degree distribution P and set S, we can achieve the result in Theorem 1.

Algorithm 1 Sparse code (master node's protocol)

repeat
  The master node assign the coded computation tasks according to Definition 1.
until the master node collects results with rank$(M)= mn$ and K is larger than a given threshold.
repeat
  Find a row $M_{k'}$ in matrix M with $\|M_{k'}\|_0 = 1$.
  if such row does not exist **then**
    Randomly pick a $k_0 \in \{1, \ldots, mn\}$ and recover corresponding block $A_i^\intercal B_j$ by (5).
  else
    Recover the block $A_i^\intercal B_j$ from $\tilde{C}_{k'}$.
  end if
  Suppose that the column index of the recovered block $A_i^\intercal B_j$ in matrix M is k_0.
  for each computation results $\tilde{C}_k$ **do**
    if M_{kk_0} is nonzero **then**
      $\tilde{C}_k = \tilde{C}_k - M_{kk_0} A_i^\intercal B_j$ and set $M_{kk_0} = 0$.
    end if
  end for
until every block of matrix C is recovered.

4. Theoretical Analysis

As discussed in the preceding section, to reduce the decoding complexity, it is good to make the coefficient matrix M as sparse as possible. However, the lower density will require that the master node collects a larger number of workers to enable the full rank of matrix M. For example, in the extreme case, if we randomly assign one nonzero element in each row of M. The analysis of the classical balls and bins process implies that, when $K = O(mn \ln(mn))$, the matrix M is full rank, which is far from the optimal recovery threshold. On the other hand, the polynomial code (Yu et al., 2017) achieves the optimal recovery threshold. Nonetheless, it exhibits the densest matrix M, i.e., $K \times mn$ nonzero elements, which significantly increases the local computation, communication and final decoding time.

In this section, we will design the sparse code between these two extremes. This code has near optimal recovery threshold $K = \Theta(mn)$ and constant number of rooting steps (5) with high probability and extremely sparse matrix M with $\alpha = \Theta(\ln(mn))$ nonzero elements in each row. The main idea is to choose the following degree distribution.

Definition 2. (Wave Soliton distribution) *The Wave Soliton distribution $P_w = [p_1, p_2, \ldots, p_{mn}]$ is defined as follows.*

$$p_k = \begin{cases}\frac{\tau}{mn}, k = 1; \frac{\tau}{70}, k = 2\\ \frac{\tau}{k(k-1)}, 3 \le k \le mn\end{cases}. \tag{7}$$

The parameter $\tau = 35/18$ is the normalizing factor.

The above degree distribution is modified from the Soliton distribution (Luby, 2002). In particular, we cap the original Soliton distribution at the maximum degree mn, and remove a constant weight from degree 2 to other larger degrees. It can be observed that the recovery threshold K of the proposed sparse code depends on two factors: (i) the full rank of coefficient matrix M; (ii) the successful decoding of peeling algorithm with constant number of rooting steps.

4.1. Full Rank Probability

Our first main result is to show that when $K = mn$, the coefficient matrix M is full rank with high probability. Suppose that $K = mn$, we can regard the formation of the matrix M via the following random graph model.

Definition 3. (Random balanced bipartite graph) *Let $G(V_1, V_2, P)$ be a random blanced bipartite graph, in which $|V_1| = |V_2| = mn$. Each node $v \in V_2$ independently and randomly connects to l nodes in partition V_1 with probability p_l.*

Define an Edmonds matrix $M(x) \in \mathbb{R}^{mn\times mn}$ of graph $G(V_1, V_2, P)$ with $[M(x)]_{ij} = x_{ij}$ if vertices $v_i \in V_1$, $v_j \in V_2$ are connected, and $[M(x)]_{ij} = 0$, otherwise. The coefficient matrix M can be obtained by assigning each x_{ij} a value from S independently and uniformly at random. Then the probability that matrix M is full rank is equal to the probability that the determinant of the Edmonds matrix $M(x)$ is nonzero at the assigning values x_{ij}. The following technical lemma (Schwartz, 1980) provides a simple lower bound of the such an event.

Lemma 2. (Schwartz-Zeppel Lemma) *Let $f(x_1, \ldots, x_N)$ be a nonzero polynomial with degree d. Let S be a finite set in $\mathbb{R}$ with $|S| = d^2$. If we assign each variable a value from S independently and uniformly at random, then*

$$\mathbb{P}(f(x_1, x_2, \ldots, x_N) \neq 0) \geq 1 - d^{-1}. \tag{8}$$

A classic result in graph theory is that *a balanced bipartite graph contains a perfect matching if and only if the determinant of Edmonds matrix, i.e., $|M(x)|$, is a nonzero polynomial*. Combining this result with Schwartz-Zeppel Lemma, we can finally reduce the analysis of the full rank probability of the coefficient matrix M to the probability that the random graph $G(V_1, V_2, P_w)$ contains a perfect matching.

$$\mathbb{P}(|M| \neq 0) = \underbrace{\mathbb{P}(|M| \neq 0 \big| |M(x)| \not\equiv 0)}_{\text{S-Z Lemma: } \geq 1-1/mn} \cdot \underbrace{\mathbb{P}(|M(x)| \not\equiv 0)}_{\text{perfect matching}}$$

Formally, we have the following result about the existence of the perfect matching in the above defined random bipartite graph.

Theorem 2. (Existence of perfect matching) *If the graph $G(V_1, V_2, P)$ is generated under the Wave Soliton distribution (7), then there exists a constant $c > 0$ such that*

$$\mathbb{P}(G \text{ contains a perfect matching}) > 1 - c(mn)^{-0.94}.$$

Proof. Here we sketch the proof. Details can be seen in the supplementary material. The basic technique is to utilize Hall's theorem to show that such a probability is lower bounded by the probability that G does not contain a structure $S \subset V_1$ or $S \subset V_2$ such that $|S| > |N(S)|$, where $N(S)$ is the neighboring set of S. We show that, if $S \subset V_1$, the probability that S exists is upper bounded by

$$\sum_{s=\Theta(1)} \frac{1}{(mn)^{0.94s}} + \sum_{s=\Omega(1)}^{s=o(mn)} \left(\frac{s}{mn}\right)^{0.94s} + \sum_{s=\Theta(mn)} c_1^{mn}.$$

If $S \subset V_2$, the probability that S exists is upper bounded by

$$\sum_{s=\Theta(1)} \frac{1}{mn} + \sum_{s=o(mn)} \frac{sc_2^s}{mn} + \sum_{s=\Theta(mn)} c_3^{mn}.$$

where the constants c_1, c_2, c_3 are strictly less than 1. Combining these results together, gives us Theorem 2. □

Analyzing the existence of perfect matching in a random bipartite graph has been developed since (Erdos & Renyi, 1964). However, existing analysis is limited to the independent generation model. For example, the Erdos-Renyi model assumes each edge exists independently with probability p. The $\kappa-$out model (Walkup, 1980) assumes each vertex $v \in V_1$ independently and randomly chooses κ neighbors in V_2. The minimum degree model (Frieze & Pittel, 2004) assumes each vertex has a minimum degree and edges are uniformly distributed among all allowable classes. There exists no work in analyzing such a probability in a random bipartite graph generated by a given degree distribution. In this case, one technical difficulty is that each node in the partition V_1 is dependent. All analysis should be carried from the nodes of the right partition, which exhibits an intrinsic complicated statistical model.

4.2. Optimality of Recovery Threshold

We now focus on quantitatively analyzing the impact of the recovery threshold K on the peeling decoding process and the number of rooting steps (5). Intuitively, a larger K implies a larger number of ripples, which leads to higher successful peeling decoding probability and therefore less number of rooting steps. The key question is: how large must K be such that all mn blocks are recovered with only constant number of rooting steps. To answer this question, we first define a distribution generation function of P_w as

$$\Omega_w(x) = \frac{\tau}{mn}x + \frac{\tau}{70}x^2 + \tau\sum_{k=3}^{mn}\frac{x^k}{k(k-1)}. \tag{9}$$

The following technical lemma is useful in our analysis.

Lemma 3. *If the degree distribution $\Omega_w(x)$ and recovery threshold K satisfy*

$$[1-\Omega'_w(1-x)/mn]^{K-1} \leq x, \text{ for } x \in [b/mn, 1], \quad (10)$$

then the peeling decoding process in Algorithm 1 can recover $mn-b$ blocks with probability at least $1-e^{-cmn}$, where b, c are constants.

Lemma 3 is tailored from applying a martingale argument to the peeling decoding process (Luby et al., 2001). This result provides a quantitative recovery condition on the degree generation function. It remains to be shown that the proposed Wave Soliton distribution (9) satisfies the above inequality with a specific choice of K.

Theorem 3. (Recovery threshold) *Given the sparse code with parameter$(P_w, [m^2n^2])$, if $K = \Theta(mn)$, then there exists a constant c such that with probability at least $1-e^{-cmn}$, Algorithm 1 is sufficient to recover all mn blocks with $\Theta(1)$ blocks recovering from rooting step (5).*

Combining the results of Theorem 2 and Theorem 3, we conclude that the recovery threshold of sparse code $(P_w, [m^2n^2])$ is $\Theta(mn)$ with high probability. Moreover, since the average degree of Wave Soliton Distribution is $O(\ln(mn))$, combining these results with (6), the complexity of Algorithm 1 is therefore $O(nnz(C)\ln(mn))$.

Remark 1. *Although the recovery threshold of the proposed scheme exhibits a constant gap to the information theoretical lower bound, the practical performance is very close to such a bound. This mainly comes from the pessimistic estimation in Theorem 3. As illustrated in Figure 4, we generate the sparse code under Robust Soliton distribution, and plot the average recovery threshold versus the number of blocks mn. It can be observed that the overhead of proposed sparse code is less than 15%.*

Remark 2. *Existing codes such as Tornado code (Luby, 1998) and LT code (Luby, 2002) also utilize the peeling decoding algorithm and can provide a recovery threshold $\Theta(mn)$. However, they exhibit a large constant, especially when mn is less than 10^3. Figure 4 compares the practical recovery threshold among these codes. We can see that our proposed sparse code results in a much lower recovery threshold. Moreover, the intrinsic cascading structure of these codes will also destroy the sparsity of input matrices.*

4.3. Optimal Design of Sparse Code

The proposed Wave Soliton distribution (7) is asymptotically optimal, however, it is far from optimal in practice when m, n is small. The analysis of the full rank probability and the decoding process relies on an asymptotic argument to enable upper bounds of error probability. Such bounds are far from tight when m, n are small. In this subsection, we focus on determining the optimal degree distribution based on our analysis in Section 4.1 and 4.2. Formally, we can formulate the following optimization problem.

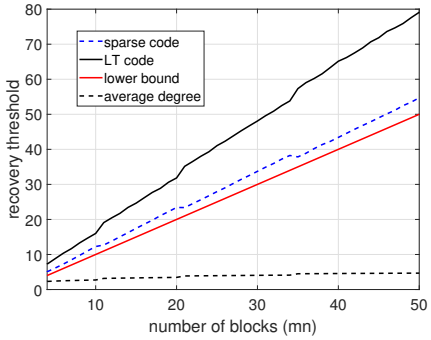

Figure 4. Recovery threshold versus the number of blocks mn.

$$\begin{aligned} \min \quad & \sum_{k=1}^{mn} k p_k \qquad (11) \\ s.t. \quad & \mathbb{P}(M \text{ is full rank}) > p_c, \\ & \left[1-\tfrac{\Omega'_w(x)}{mn}\right]^{mn+c} \leq 1-x-c_0\sqrt{\tfrac{1-x}{mn}}, \\ & x \in [0, 1-b/mn], [p_k] \in \Delta_{mn}, \end{aligned}$$

The objective is to minimize the average degree, namely, to minimize the computation and communication overhead at each worker. The first constraint represents that the probability of full rank is at least p_c. Since it is difficult to obtain the exact form of such a probability, we can use the analysis in Section 4.1 to replace this condition by requiring the probability that the balanced bipartite graph $G(V_1, V_2, P)$ contains a perfect matching is larger than a given threshold, which can be calculated exactly. The second inequality represents the decodability condition that when $K = mn+c+1$ results are received, $mn-b$ blocks are recovered through the peeling decoding process and b blocks are recovered from the rooting step (5). This condition is modified from (10) by adding an additional term, which is useful in increasing the expected ripple size (Shokrollahi, 2006). By discretizing the interval $[0, 1-b/mn]$ and requiring the above inequality to hold on the discretization points, we obtain a set of linear inequalities constraints. Details regarding the exact form of the above optimization model and solutions are provided in the supplementary material.

5. Experimental Results

In this section, we present experimental results at Ohio Supercomputer Center (Center, 1987). We compare our proposed coding scheme against the following schemes: (i) **uncoded scheme**: the input matrices are divided uniformly across all workers and the master waits for all workers to send their results; (ii) **sparse MDS code** (Lee et al., 2017b): the generator matrix is a sparse random Bernoulli matrix with average computation overhead $\Theta(\ln(mn))$,

Table 2. Timing Results for Different Sparse Matrix Multiplications (in sec)

Data	uncoded	LT code	sparse MDS code	product code	polynomial code	sparse code
square	6.81	3.91	6.42	6.11	18.44	**2.17**
tall	7.03	2.69	6.25	5.50	18.51	**2.04**
fat	6.49	1.36	3.89	3.08	9.02	**1.22**
amazon-08 / web-google	15.35	17.59	46.26	38.61	161.6	**11.01**
cont1 / cont11	7.05	5.54	9.32	14.66	61.47	**3.23**
cit-patents / patents	22.10	29.15	69.86	56.59	1592	**21.07**
hugetrace-00 / -01	18.06	21.76	51.15	37.36	951.3	**14.16**

recovery threshold of $\Theta(mn)$ and decoding complexity $\tilde{O}(mn \cdot nnz(C))$. (iii) **product code** (Lee et al., 2017c): two-layer MDS code that can achieves the probabilistic recovery threshold of $\Theta(mn)$ and decoding complexity $\tilde{O}(rt)$. We use the above sparse MDS code to ensemble the product code to reduce the computation overhead. (iv) **polynomial code** (Yu et al., 2017): coded matrix multiplication scheme with optimum recovery threshold; (v) **LT code** (Luby, 2002): rateless code widely used in broadcast communication. It has low decoding complexity due to the peeling decoding algorithm. To simulate straggler effects in large-scale system, we randomly pick number of s workers that are running a background thread which increases the computation time.

We implement all methods in python using MPI4py. To simplify the simulation, we fix the number of workers N and randomly generate a coefficient matrix $M \in \mathbb{R}^{N \times mn}$ under given degree distribution offline such that it can resist one straggler. Then, each worker loads a certain number of partitions of input matrices according to the coefficient matrix M. In the computation stage, each worker computes the product of their assigned submatrices and returns the results using `Isend()`. Then the master node actively listens to the responses from each worker via `Irecv()`, and uses `Waitany()` to keep polling for the earliest finished tasks. Upon receiving enough results, the master stops listening and starts decoding the results.

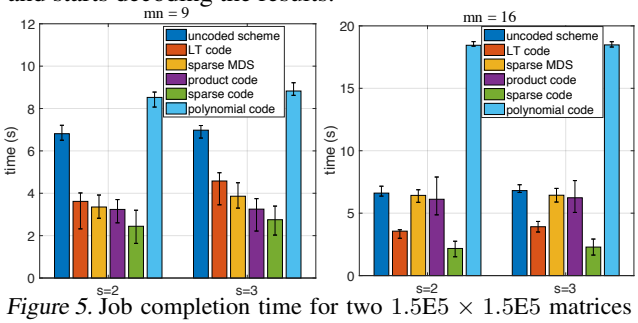

Figure 5. Job completion time for two 1.5E5 × 1.5E5 matrices with 6E5 nonzero elements.

We first generate two random Bernoulli sparse matrices with $r = s = t = 150000$ and 600000 nonzero elements. Figure. 5 reports the job completion time under $m = n = 3$, $m = n = 4$ and number of stragglers $s = 2, 3$, based on 20 experimental runs. It can be observed that our proposed sparse code requires the minimum time, and outperforms LT code (in 20-30% the time), sparse MDS code and product code (in 30-50% the time) and polynomial code (in 15-20% the time). The uncoded scheme is faster than the polynomial code. The main reason is that, due to the increased number of nonzero elements of coded matrices, the per-worker computation time for these codes is increased. Moreover, the data transmission time is also greatly increased, which leads to additional I/O contention at the master node. We further compare our proposed sparse code with the existing schemes from the point of view of the time required to communicate inputs to each worker, compute the matrix multiplication in parallel, fetch the required outputs, and decode. Results can be seen in the supplementary material.

We finally compare our scheme with these schemes for other type of matrices and larger matrices. The data statistics can be seen in the supplementary material. The first three data sets `square`, `tall` and `fat` are randomly generated square, fat and tall matrices. We also consider 8 sparse matrices from real data sets (Davis & Hu, 2011). We evenly divide each input matrices into $m = n = 4$ submatrices and number of stragglers is equal to 2. We match the column dimension of A and row dimension of B using the smaller one. The timing results are results averaged over 20 experimental runs. Among all experiments, we can observe in Table 2 that our proposed sparse code speeds up $1 - 3\times$ of uncoded scheme and outperforms the existing codes, with the effects being more pronounced for the real data sets. The job completion of the LT code, random sparse, product code is smaller than uncoded scheme in `square`, `tall` and `fat` matrix and larger than uncoded scheme in those real data sets.

6. Conclusion

In this paper, we proposed a new coded matrix multiplication scheme, which achieves near optimal recovery threshold, low computation overhead, and decoding time linear in the number of nonzero elements. Both theoretical and simulation results exhibit order-wise improvement of the proposed sparse code compared with the existing schemes. In the future, we will extend this idea to the case of higher-dimensional linear operations such as tensor operations.

References

Bourgain, Jean, Vu, Van H, and Wood, Philip Matchett. On the singularity probability of discrete random matrices. *Journal of Functional Analysis*, 258(2):559–603, 2010.

Center, Ohio Supercomputer. Ohio supercomputer center. http://osc.edu/ark:/19495/f5s1ph73, 1987.

Davis, Timothy A and Hu, Yifan. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

Dean, Jeffrey and Barroso, Luiz André. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

Dean, Jeffrey and Ghemawat, Sanjay. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Dutta, Sanghamitra, Cadambe, Viveck, and Grover, Pulkit. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pp. 2100–2108, 2016.

Erdos, Paul and Renyi, Alfred. On random matrices. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 8(455-461):1964, 1964.

Frieze, Alan and Pittel, Boris. Perfect matchings in random graphs with prescribed minimal degree. In *Mathematics and Computer Science III*, pp. 95–132. Springer, 2004.

Lee, Kangwook, Lam, Maximilian, Pedarsani, Ramtin, Papailiopoulos, Dimitris, and Ramchandran, Kannan. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 2017a.

Lee, Kangwook, Pedarsani, Ramtin, Papailiopoulos, Dimitris, and Ramchandran, Kannan. Coded computation for multicore setups. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2413–2417. IEEE, 2017b.

Lee, Kangwook, Suh, Changho, and Ramchandran, Kannan. High-dimensional coded matrix multiplication. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2418–2422. IEEE, 2017c.

Li, Songze, Maddah-Ali, Mohammad Ali, Yu, Qian, and Avestimehr, A Salman. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1): 109–128, 2018.

Luby, Michael. Tornado codes: Practical erasure codes based on random irregular graphs. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pp. 171–171. Springer, 1998.

Luby, Michael. Lt codes. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pp. 271–280. IEEE, 2002.

Luby, Michael G, Mitzenmacher, Michael, Shokrollahi, Mohammad Amin, and Spielman, Daniel A. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

Schwartz, Jacob T. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

Shokrollahi, Amin. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.

Tandon, Rashish, Lei, Qi, Dimakis, Alexandros G, and Karampatziakis, Nikos. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pp. 3368–3376, 2017.

Tao, Terence and Vu, Van. On the singularity probability of random bernoulli matrices. *Journal of the American Mathematical Society*, 20(3):603–628, 2007.

Walkup, David W. Matchings in random regular bipartite digraphs. *Discrete Mathematics*, 31(1):59–64, 1980.

Wang, Sinong, Liu, Jiashang, Shroff, Ness, and Yang, Pengyu. Fundamental limits of coded linear transform. *arXiv preprint arXiv:1804.09791*, 2018.

Yu, Qian, Maddah-Ali, Mohammad, and Avestimehr, Salman. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Advances in Neural Information Processing Systems*, pp. 4406–4416, 2017.

Zaharia, Matei, Chowdhury, Mosharaf, Franklin, Michael J, Shenker, Scott, and Stoica, Ion. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.