Computation Efficient Coded Linear Transform

Sinong Wang[1], **Jiashang Liu**[1], **Ness Shroff**[1,2] **Pengyu Yang**[3]
[1]Department of Electrical and Computer Engineering, The Ohio State University
[2]Department of Computer Science and Engineering, The Ohio State University
[3]Department of Mathematics, The Ohio State University

Abstract

In large-scale distributed linear transform problems, coded computation plays an important role to reduce the delay caused by slow machines. However, existing coded schemes could end up destroying the significant sparsity that exists in large-scale machine learning problems, and in turn increase the computational delay. In this paper, we propose a coded computation strategy, referred to as diagonal code, that achieves the optimum recovery threshold and the optimum computation load. Furthermore, by leveraging the ideas from random proposal graph theory, we design a random code that achieves a constant computation load, which significantly outperforms the existing best known result. We apply our schemes to the distributed gradient descent problem and demonstrate the advantage of the approach over current fastest coded schemes.

1 Introduction

In this paper, we consider a distributed linear transformation problem, where we aim to compute $\mathbf{y} = \mathbf{A}\mathbf{x}$ from input matrix $\mathbf{A} \in \mathbb{R}^{r\times t}$ and vector $\mathbf{x} \in \mathbb{R}^t$. 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. Optimization-based training algorithms such as gradient descent in regression and classification problems and backpropagation algorithms in deep neural networks, require the computation of large linear transforms of high-dimensional data. It is also the critical step in the dimensionality reduction techniques such as principal component analysis and linear discriminant analysis. Many such applications have large-scale datasets and massive computational tasks that force practitioners to adopt distributed computing frameworks such as Hadoop [Dean and Ghemawat, 2008] and Spark [Zaharia et al., 2010] to increase the learning speed.

Classical approaches of distributed linear transforms rely on dividing the input matrix $\mathbf{A}$ equally among all available worker nodes, and the master node has to collect the results from all workers to output $\mathbf{y}$. As a result, a major performance bottleneck is the latency incurred in waiting for a few slow or faulty processors – called "stragglers" to finish their tasks [Dean et al., 2012, Dean and Barroso, 2013]. Recently, forward error correction and other coding techniques have shown to be effective in dealing with the stragglers in distributed computation tasks [Dutta et al., 2016, Lee et al., 2017a,b, Tandon et al., 2017, Yu et al., 2017, Karakus et al., 2017, Yang et al., 2017, Wang et al.]. By exploiting coding redundancy, the vector $\mathbf{y}$ is recoverable even if all workers have not finished their computations, thus reducing the delay caused by straggler nodes. For example, consider a distributed system with 3 worker nodes, the coding scheme first splits the matrix A into two submatrices, i.e., $\mathbf{A} = [\mathbf{A}_1; \mathbf{A}_2]$. Then each worker computes $\mathbf{A}_1\mathbf{x}$, $\mathbf{A}_2\mathbf{x}$ and $(\mathbf{A}_1 + \mathbf{A}_2)\mathbf{x}$. The master node can compute $\mathbf{A}\mathbf{x}$ as soon as **any** 2 out of the 3 workers finish, and can overcome one straggler (slow worker).

In a general setting with m workers, the input matrix $\mathbf{A}$ is evenly divided into n ($n \leq m$) submatrices $\mathbf{A}_i \in \mathbb{R}^{\frac{r}{n}\times t}$ along the row side. Each worker computes a partially coded linear transform and returns the result to the master node. Given a computation strategy, the *recovery threshold* is defined as the minimum number of workers that the master needs to wait for in order to compute $\mathbf{A}\mathbf{x}$. The above MDS coded scheme is shown to achieve a recovery threshold $\Theta(m)$ [Lee et al., 2017a]. An improved scheme proposed in [Dutta et al., 2016], referred to as the *short dot code*, can offer a larger recovery threshold $\Theta(m(1+\epsilon))$ but with a constant reduction of the computation load by imposing some sparsity of the encoded submatrices. More recently, the work in [Yu et al., 2017] designs a type of *polynomial code*, which achieves the information-theoretical optimal recovery threshold n. However, many problems in machine learning exhibit both extremely **large-scale** and **sparse** targeting

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

Table 1: Comparison of Existing Schemes in Coded Computation.

Scheme	MDS code	Short-dot code	Polynomial code	s-MDS code	Sparse code	s-diagonal code	(d_1, d_2)-cross code
Recovery threshold	$\Theta(m)$	$\Theta(m(1+\epsilon))$	n	n^*	$\Theta(n)^*$	n	n^*
Computation load/m	$O(n)$	$O(n(1-\epsilon))$	n	$\Theta(\log(n))^*$	$\Theta(\log(n))^*$	$O(s)$	$\Theta(1)^*$

* The result holds with high probability, i.e., $1 - e^{-cn}$.

data, i.e., $nnz(\mathbf{A}) \ll rt$. The key question that arises in this scenario is: *is coding really an efficient way to mitigate the straggler in the distributed sparse linear transformation problem?*

1.1 Motivation: Coded Computation Overhead

To answer the aforementioned question, we use the PageRank problem [Page et al., 1999] and existing polynomial code [Yu et al., 2017] as an example. This problem aims to measure the importance score of the nodes on a graph, which is typically solved by the following power-iteration,

$$\mathbf{x}_t = c\mathbf{r} + (1-c)\mathbf{A}\mathbf{x}_{t-1}. \tag{1}$$

where $c = 0.15$ is a constant and $\mathbf{A}$ is the graph adjacency matrix. In practical applications such as web searching, the graph adjacent matrix $\mathbf{A}$ is extremely large and sparse. The naive way to solve (1) is to partition $\mathbf{A}$ into n equal blocks $\{\mathbf{A}_i\}_{i=1}^n$ and store them in the memory of several workers. In each iteration, the master node broadcasts the $\mathbf{x}_t$ into all workers, each worker computes a partial linear transform $\mathbf{A}_i\mathbf{x}_t$ and sends it back to the master node. Then the master node collects all the partial results and updates the vector. The polynomial code [Yu et al., 2017] works as follows: in each iteration, the kth worker essentially computes

$$\tilde{\mathbf{y}}_k = \tilde{\mathbf{A}}_k\mathbf{x} \triangleq \left(\sum\nolimits_{i=1}^{n} \mathbf{A}_i x_k^i\right)\mathbf{x}, \tag{2}$$

and x_k is a given integer. One can observe that, due to the sparsity of matrix $\mathbf{A}$ and matrices additions, the density of the coded submatrices $\mathbf{A}_k$ will increase at most n times, and the time of sublinear transform $\tilde{\mathbf{A}}_k\mathbf{x}$ will increase roughly $O(n)$ times of the simple uncoded one.

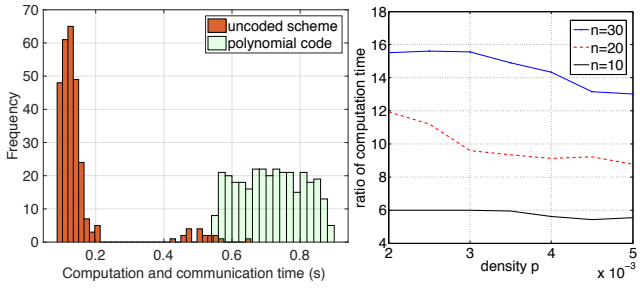

Figure 1: Measured local computation time per worker.

In the Figure 1(a), we show measurements on local computation and communication time required for $m = 20$ workers to operate the polynomial code and naive uncoded distributed scheme for the above problem with dimension roughly equal to 10^6 and number of nonzero elements equal to 10^7. Our observation 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 increased computation time. In the Figure 1(b), we generate a 10^5 random square Bernoulli matrix 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 this ratio is generally large and equal to the order of $O(n)$, when the matrix is sparse.

Therefore, inspired by this phenomenon, we propose a new metric, named as *computation load*, which is defined as total number of submatrices the local works access (formal definition can be seen in Section 2). For example, the polynomial code achieves optimal recovery threshold but a large computation load of mn. In certain suboptimal codes such as short-dot code, they achieve slightly lower computation load but still on the order of $O(mn)$. Specifically, we are interested in the following key problem: *can we find a coded linear transformation scheme that achieves optimal recovery threshold and low computation load?*

1.2 Main Contribution

In this paper, we provide a fundamental limit of the minimum computation load: given n partitions and s stragglers, the minimum computation load of any coded computation scheme is $n(s+1)$. We then design a novel scheme, we call *s-diagonal code*, that achieves both the optimum recovery threshold and the minimum computation load. We also exploit the diagonal structure to design a hybrid decoding algorithm between peeling decoding and Gaussian elimination that achieves the nearly linear decoding time of the output dimension $O(r)$.

We further show that, under random code designs, the computation time can be reduced even lower to a **constant** with high probability. Specifically, we define a performance metric *probabilistic recovery threshold* that allows the coded computation scheme to provide the decodability with high probability. Based on this metric, there exist several schemes, i.e., sparse MDS code [Lee et al., 2017b] and sparse code [Wang et al.] achieving the optimal probabilistic recovery threshold but a small computation load $\Theta(n\log(n))$. In this work, we construct a new (d_1, d_2)-*cross code* with optimal probabilistic recovery threshold but constant computation load $\Theta(n)$. The comparison of existing and our results are listed in TABLE 1.

The theoretical analysis to show the optimality of the re-

covery threshold is based on a determinant analysis of a random matrix in $\mathbb{R}^{m\times n}$. The state-of-the-art in this field is limited to the Bernoulli case [Tao and Vu, 2007, Bourgain et al., 2010], in which each element is identically and independently distributed random variable. However, in our proposed (d_1, d_2)-*cross code*, the underlying coding matrix is generated based on a hypergeometric degree distribution, which leads to dependencies among the elements in the same row. To overcome this difficulty, we propose a new 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 random proposal graph theory and the probabilistic method to show that when n tasks are collected, the coefficient matrix is full rank with high probability.

Further, we apply our proposed random codes to the gradient coding problem. This problem has wide applicability in mitigating the stragglers of distributed machine learning, and was first investigated in [Tandon et al., 2017]. It designs a cyclic code that achieves the optimum recovery threshold n and computation load $s+1$ per worker. The work [Maity et al., 2018] proposes an LDPC code to further reduce the average computation load to $\Theta(\log(n))$. Another line of works [Karakus et al., 2017, Charles et al., 2017, Wang et al., 2019] try to reduce the computation load by approximated gradient computation. In this paper, we show that our constructed (d_1, d_2)-cross code can not only exactly recover the gradient (sum of functions) but also provides a constant computation load $\Theta(1)$.

Finally, we implement the constructed codes and demonstrate their improvement compared with existing strategies.

2 Problem Formulation

We are interested in distributedly computing a linear transform with matrix $\mathbf{A} \in \mathbb{R}^{r\times t}$ and input vector $\mathbf{x} \in \mathbb{R}^t$ for some integers r, t. The matrix A is evenly divided along the row side into n submatrices.

$$\mathbf{A}^T = [\mathbf{A}_1^T, \mathbf{A}_2^T, \mathbf{A}_3^T, \ldots, \mathbf{A}_n^T] \tag{3}$$

Suppose that we have a master node and m worker nodes. Worker i first stores $1/n$ fraction of matrix A, defined as $\tilde{\mathbf{A}}_i \in \mathbb{R}^{\frac{r}{n}\times t}$. Then it can compute a partial linear transform $\tilde{\mathbf{y}}_i = \tilde{\mathbf{A}}_i\mathbf{x}$ and return it to the master node. The master node waits only for the results of a subset of workers $\{\tilde{\mathbf{A}}_i\mathbf{x} | i \in I \subseteq [m]\}$ to recover the final output $\mathbf{y}$ using certain decoding algorithms. The main framework is illustrated in Figure 2. Given the above system model, we can formulate the coded distributed linear transform problem based on the following definitions.

Definition 1. (Coded computation strategy) *A coded computation strategy is an $m \times n$ coding matrix $\mathbf{M} = [m_{ij}]_{i\in[m],j\in[n]}$ that is used to compute each $\tilde{\mathbf{A}}_i$,*

$$\tilde{\mathbf{A}}_i = \sum_{j=1}^{n} m_{ij}\mathbf{A}_j, \forall i \in [m]. \tag{4}$$

Then each worker i computes $\tilde{\mathbf{y}}_i = \tilde{\mathbf{A}}_i\mathbf{x}$.

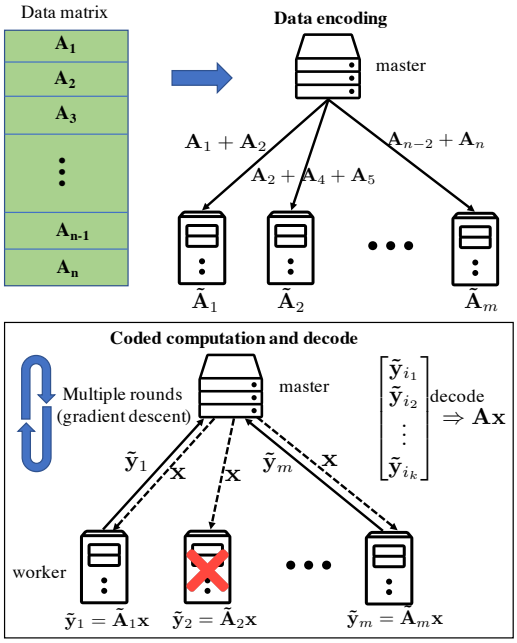

Figure 2: Framework of coded distributed linear transform.

This is a general definition of a large class of coded computation schemes. For example, in the polynomial code [Yu et al., 2017], the coding matrix $\mathbf{M}$ is the Vandermonde matrix. In the MDS type of codes [Dutta et al., 2016, Lee et al., 2017a,b], $\mathbf{M}$ is a specific form of corresponding generator matrices.

Definition 2. (Recovery threshold) *A coded computation strategy $\mathbf{M}$ is k-recoverable if for any subset $I \subseteq [m]$ with $|I| = k$, the master node can recover $\mathbf{Ax}$ from $\{\tilde{\mathbf{y}}_i | i \in I\}$. The recovery threshold $\kappa(\mathbf{M})$ is defined as the minimum integer k such that strategy $\mathbf{M}$ is k-recoverable.*

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

$$\kappa^* = \min_{\mathbf{M}\in\mathbb{R}^{m\times n}} \kappa(\mathbf{M}) \geq n. \tag{5}$$

Definition 3. (Computation load) *The computation load of strategy $\mathbf{M}$ is defined as $l(\mathbf{M}) = \|\mathbf{M}\|_0$, the number of the nonzero elements of coding matrix.*

The goal of this paper is to design a coded computation strategy $\mathbf{M}$ that achieves optimal recovery threshold n with low computation load $l(\mathbf{M})$. Due to the space limit, all the technical proofs in the sequel are provided in the Appendix.

3 Fundamental Limits and Diagonal Code

In this section, we will describe the optimum computation load and the s-diagonal code that exactly matches such lower bound. Then we will provide a fast decoding algorithm with nearly linear decoding time.

3.1 Fundamental Limits on Computation Load

We first establish the lower bound on the computation load, i.e., the density of coding matrix $\mathbf{M}$.

Theorem 1. (Optimum computation load) *For any coded computation scheme $\mathbf{M} \in \mathbb{R}^{m\times n}$ using m workers that can each store $\frac{1}{n}$ fraction of $\mathbf{A}$, to resist s stragglers, we have*

$$l(\mathbf{M}) \geq n(s+1). \tag{6}$$

The polynomial code [Yu et al., 2017] achieves the optimum recover threshold of n. However, the coding matrix (Vandermonde matrix) is fully dense, i.e., $l(\mathbf{M}) = nm$, and far beyond the above bound. The short-dot code [Dutta et al., 2016] can reduce the computation load but sacrifices the recovery threshold. Therefore, a natural question that arises is, can we design a code that achieves such lower bound? We will answer this question in the sequel of this paper.

3.2 Diagonal Code

We now present the s-diagonal code that achieves both the optimum recovery threshold and optimum computation load for any given parameter values of n, m and s.

Definition 4. (s-diagonal code) *Given parameters m,n and s, the s-diagonal code is defined as*

$$\tilde{\mathbf{A}}_i = \sum\nolimits_{j=\max\{1,i-s\}}^{\min\{i,n\}} m_{ij}\mathbf{A}_j, \forall i \in [m], \tag{7}$$

where each coefficient m_{ij} is chosen from a finite set S independently and uniformly at random.

The reason we name this code as the s-diagonal code is that the nonzero positions of the coding matrix $\mathbf{M}$ exhibit the following block diagonal structure.

$$\mathbf{M}^T = \begin{bmatrix} \overbrace{* & * & \cdots & *}^{s+1} & 0 & 0 & 0 & \cdots & 0 \\ 0 & * & * & \cdots & * & 0 & 0 & \cdots & 0 \\ 0 & 0 & * & * & \cdots & * & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & * & * & \cdots & * & * \end{bmatrix},$$

where $*$ indicates the nonzero entries of $\mathbf{M}$. Before we analyze the recovery threshold of the diagonal code, the following example provides an instantiation for $n = 4$, $s = 1$ and $m = 5$.

Example 1: (1-Diagonal code) Consider a distributed linear transform task $\mathbf{Ax}$ using $m = 5$ workers. We evenly divide the matrix $\mathbf{A}$ along the row side into 4 submatrices: $\mathbf{A}^T = [\mathbf{A}_1^T, \mathbf{A}_2^T, \mathbf{A}_3^T, \mathbf{A}_4^T]$. Given this notation, we need to compute the following 4 uncoded components $\{\mathbf{A}_1\mathbf{x}, \mathbf{A}_2\mathbf{x}, \mathbf{A}_3\mathbf{x}, \mathbf{A}_4\mathbf{x}\}$. Based on the definition of the 1-diagonal code, each worker stores the following submatrices: $\tilde{\mathbf{A}}_1 = \mathbf{A}_1,\ \tilde{\mathbf{A}}_2 = \mathbf{A}_1 + \mathbf{A}_2,\ \tilde{\mathbf{A}}_3 = \mathbf{A}_2 + \mathbf{A}_3,\ \tilde{\mathbf{A}}_4 = \mathbf{A}_3 + \mathbf{A}_4,\ \tilde{\mathbf{A}}_5 = \mathbf{A}_4$. Suppose that the first worker is a straggler and the master node receives results from worker $\{2, 3, 4, 5\}$. According to the above coded computation strategy, we have

$$\begin{bmatrix} \tilde{\mathbf{y}}_2 \\ \tilde{\mathbf{y}}_3 \\ \tilde{\mathbf{y}}_4 \\ \tilde{\mathbf{y}}_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}_1\mathbf{x} \\ \mathbf{A}_2\mathbf{x} \\ \mathbf{A}_3\mathbf{x} \\ \mathbf{A}_4\mathbf{x} \end{bmatrix} \tag{8}$$

The coefficient matrix is an upper diagonal matrix, which is invertible since the elements in the main diagonal are nonzero. Then we can recover the uncoded components $\{\mathbf{A}_i\mathbf{x}\}$ by direct inversion of the above coefficient matrix. The decodability for the other 4 possible scenarios can be proved similarly. Therefore, this code achieves the optimum recovery threshold of 4.

Obviously, the computation load of the s-diagonal code is optimal and can be easily obtained by counting the number of nonzero elements of coding matrix $\mathbf{M}$. The following result gives us the recovery threshold of the s-diagonal code.

Theorem 2. *Let finite set S satisfy $|S| \geq 2n^2C_m^n$. Then there exists an s-diagonal code that achieves the recovery threshold n, and can be constructed, on average, in 2 trails.*

The theoretical framework on analyzing the recovery threshold is provided in Section 4.

3.3 Fast Decoding Algorithm

The original decoding algorithm of s-diagonal code is based on inverting the coding matrix, and the mappings from $[\tilde{\mathbf{y}}_{i_1}, \tilde{\mathbf{y}}_{i_2}, \ldots, \tilde{\mathbf{y}}_{i_n}]$ to vector $[\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$ incurs a complexity of $O(nr)$. We next show that it can be further reduced by a hybrid decoding algorithm between the peeling decoding and Gaussian elimination. The key idea is to utilize the peeling decoding algorithm to reduce the number of blocks recovered by above mapping, and Gaussian elimination to guarantee the existence of the ripple node.

Suppose that the master node receives results from workers indexed by $U = \{i_1, \ldots, i_n\} \subseteq [n + s]$ with $1 \leq i_1 \leq \cdots \leq i_n \leq n + s$. Let $\mathbf{M}^U$ be a $n \times n$ submatrix consisting of rows of $\mathbf{M}$ index by U. Let the index $k \in [n]$ be $i_k \leq n < i_{k+1}$.Then recover the blocks indexed by $[n]\backslash\{i_1, \ldots, i_k\}$ from the following **rooting step**, and add the recovered results into the ripples set $\mathcal{R} = \{\mathbf{A}_i\mathbf{x}\}_{i\in[n]\backslash\{i_1,\ldots,i_k\}}$.

Lemma 1. (Rooting step) *If rank$(\mathbf{M}^U) = n$, then for any $k_0 \in \{1, 2, \ldots, n\}$, we can recover a particular block $\mathbf{A}_i\mathbf{x}$*

Algorithm 1 Fast decoding algorithm for s-diagonal code

Receive n results with coefficient matrix. Find the index $k \in [n]$ with $i_k \leq n < i_{k+1}$.
Recover the blocks indexed by $[n]\backslash\{i_1, \ldots, i_k\}$ by (16).
Construct ripples set $\mathcal{R} = \{\mathbf{A}_i\mathbf{x}\}_{i\in[n]\backslash\{i_1,\ldots,i_k\}}$
repeat
  if $\mathcal{R}$ is not empty **then**
    Choose a result $\mathbf{A}_i\mathbf{x}$ from $\mathcal{R}$.
    for each computation results $\tilde{\mathbf{y}}_j$ **do**
      if M_{ji} is nonzero **then**
        $\tilde{\mathbf{y}}_j = \tilde{\mathbf{y}}_j - m_{ji}\mathbf{A}_i\mathbf{x}$ and set $m_{ji} = 0$.
      end if
    end for
  else
    Find a row $\mathbf{M}_{i'}^U$ in matrix $\mathbf{M}^U$ with $\|\mathbf{M}_{i'}^U\|_0 = 1$.
    $\mathcal{R} = \mathcal{R} \cup \tilde{\mathbf{y}}_{i'}$.
  end if
until every block of vector $\mathbf{y}$ is recovered.

with column index k_0 in matrix $\mathbf{M}^U$ via the following linear combination.

$$\mathbf{A}_i\mathbf{x} = \sum\nolimits_{k=1}^{n} u_k\tilde{\mathbf{y}}_k. \tag{9}$$

The vector $u = [u_1, \ldots, u_n]$ can be determined by solving $\mathbf{u}^\intercal\mathbf{M} = \mathbf{e}_{k_0}$, where $\mathbf{e}_{k_0} \in \mathbb{R}^{1\times n}$ is a unit vector with unique 1 locating at the index k_0.

Then the master node goes to a peeling decoding process: the master node first finds a ripple in set $\mathcal{R}$, i.e., $\mathbf{A}_i\mathbf{x}$. For each collected results $\tilde{\mathbf{y}}_j$, it subtracts this block if the computation task $\tilde{\mathbf{A}}_j\mathbf{x}$ contains this block ($M_{ji} \neq 0$). If the set $\mathcal{R}$ is empty, the master node finds a new ripple that computes a uncoded task in the rest workers, add it to set $\mathcal{R}$ and continue above process.

Example 2: (Fast decoding algorithm) Consider a similar setup with **Example 1**. Suppose that we receive the results from workers index by $U = \{2, 3, 4, 5\}$. The naive inverse decoding algorithm will lead to decoding complexity $4r$ (complexity of inverse mapping of (8)). In the fast decoding algorithm, we first recover the block $\mathbf{A}_1\mathbf{x}$ ($\{1, 2, 3, 4\}\backslash\{2, 3, 4\}$) by rooting step $\mathbf{A}_1\mathbf{x} = \tilde{\mathbf{y}}_1 - \tilde{\mathbf{y}}_2 + \tilde{\mathbf{y}}_3 - \tilde{\mathbf{y}}_4$. Then we start the peeling decoding process: (i) recover the block $\mathbf{A}_2\mathbf{x}$ by $\tilde{\mathbf{y}}_2 - \mathbf{A}_1\mathbf{x}$, add the result to set $\mathcal{R}$; (ii) recover the block $\mathbf{A}_3\mathbf{x}$ by $\tilde{\mathbf{y}}_3 - \mathbf{A}_2\mathbf{x}$, add the result to set $\mathcal{R}$; (iii) recover the block $\mathbf{A}_4\mathbf{x}$ by $\tilde{\mathbf{y}}_4 - \mathbf{A}_3\mathbf{x}$. Obviously, the whole procedure of new algorithm leads to complexity $7r/4$, which is smaller than $4r$.

The main procedure is listed in Algorithm 1, and the decoding complexity is given by the following theorem.

Theorem 3. (Nearly linear time decoding of s-diagonal codes) *The Algorithm 1 uses the at most s times of the rooting steps (16) and the total decoding time is $O(rs)$.*

4 Graph-based Analysis of Recovery Threshold

In this section, we will present our main technical tool to analyze the recovery threshold of proposed code, which is also the basis in our random code construction of next section.

For each subset $U \subseteq [m]$ with $|U| = n$, let $\mathbf{M}^U$ be an $n \times n$ submatrix consisting of rows of $\mathbf{M}$ index by U. To prove that our s-diagonal code achieves the recovery threshold of n, we need to show that all the $n \times n$ submatrices $\mathbf{M}^U$ are full rank. The basic idea is to reduce the full rank analysis to the analysis of the existence of perfect matching in the corresponding bipartite graph. We first define the following bipartite graph model between the set of m workers indexed by $[m]$ and the set of n data partitions indexed by $[n]$.

Definition 5. *Let $G^D(V_1, V_2)$ be a bipartite graph with $V_1 = [m]$ and $V_2 = [n]$. Each node $i \in V_1$ is connected to nodes $j \subseteq V_2$ if $M_{ij} \neq 0$.*

Definition 6. *Define an Edmonds matrix $\mathbf{M}(\mathbf{x}) \in \mathbb{R}(\mathbf{x})^{m\times n}$ of graph $G^D(V_1, V_2)$ with $[\mathbf{M}(\mathbf{x})]_{ij} = x_{ij}$ if nodes $i \in V_1$ and $j \in V_2$ are connected; $[\mathbf{M}(x)]_{ij} = 0$, otherwise.*

Based on this notation, our main result is given by the following theorem.

Theorem 4. *For a coded computation scheme $\mathbf{M} \in \mathbb{R}^{m\times n}$, if every subgraph $G^D(U, V_2)$ of $G^D(V_1, V_2)$ with $U \subseteq [m]$ and $|U| = n$ contain a perfect matching, the recovery threshold $\kappa(\mathbf{M}) = n$ with probability at least $1 - n^2\binom{m}{n}/|S|$.*

Proof. Based on the above definition, the coding matrix $\mathbf{M}$ of the s-diagonal code can be obtained by assigning each intermediate x_{ij} of the Edmonds matrix $\mathbf{M}(\mathbf{x})$ a value from set S independently and uniformly at random. Given a subgraph $G^D(U, V_2)$ of $G^D(V_1, V_2)$, let $\mathbf{M}^U(\mathbf{x})$ be the corresponding Edmonds matrix. Then the probability that matrix $\mathbf{M}^U$ is full rank is equal to the probability that the determinant of the Edmonds matrix $\mathbf{M}^U(\mathbf{x})$ is nonzero at the given value $\mathbf{x}$. The following technical lemma from [Schwartz, 1980] provides a simple lower bound of such an event.

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

$$\mathbb{P}(f(x_1, x_2, \ldots, x_N) \neq 0) \geq 1 - n^2/|S|. \tag{10}$$

A classical result in graph theory is that a bipartite graph $G^D(U, V_2)$ contains a perfect matching if and only if the determinant of the Edmonds matrix, i.e., $|\mathbf{M}^U(\mathbf{x})|$, is a nonzero polynomial. Combining this result with Schwartz-Zeppel Lemma, we can reduce the analysis of the full rank

probability of the submatrix $\mathbf{M}^U$ to the probability that the subgraph $G^D(U, V_2)$ contains a perfect matching.

$$\mathbb{P}(|\mathbf{M}^U| \neq 0) = \underbrace{\mathbb{P}(|\mathbf{M}^U| \neq 0 \big| |\mathbf{M}^U(x)| \not\equiv 0)}_{\text{S-Z Lemma: } \geq 1-1/2C_n^m} \cdot \underbrace{\mathbb{P}(|\mathbf{M}^U(x)| \not\equiv 0)}_{\text{contains perfect matching}} + \underbrace{\mathbb{P}(|\mathbf{M}^U| \neq 0 \big| |\mathbf{M}^U(x)| \equiv 0)}_{0} \cdot \mathbb{P}(|\mathbf{M}^U(x)| \equiv 0) \quad (11)$$

Therefore, utilizing the union bound, we conclude that the probability that there exists a submatrix $\mathbf{M}^U$ is not full rank is upper bound by $n^2\binom{m}{n}/|S|$. □

The next technical lemma shows that for all subsets $U \subseteq [m]$ with $|U| = n$, the subgraph $G^D(U, V_2)$ exactly contains a perfect matching for diagonal code. Therefore, let $|S| \geq 2n^2\binom{m}{n}$, we conclude that, with probability at least $1/2$, all the $n \times n$ submatrices of $\mathbf{M}$ are full rank. Since we can generate the coding matrix $\mathbf{M}$ offline, with a few rounds of trials (2 on average), we can find a coding matrix with all $n \times n$ submatrices being full rank. Therefore, we arrive the Theorem 2.

Lemma 3. *Let $\mathbf{M}$ be constructed as Definition 4 and the bipartite graph $G^D(V_1, V_2)$ be constructed as Definition 5. For each $U \subseteq [m]$ with $|U| = n$, the subgraph $G^D(U, V_2)$ contains a perfect matching.*

One special case of the s-diagonal code is that, when we are required to resist only one straggler, all the nonzero elements of matrix $\mathbf{M}$ can be equal to 1.

Corollary 1. *Given the parameters n and $m = n+1$, define the 1-diagonal code:*

$$\tilde{\mathbf{A}}_i = \begin{cases} \mathbf{A}_1, i = 1; \mathbf{A}_n, i = n+1 \\ \mathbf{A}_{i-1} + \mathbf{A}_i, 2 \leq i \leq n \end{cases}.$$

It achieves the optimum computation load $2n$ and optimum recovery threshold n.

5 Random Code: "Break" the Limits

In this section, we utilize our proposed theoretical framework in Theorem 4 to construct a random code that achieve the optimal recovery threshold with high probability but with constant computation load.

5.1 Probabilistic Recovery Threshold

In practice, the stragglers randomly occur in each worker, and any specific straggling configuration happens with very low probability. We first demonstrate the main idea through the following motivating examples.

Example 3: Consider a distributed linear transform task with $n = 20$ data partitions, $s = 5$ stragglers and $m = 25$ workers. The recovery threshold of 20 implies that all $C_{25}^{20} = 53130$ square 20×20 submatrices of coding matrix are full rank. Suppose that a worker being a straggler is identically and independently Bernoulli random variable with probability 10%. Then, the probability that workers $\{1, 2, 3, 4, 5\}$ are all stragglers is 10^{-5}. Now, if there exists a scheme that can guarantee that the master can decode the results in all configurations except the straggling configuration $\{1, 2, 3, 4, 5\}$, we can argue that this scheme achieves a recovery threshold 20 with probability $1 - 10^{-5}$.

Example 4: Consider the same scenario of **Example 1** ($n = 4, s = 1, m = 5$). We change the coded computation strategy of the second worker from $\tilde{\mathbf{A}}_2 = \mathbf{A}_1 + \mathbf{A}_2$ to $\tilde{\mathbf{A}}_2 = \mathbf{A}_2$. Based on the similar analysis, we can show that the new strategy can recover the final result from 4 workers except the scenario that the first worker is a straggler. Therefore, the new strategy achieves recovery threshold 4 with probability 0.75, and reduces the computation load by 1.

Based on the above two examples, we observe that the computation load can be reduced when we allow the coded computation strategy to fail in some specific scenarios. Formally, it motivates us to define the following metric.

Definition 7. (Probabilistic recovery threshold) *A coded computation strategy $\mathbf{M}$ is probabilistic k-recoverable if for each subset $I \subseteq [m]$ with $|I| = \kappa(\mathbf{M})$, the master node can recover $\mathbf{Ax}$ from $\{\tilde{\mathbf{A}}_i\mathbf{x} | i \in I\}$ with high probability, i.e., $1 - O(2^{-n})$. The probabilistic recovery threshold $\kappa(\mathbf{M})$ is defined as the minimum integer k such that strategy $\mathbf{M}$ is probabilistic k-recoverable.*

The new definition provides a probabilistic relaxation such that a small vanishing percentage (as $n \to \infty$) of all straggiling configurations, are allowed to be unrecoverable. In the sequel, we show that, under such a relaxation, one can construct a coded computation scheme that achieves a probabilistic recovery threshold n and a constant (regarding parameter s) computation load.

5.2 Construction of the Random Code

Based on our previous analysis of the recovery threshold of the $s-$diagonal code, we show that, for any subset $U \subseteq [m]$ with $|U| = n$, the probability that $\mathbf{M}^U$ is full rank is lower bounded by the probability (multiplied by $1 - o(1)$) that the corresponding subgraph $G(U, V_2)$ contains a perfect matching. This technical path motivates us to utilize the random proposal graph to construct the coded computation scheme. The first one is the following p-Bernoulli code, which is constructed from the ER random bipartite graph model [Erdos and Renyi, 1964].

Definition 8. (p-Bernoulli code) *Given parameters m, n, construct the coding matrix $\mathbf{M}$ as follows:*

$$m_{ij} = \begin{cases} t_{ij}, \textit{with probability } p \\ 0, \textit{with probability } 1-p \end{cases}. \quad (12)$$

where t_{ij} is picked independently and uniformly from the finite set S.

Theorem 5. *For any parameters m, n and s, if $p = 2\log(n)/n$, the p-Bernoulli code achieves the probabilistic recovery threshold n.*

This result implies that each worker of the p-Bernoulli code requires accessing $2\log(n)$ submatrices on average, which is independent of number of the stragglers. Note that the existing work in distributed functional computation [Lee et al., 2017b] proposes a random sparse code that also utilizes Bernoulli random variables to construct the coding matrix. There exist two key differences: (i) the elements of our matrix can be integer valued, while the random sparse code adopts real-valued matrix; (ii) the density of the p-Bernoulli code is $2\log(n)$, while the density of the random sparse code is an unknown constant. The second random code is the following (d_1, d_2)-cross code.

Definition 9. ((d_1, d_2)-cross code) *Given parameters m, n, construct the coding matrix $\mathbf{M}$ as follows: (1) Each row (column) independently and uniformly chooses d_1 (d_2) nonzero positions; (2) For those nonzero positions, assign the value independently and uniformly from the finite set S.*

The computation load of (d_1, d_2)-cross code is upper bounded by $d_1 m + d_2 n$. The numerical analysis of proposed random codes can be seen in Appendix. The next theorem shows that a constant choice of d_1 and d_2 can guarantee the probabilistic recovery threshold n.

Theorem 6. *For any parameters m, n, if $s = poly(\log(n))$, the $(2, 3)$-cross code achieves the probabilistic recovery threshold n. If $s = \Theta(n^\alpha)$, $\alpha < 1$, the $(2, 2/(1-\alpha))$-cross code achieves the probabilistic recovery threshold n.*

The proof of this theorem is based on analyzing the existence of perfect matching in a random bipartite graph constructed as follows: (i) each node in the left partition randomly and uniformly connects to d_1 nodes in the opposite class; (ii) each node in the right partition randomly and uniformly connects to l nodes in the opposite class, where l is chosen under a specific degree distribution. This random graph model can be regarded as a generalization of Walkup's 2-out bipartite graph model [Walkup, 1980]. The main technical difficulty in this case derives from the intrinsic complicated statistical model of the node degree of the right partition.

6 Applications to Gradient Coding

In this section, we discuss the application of proposed coded schemes into the distributed gradient descent problem. Given data sets $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where each block $(\mathbf{x}_i, y_i) \in \mathbb{R}^{kp} \times \mathbb{R}^k$ consists of k data samples. Several machine learning tasks aim to solve the following problem

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^r} \sum_{i=1}^{n} l(\mathbf{w}; \mathbf{x}_i, y_i) + \lambda R(\mathbf{w}), \tag{13}$$

where $l(\cdot)$ is a task-specific loss function and $R(\cdot)$ is the regularization function. Typically, this problem is solved by the following gradient-based approaches.

$$\mathbf{w}_{t+1} = h_R\left(\mathbf{w}_t - \eta_t \sum_{i=1}^{n} \nabla l(\mathbf{w}_t; \mathbf{x}_i, y_i)\right), \tag{14}$$

where $h_R(\cdot)$ is the proximal mapping, which depends on the regularization function. Several methods such as gradient descent, accelerated gradient, Frank-Wolfe, proximal methods, LBFGS, and bundle methods fits in this framework. The gradient coding framework is: (i) Initially, each data blocks are assigned to each worker based on the given coefficient matrix; (ii) In each iteration t, the master node broadcasts the vector $\mathbf{w}_t$ to each worker, and each worker i calculates a single linear combination of gradient vectors

$$\tilde{\mathbf{g}}_i = \sum_{j=1}^{n} m_{ij} \nabla l(\mathbf{w}_t; \mathbf{x}_j, y_j), \forall i \in [m]. \tag{15}$$

(iii) The master node collects a subset of results from m workers, decodes the full gradient, and update the weight vector based on the proximal gradient descent step (14). The algorithm is terminated until converged, i.e., the gradient vanishes. One instantiation of this framework is the linear regression problem that is provided in subsection H.2. We can see that, in this problem, the defined computation load $l(\mathbf{M})$ is exactly the number of partial gradients evaluations locally. Using the proposed (d_1, d_2)-cross code, we can obtain a gradient coding scheme that each worker only requires to store a constant number of data blocks, and the recovery threshold is n with high probability. The straightforward inverse decoding will lead to a complexity of $O(n^2 r)$ (decode n partial gradients from the results of n workers). However, the following technical lemma shows that we can actually decode in $O(nr)$ time.

Lemma 4. (gradient decoding) *Assume received coding matrix $\mathbf{M}^U$. If rank$(\mathbf{M}^U) = n$, then we can recover full gradient $\sum_{i=1}^{n} \nabla l(\mathbf{w}_t; \mathbf{x}_i, y_i)$ via the following linear combination.*

$$\sum\nolimits_{i=1}^{n} \nabla l(\mathbf{w}_t; \mathbf{x}_i, y_i) = \sum\nolimits_{i=1}^{n} u_i \tilde{\mathbf{g}}_i \tag{16}$$

The vector $u = [u_1, \ldots, u_n]$ can be determined by solving $\mathbf{u}^\intercal \mathbf{M}^U = \mathbf{1}_n$, where $\mathbf{1}_n \in \mathbb{R}^{1\times n}$ is an all one vector.

The basic intuition is to find a linear combination of row vectors of matrix $\mathbf{M}^U$ such that the row vectors span the all one vector $\mathbf{1}_n$.

7 Experimental Results

In this section, we present the experimental results on Ohio Supercomputer Center Center [1987]. We compare our proposed coding schemes including the $s-$diagonal code and

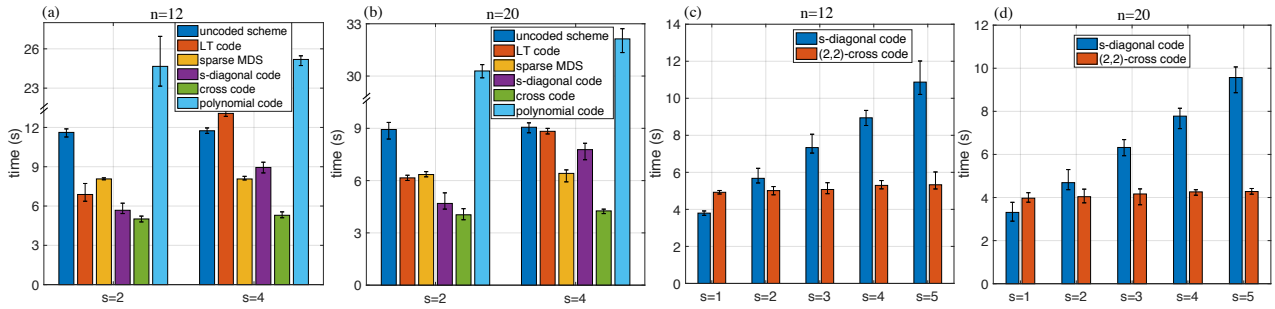

Figure 3: Comparison of total time including transmission, computation and decoding time for $n = 12, 20$ and $s = 2, 4$.

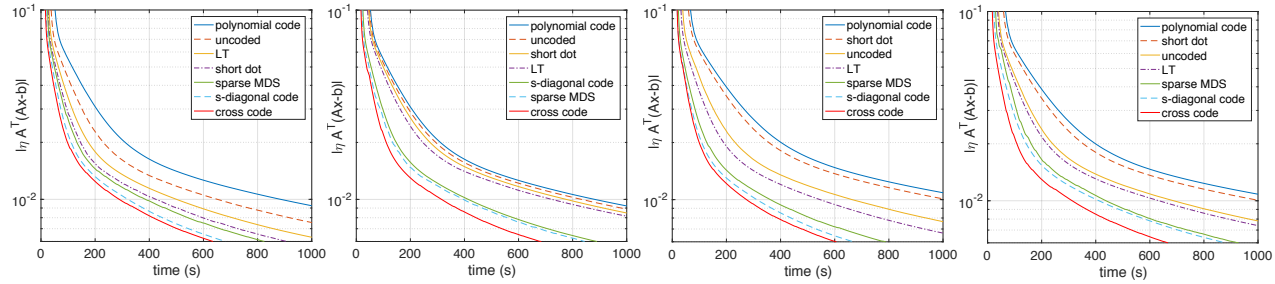

Figure 4: Magnitude of gradient versus time for number of data partitions $n = 12$, 20 and number of stragglers $s = 2$,4.

(d_1, d_2)-cross codes against the following existing schemes in both single matrix vector multiplication and gradient coding problem: (i) **uncoded scheme**: the input matrix is divided uniformly across all workers without replication 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 $\Theta(\log(n))$; (iii) **polynomial code** [Yu et al., 2017]: coded matrix multiplication scheme with optimum recovery threshold and nearly linear decoding time; (iv) **short dot code** [Dutta et al., 2016]: appending the dummy vectors to data matrix $\mathbf{A}$ before applying the MDS code, which provides certain sparsity of encoded data matrix with cost of increased recovery threshold; (v) **LT code** [Luby, 2002]: rateless code widely used in broadcast communication. It achieves an average computation load of $\Theta(\log(n))$ and a nearly linear decoding time using peeling decoder. To simulate straggler effects in large-scale system, we randomly pick s workers that are running a background thread. More details can be seen in Appendix.

We first use a matrix with $r = t = 1048576$ and $\text{nnz}(\mathbf{A}) = 89239674$ from data sets [Davis and Hu, 2011] , and evenly divide this matrix into $n = 12$ and 20 partitions. In Figure 7 (a)(b), we report the job completion time under $s = 2$ and $s = 4$, based on 20 experimental runs. It can be observed that both $(2, 2)$-cross code outperforms uncoded scheme (in 50% the time), LT code (in 70% the time), sparse MDS code (in 60% the time), polynomial code (in 20% the time) and our s-diagonal code. Moreover, we compare our proposed s-diagonal code with $(2, 2)$-cross code versus the number of stragglers s. As shown in Figure 7 (a)(b), when the number of stragglers s increases, the job completion time of the s-diagonal code increases while that of $(2, 2)$-cross code does not change. Another interesting observation is that the *irregularity of the work load* can decrease the I/O contention. For example, when $s = 2$, the computation load of the 2-diagonal code is similar as $(2, 2)$-cross code, which is equal to 36 in the case of $n = 12$. However, the $(2, 2)$-cross code cost less time due to the random worker load.

We finally compare our proposed codes with existing schemes in a gradient coding problems. Details can be seen in the Appendix. We use data from LIBSVM dataset repository with $r = 19264097$ samples and $t = 1163024$ features. We evenly divide the data matrix $\mathbf{A}$ into $n = 12$ submatrices. In Fig. 7 (c)(d) , we plot the magnitude of scaled gradient $\|\eta\mathbf{A}^\intercal(\mathbf{A}\mathbf{x} - \mathbf{b})\|$ versus the running time of the above seven different schemes under $n = 12$ and $s = 2, 4$. Among all experiments, we can see that the $(2, 2)$-cross code converges at least 30% faster than sparse MDS code, 2 times faster than both uncoded scheme and LT code and at least 4 times faster than short dot and polynomial code. The $(2, 2)$-cross code performs similar with s-diagonal code when $s = 2$ and converges 30% faster than s-diagonal code when number of stragglers increases to $s = 4$.

8 Conclusion

In this paper, we characterized the fundamental limits of the minimum computation load in the coded computation schemes, and proposed a new code, we call *s-diagonal code*, exactly achieves such lower bound and optimal recovery threshold. Furthermore, we exploited our proposed theoretical framework to design two *random codes* that achieves the optimal recovery threshold with high probability but with constant computation load. We implemented our proposed schemes in the distributed gradient descent problem and demonstrated the advantages over existing fastest schemes.

References

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

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

Zachary Charles, Dimitris Papailiopoulos, and Jordan Ellenberg. Approximate gradient coding via sparse random graphs. *arXiv preprint arXiv:1711.06771*, 2017.

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

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

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

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

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

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

Can Karakus, Yifan Sun, Suhas Diggavi, and Wotao Yin. Straggler mitigation in distributed optimization through data encoding. In *Advances in Neural Information Processing Systems*, pages 5440–5448, 2017.

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

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

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

Raj Kumar Maity, Ankit Singh Rawat, and Arya Mazumdar. Robust gradient descent via moment encoding with ldpc codes. In *SysML*, 2018.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

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

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

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

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

Sinong Wang, Jiashang Liu, and Ness Shroff. Coded sparse matrix multiplication. *International Conference on Machine Learning*, pages 5152–5160.

Sinong Wang, Jiashang Liu, and Ness Shroff. Fundamental limits of approximate gradient coding. *arXiv preprint arXiv:1901.08166*, 2019.

Yaoqing Yang, Pulkit Grover, and Soummya Kar. Coded distributed computing for inverse problems. In *Advances in Neural Information Processing Systems*, pages 709–719, 2017.

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

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