Factored LT and Factored Raptor Codes for Large-Scale Distributed Matrix Multiplication

Asit Kumar Pradhan, Anoosheh Heidarzadeh, and Krishna R. Narayanan

***Abstract*— We propose two coding schemes for distributed matrix multiplication in the presence of stragglers. These coding schemes are adaptations of LT codes and Raptor codes to distributed matrix multiplication and are termed *Factored LT (FLT) codes* and *Factored Raptor (FR) codes*. Empirically, we show that FLT codes have a near-optimal recovery threshold when the number of worker nodes is very large, and that FR codes have an excellent recovery threshold while the number of worker nodes is moderately large. FLT and FR codes have better recovery thresholds when compared to Product codes and they are expected to have better numerical stability when compared to Polynomial codes, while they can also be decoded with a low-complexity decoding algorithm.**

I. Introduction

We consider a matrix multiplication problem where the goal is to compute $\mathbf{C} = \mathbf{A}^\intercal \mathbf{B}$ given two input matrices $\mathbf{A} \in \mathbb{R}^{s\times r}$ and $\mathbf{B} \in \mathbb{R}^{s\times t}$. Many applications in optimization and machine learning require multiplications of large matrices of dimension of the order of $10^5 \times 10^5$. These large-scale matrix multiplications cannot be carried out in a single machine mainly due to low-latency requirement in many applications. This requirement can be met by dividing input matrices $\mathbf{A}$ and $\mathbf{B}$ into m and n sub-matrices, respectively, and distributing the tasks of computing the product of these sub-matrices among P worker nodes [1]–[12]. A master node then collects the results of the worker nodes, and aggregates them appropriately to obtain $\mathbf{C}$. Since the master node requires the results of all worker nodes so as to successfully compute $\mathbf{C}$, a few slow worker nodes, referred to as *stragglers*, can significantly increase the computational delay.

In [6], Lee *et al.* proposed a scheme, referred to as *1-D maximum distance separable (MDS) codes*, to mitigate the effect of stragglers for the case of $n = 1$, in which the sub-matrices of the matrix $\mathbf{A}$ are encoded using a (P, m) MDS code, and the task of computing $\mathbf{A}_i^\intercal \mathbf{B}$, for $i \in [P]$, is distributed among P worker nodes. This scheme can be extended to matrix-matrix multiplication in a natural way by considering each column of $\mathbf{B}$ as a vector; however, as shown in [2], the *recovery threshold* of this scheme —defined as the minimum number of worker nodes that the master node needs to wait for to compute $\mathbf{C}$, will not be optimal. In [2], Yu *et al.* proposed a coding scheme, referred to as *Polynomial codes*, which divides both $\mathbf{A}$ and $\mathbf{B}$ into sub-matrices. As was shown in [2], the Polynomial codes achieve the minimum possible recovery threshold, i.e., $K = mn$. The main drawback of Polynomial codes is that the decoding process requires interpolating polynomials of degree K, which is equivalent to inverting a $K \times K$ Vandermonde matrix. However, inverting real-valued Vandermonde matrices is highly numerically unstable even for moderate K [13].

In [9], Lee *et al.* proposed a coding scheme using Product codes, where both the input matrices $\mathbf{A}$ and $\mathbf{B}$ are split (column-wise) into $m = n$ sub-matrices, and encoded using an $(\sqrt{P}, m)$ MDS code, unlike the scheme in [6]. Product codes can be implemented using several component codes such as Polynomial codes [1], OrthoPoly codes [14], Random Khatri-Rao-Product (RKRP) codes [15], etc. [4], [16]. Product codes are generally better than Polynomial codes in terms of numerical stability. For example, decoding of a Product code of dimension K $(= m^2)$ built from Polynomial component codes of dimension $\sqrt{K}$ requires the inversion of $\sqrt{K} \times \sqrt{K}$ Vandermonde matrices, whereas decoding of a Polynomial code of dimension K would require inversion of a $K \times K$ Vandermonde matrix. The main drawback of Product codes is that their recovery threshold is not optimal unlike Polynomial codes. In [17], Baharav *et al.* proposed a scheme, referred to as *d-dimensional Product codes* for matrix multiplication, by spreading component matrices of $\mathbf{A}$ and $\mathbf{B}$ over $d/2$ dimensions each, and encoding them using a d-dimensional Product code. While d-dimensional Product codes can perform better than 2-dimensional Product codes for certain regimes of P and K, the recovery threshold of d-dimensional Product codes is still not optimal when the number of stragglers is linear in P. Moreover, (P, K) d-dimensional Product codes of rate R are built upon $(P^{1/d}, K^{1/d})$ MDS component codes of rate $R^{1/d}$. Unless P is very large, it is impractical to use large values of d since the set of possible MDS component codes becomes trivial for large d.

In [18], Mallick *et al.* proposed a scheme for matrix-vector multiplication using Luby Transform (LT) codes [19], a type of rateless codes, which has several desired properties: (i) high numerical stability; (ii) linear decoding complexity, and (iii) recovery threshold of $P(1 - \alpha)$, where $\alpha \in [0, 1)$ is the fraction of worker nodes that are stragglers. However, this scheme is not directly extendable to the matrix-matrix multiplication problem. In [20], Wang *et al.* proposed the use of LT codes for distributed matrix multiplication. The scheme of [20] has all the properties (i)-(iii); however, for this scheme, both the communication and the computation at each worker node are substantially more expensive than those of Polynomial codes and Product codes.

The authors are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (E-mail:{asit.pradhan,anoosheh,krn}@tamu.edu).

This material is based upon work supported by the National Science Foundation (NSF) under Grants No. 1718658, 1642983, and 1611285.

A. Main Contributions

In this work, we propose two novel encoding schemes. The first scheme is based on LT codes, referred to as *Factored LT (FLT) codes*, which is better in terms of numerical stability as well as decoding complexity, when compared to Polynomial codes. In particular, the decoding complexity of FLT codes is $\mathcal{O}(rt \log K)$, whereas the decoding complexity of Polynomial codes is $\mathcal{O}(rt \log^2 K \log\log K)$, where r and t are the number of columns in $\mathbf{A}$ and $\mathbf{B}$, respectively, and $K = mn$ where m and n are the number of sub-matrices into which $\mathbf{A}$ and $\mathbf{B}$ are split, respectively. As in the case of LT codes for erasure channels, the performance of FLT codes can be improved for finite lengths, particularly for moderate values of K. For this regime, we propose a Raptor code based scheme, termed *Factored Raptor (FR) codes*, which performs well even for moderately large K. Our simulation results show that the recovery threshold of FR codes is better than that of Product codes. For example, a $(10000, 6400)$ FR code has recovery threshold 7060, whereas the recovery threshold of a $(21, 18) \times (22, 19) \times (22, 19)$ Product code is 7275.

II. Notation and Problem Setup

A. Notation

We use boldface capital letters for matrices and underlined variables to represent vectors. We denote the (i, j)th element of the matrix $\mathbf{A}$ by A_{ij}, and represent the ith row and jth column of the matrix $\mathbf{A}$ by $\mathbf{A}_{i,:}$ and $\mathbf{A}_{:,j}$, respectively. We assume that vectors without transposes are column vectors unless stated otherwise. We denote $\{1, \ldots, i\}$ by $[i]$, and denote the cardinality of a set $\mathcal{S}$ by $|\mathcal{S}|$.

B. Problem Setup

We consider a system where there is one master node and P worker nodes. The goal of the master node is to compute $\mathbf{C} = \mathbf{A}^\mathsf{T}\mathbf{B}$ in a distributed fashion using P worker nodes. Worker nodes can only communicate with the master node and cannot communicate among themselves. To distribute the computation among workers, $\mathbf{A}$ and $\mathbf{B}$ are divided along columns to m and n sub-matrices of size $s \times \frac{r}{m}$ and $s \times \frac{t}{n}$, respectively, as shown below.

$$\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_m], \quad \text{and} \quad \mathbf{B} = [\mathbf{B}_1, \mathbf{B}_2, \cdots \mathbf{B}_n].$$

Alternatively, the output matrix $\mathbf{C}$ can be written in terms of the components of $\mathbf{A}$ and $\mathbf{B}$ as follows:

$$\mathbf{C} = \begin{bmatrix} \mathbf{A}_1^\mathsf{T}\mathbf{B}_1 & \mathbf{A}_1^\mathsf{T}\mathbf{B}_2 & \ldots & \mathbf{A}_1^\mathsf{T}\mathbf{B}_n \\ \mathbf{A}_2^\mathsf{T}\mathbf{B}_1 & \mathbf{A}_2^\mathsf{T}\mathbf{B}_1 & \ldots & \mathbf{A}_2^\mathsf{T}\mathbf{B}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_m^\mathsf{T}\mathbf{B}_1 & \mathbf{A}_m^\mathsf{T}\mathbf{B}_2 & \ldots & \mathbf{A}_m^\mathsf{T}\mathbf{B}_n \end{bmatrix}.$$

Then, computing $\mathbf{C}$ is equivalent to computing mn blocks $\mathbf{A}_i^\mathsf{T}\mathbf{B}_j$ for $i \in [m]$ and $j \in [n]$. For each $p \in [P]$, let $\mathcal{I}_\mathbf{A}^p$ and $\mathcal{I}_\mathbf{B}^p$ be two subsets of $[m]$ and $[n]$, respectively. For each $p \in [P]$, we define

$$(\widetilde{\mathbf{A}}^p)^\mathsf{T} = \sum_{i \in [m]} a_i^p \mathbf{A}_i^\mathsf{T}, \quad \widetilde{\mathbf{B}}^p = \sum_{j \in [n]} b_j^p \mathbf{B}_j, \tag{1}$$

where a_i^p for $i \in \mathcal{I}_\mathbf{A}^p$ and b_j^p for $j \in \mathcal{I}_\mathbf{B}^p$ are randomly sampled from a Gaussian distribution with zero mean and unit variance; and we choose $a_i^p = b_j^p = 0$ for $i \notin \mathcal{I}_\mathbf{A}^p$ and $j \notin \mathcal{I}_\mathbf{B}^p$. The master node computes $(\widetilde{\mathbf{A}}^p)^\mathsf{T}$ and $\widetilde{\mathbf{B}}^p$ and sends them to the worker node p. The worker node p computes $\widetilde{\mathbf{C}}^p = (\widetilde{\mathbf{A}}^p)^\mathsf{T}\widetilde{\mathbf{B}}^p$ and returns $\widetilde{\mathbf{C}}^p$ to the master node. We refer to $\widetilde{\mathbf{C}}^\mathsf{T} = [\widetilde{\mathbf{C}}^1, \widetilde{\mathbf{C}}^2, \cdots \widetilde{\mathbf{C}}^P]$ as the *encoding function*. The master node collects $\widetilde{\mathbf{C}}^p$'s from a subset of worker nodes, referred to as *non-stragglers*, and attempts to recover the matrix $\mathbf{C}$ from the results of the non-straggling worker nodes using a decoding function f. Given an encoding function $\widetilde{\mathbf{C}}$ and a decoding function f, the *recovery threshold* is defined as the minimum integer N such that the master node can recover the matrix $\mathbf{C}$ from the results of any N (non-straggling) worker nodes.

The goal is to design an encoding function and a decoding function so as to minimize the recovery threshold.

III. Proposed Coding Schemes

A. LT-Coded Distributed Matrix Multiplication

In this section, we propose *Factored LT (FLT) codes*, which, in this paper, will be used as a component of the distributed matrix multiplication scheme proposed in Section III-B. The application of FLT codes as a standalone scheme for distributed matrix multiplication is left for future work.

In the following, we briefly describe the encoding and decoding of FLT codes.

1) Encoding: LT codes, introduced by Luby in [19], are a class of rateless erasure codes that can be used to generate a (potentially infinite) sequence of output symbols from K source symbols. The number of source symbols involved in generating an output symbol is referred to as the *degree of the output symbol*. The degree of output symbols follow a degree distribution $\Omega(x)$. In [19], it was shown that in the case of a single erasure channel, the source symbols can be recovered from any $N = K(1 + \epsilon)$ output symbols with high probability, where ϵ is the overhead. In particular, as shown in [19], for some proper choice of degree distribution $\Omega(x)$, the overhead ϵ vanishes as K grows unbounded. To apply LT codes to the task of matrix-matrix multiplication, we treat $\mathbf{A}_i^\mathsf{T}\mathbf{B}_j$'s for $i \in [m]$ and $j \in [n]$ as source symbols, where $\mathbf{A}_i$'s and $\mathbf{B}_j$'s are component matrices of $\mathbf{A}$ and $\mathbf{B}$, respectively. As described in Section II, the matrix-matrix multiplication problem requires each output symbol to be the product of $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{B}}$, where $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{B}}$ are sum of randomly chosen chunks of $\mathbf{A}$ and $\mathbf{B}$, respectively. Unlike the case of a single erasure channel, for the matrix-matrix multiplication problem a degree-d output symbol cannot be generated from any arbitrary d source symbols. Thus, the encoding of LT codes is not directly applicable to our setting.

Let $\Omega(x) = \sum_{i=1}^{K} \Omega_i x^i$ be a degree distribution, where $\Omega_i = 0$ for all prime $i > \max(m, n)$. The encoding process performed by the master node for each worker node $p \in [P]$ is described as follows:

1. Randomly choose a degree d by sampling from the degree distribution $\Omega(x)$.

2. Randomly choose a divisor, denoted by d_1, of d in such a way that $d_1 \leq m$ and $d_2 = \frac{d}{d_1} \leq n$.
3. Choose $\mathcal{I}_{\mathbf{A}}^p$ and $\mathcal{I}_{\mathbf{B}}^p$ uniformly randomly from $\mathcal{S}_m^p$ and $\mathcal{S}_n^p$, respectively, where $\mathcal{S}_m^p$ (or $\mathcal{S}_n^p$) is the collection of all subsets of $[m]$ (or $[n]$) that are of size d_1 (or d_2).
4. Compute $(\widetilde{\mathbf{A}}^p)^\mathsf{T}$ and $\widetilde{\mathbf{B}}^p$ as in (1), and send them to the worker node p; and request the worker node p to compute $\widetilde{\mathbf{C}}^p = (\widetilde{\mathbf{A}}^p)^\mathsf{T}\widetilde{\mathbf{B}}^p$, and send it back.

The encoded symbols $\widetilde{\mathbf{C}}^p$'s for $p \in [P]$ constitute a codeword of the FLT code.

Example 1: Let $m = 3$, $n = 3$, and $\Omega(x) = 0.2x + 0.7x^2 + 0.1x^4$. Consider the generation of an encoded symbol at the worker node p. The master node randomly samples a degree d from $\Omega(x)$, say $d = 4$. Then, the master node randomly chooses a divisor d_1 of $d = 4$, say $d_1 = 2$, which implies that $d_2 = \frac{d}{d_1} = 2$. Then, for this example, $\mathcal{S}_m^p = \mathcal{S}_n^p = \{\{1,2\},\{1,3\},\{2,3\}\}$. The master node then chooses $\mathcal{I}_{\mathbf{A}}^p$ and $\mathcal{I}_{\mathbf{B}}^p$ uniformly at random from $\mathcal{S}_m^p$ and $\mathcal{S}_n^p$, respectively; say, $\mathcal{I}_{\mathbf{A}}^p = \{2,3\}$ and $\mathcal{I}_{\mathbf{B}}^p = \{1,3\}$. For simplicity, in this example, the coefficients a_i^p's and b_j^p's are chosen as follows:

$$a_i^p = \begin{cases} 1, & \text{if } i \in \mathcal{I}_{\mathbf{A}}^p \\ 0, & \text{Otherwise} \end{cases}, \quad b_j^p = \begin{cases} 1, & \text{if } j \in \mathcal{I}_{\mathbf{B}}^p \\ 0, & \text{Otherwise,} \end{cases}$$

The master node computes $(\widetilde{\mathbf{A}}^p)^\mathsf{T} = \mathbf{A}_2^\mathsf{T} + \mathbf{A}_3^\mathsf{T}$ and $\widetilde{\mathbf{B}}^p = \mathbf{B}_1 + \mathbf{B}_3$, and sends them to the worker node p. The worker node p is tasked to compute $\widetilde{\mathbf{C}}^p = (\widetilde{\mathbf{A}}^p)^\mathsf{T}\widetilde{\mathbf{B}}^p = (\mathbf{A}_2^\mathsf{T} + \mathbf{A}_3^\mathsf{T})(\mathbf{B}_1 + \mathbf{B}_3)$ and send it back to the master node. Notice that $\widetilde{\mathbf{C}}^p = \mathbf{A}_2^\mathsf{T}\mathbf{B}_1 + \mathbf{A}_2^\mathsf{T}\mathbf{B}_3 + \mathbf{A}_3^\mathsf{T}\mathbf{B}_1 + \mathbf{A}_3^\mathsf{T}\mathbf{B}_3$ is a linear combination of $d = 4$ source symbols, as desired. The encoding process is illustrated in Fig. 1.

2) Decoding: Without loss of generality, suppose that the master node collects results from the first N workers with $N \leq P$. Given the above encoding scheme, we have

$$\begin{bmatrix} \widetilde{\mathbf{C}}^1 \\ \widetilde{\mathbf{C}}^2 \\ \vdots \\ \widetilde{\mathbf{C}}^N \end{bmatrix} = \underbrace{\begin{bmatrix} a_1^1 b_1^1 & a_1^1 b_2^1 & \dots & a_m^1 b_n^1 \\ a_1^2 b_1^2 & a_1^2 b_1^2 & \dots & a_m^2 b_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^N b_1^N & a_1^N b_2^N & \dots & a_m^N b_n^N \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} \mathbf{A}_1^\mathsf{T}\mathbf{B}_1 \\ \mathbf{A}_1^\mathsf{T}\mathbf{B}_2 \\ \vdots \\ \mathbf{A}_m^\mathsf{T}\mathbf{B}_n \end{bmatrix}$$

We use $\mathbf{M} \in \mathbb{R}^{N \times mn}$ 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 $\mathbf{M}$ has full column rank. Given such M, the master node then uses a peeling decoding algorithm, which can be described using a bipartite graph as follows. The bipartite graph is constructed with one part of the nodes (referred to as the *left nodes*) being the set of source symbols $\mathbf{A}_i^\mathsf{T}\mathbf{B}_j$, $i \in [m]$ and $j \in [n]$, and the other part (referred to as the *right nodes*) being the set of output symbols $\widetilde{\mathbf{C}}^p$ for $p \in [N]$. There is an edge between the vertices $\mathbf{A}_i^\mathsf{T}\mathbf{B}_j$ and $\widetilde{\mathbf{C}}^p$ if $\mathbf{A}_i^\mathsf{T}\mathbf{B}_j$ is involved in the computation of $\widetilde{\mathbf{C}}^p$. The peeling decoding algorithm works in iterations. In each iteration, the master node searches for a right node of degree 1. If such a node cannot be found, the decoding algorithm terminates. Otherwise, the master node recovers the (unique) left node connected to the found degree-1 right node, and removes all edges adjacent to the recovered left node. The master node runs this process iteratively until all left nodes are recovered or no degree-1 right node can be found. An early termination of the decoding process (before all left nodes are recovered) yields a decoding failure; otherwise, the decoding process is dubbed successful.

B. Raptor-Coded Distributed Matrix Multiplication

In the FLT coding scheme presented in section III-A.1, the output symbols are generated by taking a linear combination of randomly chosen source symbols. Therefore, when the number of encoded symbols are close to the number of source symbols, a small fraction of source symbols remain uncovered by any output symbols, which means there exists a fraction of all zero columns in the coefficient matrix $\mathbf{M}$. These uncovered source symbols cannot be recovered at the end of the peeling decoding process. This implies that FLT codes do not perform well for moderate values of N. This is a well known issue with LT codes [19]. To address this issue, we propose a Raptor code based scheme, termed *Factored Raptor (FR) codes*, which is described below.

1) Encoding: An FR code is an FLT code concatenated with an outer code. In the case of a single erasure channel, the source symbols are encoded using the outer code, and the input symbols of the LT code are formed by a codeword of the outer code. Unlike the case of a single erasure channel, the outer code of FR codes cannot be chosen arbitrarily, due to the structure imposed by the matrix-matrix multiplication problem. In distributed matrix multiplication problem, the worker nodes are asked to compute product of a linear combination of chunks of one matrix with that of the other matrix. Collection of these products must form a codeword of the FLT code, and the input symbols corresponding to the codeword of the FLT code must form a codeword of the outer code. This requirement is met by encoding $\mathbf{A}$ and $\mathbf{B}$ with an MDS code and sending the linear combination of chunks of the respective encoded matrix to the worker nodes. We briefly describe the encoding process below.

1. For $\tilde{m} \geq m$ and $\tilde{n} \geq n$ such that $P = \tilde{m}\tilde{n}$, the master node encodes the matrix $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_m]$ or $\mathbf{B} = [\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_n]$ using an $(\tilde{m}, m)$ or $(\tilde{n}, n)$ MDS code to obtain the coded matrix $\widetilde{\mathbf{A}} = [\mathbf{A}_1, \widetilde{\mathbf{A}}_2, \cdots \widetilde{\mathbf{A}}_{\tilde{m}}]$ or $\widetilde{\mathbf{B}} = [\mathbf{B}_1, \widetilde{\mathbf{B}}_2, \cdots \widetilde{\mathbf{B}}_{\tilde{n}}]$, respectively.
2. The master node then encodes $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{B}}$ according to the degree distribution of an FLT code as described in Section III-A.1.

By the above construction, $\widetilde{\mathbf{C}}$ is a codeword of the FLT code, and $\mathbf{A}_i\mathbf{B}_j$'s for $i \in [\tilde{m}]$ and $j \in [\tilde{n}]$ form a codeword of an $(\tilde{m}, m) \times (\tilde{n}, n)$ outer Product code.

2) Decoding: Decoding of FR codes consists of alternating decoding iterations on the FLT code and the outer Product code. At the master node, encoding of FR codes induces a Tanner graph which has $\mathbf{A}_i\mathbf{B}_j$'s for $i \in [\tilde{m}]$ and $j \in [\tilde{n}]$ as the input symbols (i.e., left nodes) and $\widetilde{\mathbf{C}}^p$'s as the output symbols (i.e., right nodes). Decoding iterations of

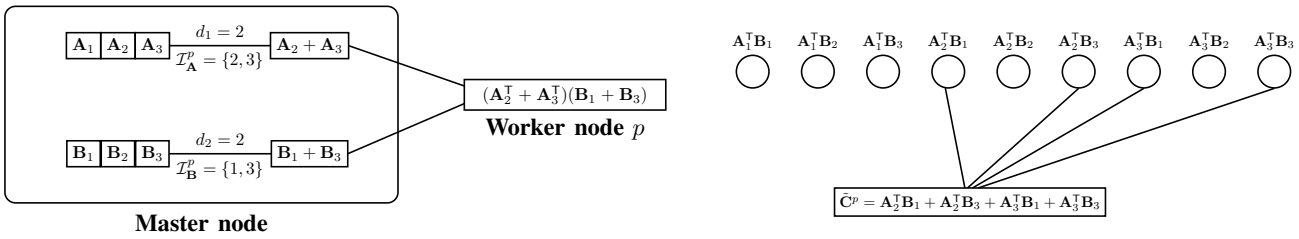


Fig. 1: Encoding of a degree-4 node

the FLT code proceed on this Tanner graph as described in Section III-A.2. The matrix $\mathbf{U} = \widetilde{\mathbf{A}}^\mathsf{T}\widetilde{\mathbf{B}}$ is a codeword of the $(\tilde{m}, m) \times (\tilde{n}, n)$ outer Product code. Each row and column of $\mathbf{U}$ is a codeword of the $(\tilde{n}, n)$ MDS code and the $(\tilde{m}, m)$ MDS code, respectively. Decoding of the outer Product code proceeds by decoding the component MDS codes.

Example 2: We now illustrate encoding of an FR code for computing the product of two matrices $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2]$ and $\mathbf{B} = [\mathbf{B}_1, \mathbf{B}_2]$. We consider a $(3, 2) \times (3, 2)$ Product code as the outer code with a $(3, 2)$ MDS code as the component code, and a $(10, 9)$ FLT code with degree distribution $\Omega(x) = 0.2x + 0.7x^2 + 0.1x^4$. Encoding of the outer Product code is done by applying the $(3, 2)$ MDS code to both $\mathbf{A}$ and $\mathbf{B}$ to obtain $\widetilde{\mathbf{A}} = [\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]$ and $\widetilde{\mathbf{B}} = [\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3]$, respectively. Using the same procedure as in Example 1 for generating an output symbol of an FLT code, for this example suppose each worker node p for $p \in [10]$ is requested to compute $\widetilde{\mathbf{C}}^p$ as follows:

$$\begin{aligned}
\widetilde{\mathbf{C}}^1 &= \mathbf{A}_1^\mathsf{T}\mathbf{B}_2, & \widetilde{\mathbf{C}}^2 &= \mathbf{A}_1^\mathsf{T}\mathbf{B}_3,\\
\widetilde{\mathbf{C}}^3 &= (\mathbf{A}_1^\mathsf{T} + \mathbf{A}_2^\mathsf{T})\mathbf{B}_1, & \widetilde{\mathbf{C}}^4 &= (\mathbf{A}_1^\mathsf{T} + \mathbf{A}_3^\mathsf{T})\mathbf{B}_1,\\
\widetilde{\mathbf{C}}^5 &= (\mathbf{A}_1^\mathsf{T} + \mathbf{A}_2^\mathsf{T})\mathbf{B}_3, & \widetilde{\mathbf{C}}^6 &= (\mathbf{A}_1^\mathsf{T} + \mathbf{A}_3^\mathsf{T})\mathbf{B}_3,\\
\widetilde{\mathbf{C}}^7 &= \mathbf{A}_1^\mathsf{T}(\mathbf{B}_1 + \mathbf{B}_2), & \widetilde{\mathbf{C}}^8 &= \mathbf{A}_2^\mathsf{T}(\mathbf{B}_2 + \mathbf{B}_3),\\
\widetilde{\mathbf{C}}^9 &= \mathbf{A}_3^\mathsf{T}(\mathbf{B}_1 + \mathbf{B}_2), & \widetilde{\mathbf{C}}^{10} &= (\mathbf{A}_2^\mathsf{T} + \mathbf{A}_3^\mathsf{T})(\mathbf{B}_2 + \mathbf{B}_3).
\end{aligned}$$

Example 3: In this example, we describe the decoding algorithm at the master node using the code described in Example 2. For this example, consider that the master node collects results from the worker nodes $\{1, 3, 5, 7\}$. The peeling decoding is run on the subgraph, denoted by G_0, induced by the worker nodes $\{1, 3, 5, 7\}$. In iteration 0, the source symbol $\mathbf{A}_1^\mathsf{T}\mathbf{B}_2$ is recovered from worker 1 since $\widetilde{\mathbf{C}}^1$ is a degree-1 node in G_0. Peel the edges connected to the source symbol $\mathbf{A}_1^\mathsf{T}\mathbf{B}_2$ in G_0 and denote the residual graph by G_1. In iteration 1, the source symbol $\mathbf{A}_1^\mathsf{T}\mathbf{B}_1 = \widetilde{\mathbf{C}}^7 - \mathbf{A}_1^\mathsf{T}\mathbf{B}_2$ is recovered from worker 1 since $\widetilde{\mathbf{C}}^1$ is a degree-1 node in G_1. Similarly, the source symbol $\mathbf{A}_2^\mathsf{T}\mathbf{B}_1$ is recovered from worker 3 in iteration 3. Since there are no degree-1 symbols left after iteration 3, the decoder of the FLT code cannot proceed further. Decoding of the FR code can proceed further by decoding the outer Product code. Arrange the input symbols to the FLT code in a 3×2 matrix as shown in Fig. 2e. Both first row and first column have two computation results as shown in Fig. 2e. In iteration 4, the missing computation results in both first row and first column are recovered by decoding the $(3, 2)$ MDS code. The first few iterations of the decoding process for this example are illustrated in Fig. 2.

3) Optimal Decoding by Inactivation Decoding: Maximum likelihood decoding of FLT and FR codes can be implemented using a low-complexity algorithm known as *inactivation decoding* [21], which is a combination of peeling decoding and matrix inversion. The inactivation decoding starts with peeling decoding, which continues until it encounters a stopping set. At this point, the decoder assigns a variable to the value of an unrecovered symbol, referred to as an *inactivated symbol*. Then, the peeling decoder starts again and computes the unrecovered symbols in terms of the variable representing the value of the inactivated symbol. This procedure is repeated again, when the peeling decoder is stuck, by choosing another inactivated symbol from the unrecovered input symbols. The decoding stops when all the input symbols are either recovered or inactivated. Finally, optimal decoding of the inactivated symbols is performed via Gaussian elimination, and the decoded values are back-substituted into the decoded input symbols which depend on them.

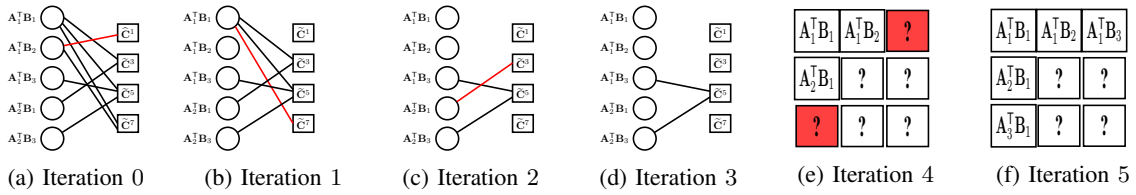
IV. Complexity Considerations

A. Encoding Complexity

In our encoding scheme, the master node sends two matrices $\widetilde{\mathbf{A}}^p$ and $\widetilde{\mathbf{B}}^p$ to each worker node p, and each worker node computes only one product. Therefore, the computation and communication costs of our proposed scheme are the same as those of Polynomial codes and Product codes. Unlike the FLT coding scheme, the LT coding scheme proposed in [20] requires the master node to send (on average) $\log K$ chunks of the input matrices to each worker node, and each worker node must compute (on average) $\log K$ products.

B. Decoding Complexity

In the following, we briefly describe the decoding complexity of FR codes. The peeling of every edge in the Tanner graph corresponds to performing $\frac{rt}{K}$ operations. Let d_{avg} be the average degree of output degree distribution. Then, each block $\mathbf{A}_i^\mathsf{T}\mathbf{B}_j$ will be involved in $\mathcal{O}(\frac{d_{\text{avg}}N}{K})$ such operations on average. We are interested in the case when $\tilde{m} - m$ and $\tilde{n} - n$ are very small and hence, the decoding complexity of the outer code is negligible when compared to that of the LT part of the FR code. In addition, we are interested in a regime where r and t are very large and hence, the approximate complexity of the decoding algorithm is $\mathcal{O}(rt\frac{d_{\text{avg}}N}{K})$.



(a) Iteration 0 (b) Iteration 1 (c) Iteration 2 (d) Iteration 3 (e) Iteration 4 (f) Iteration 5

Fig. 2: Illustration of the decoding algorithm for FR codes

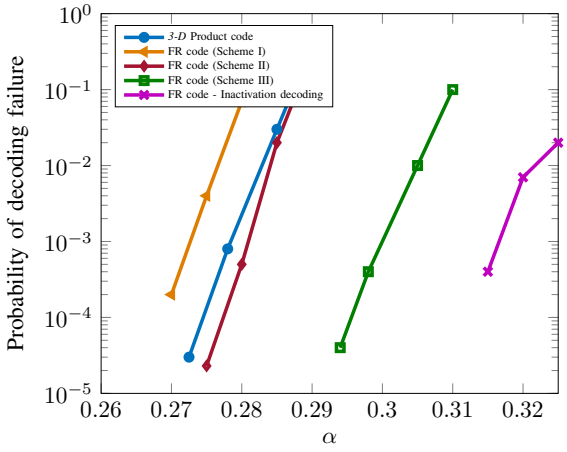


Fig. 3: Probability of decoding failure versus fraction of straggling workers $\alpha = \frac{P-N}{P}$.

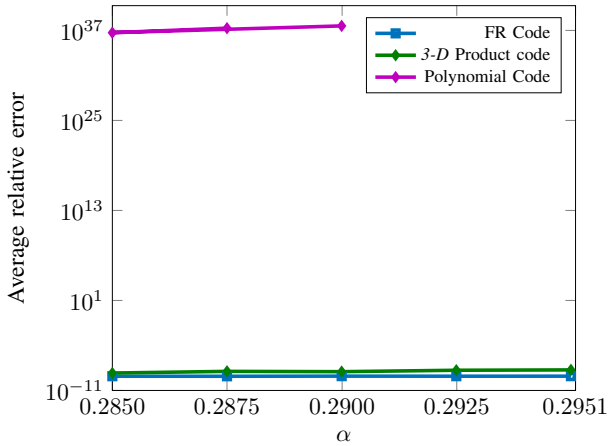


Fig. 4: Average relative error versus fraction of straggling workers $\alpha = \frac{P-N}{P}$.

V. Simulation Results

First, we present simulation results to compare the recovery threshold of FR codes and *3-D* Product codes. For the FR code, we set $m = n = 80$ and choose the entries of $\mathbf{A}$ and $\mathbf{B}$ to be realizations of an i.i.d. Gaussian random variable with zero mean and unit variance. We encode the source symbols $\mathbf{A}_i^{\mathsf{T}}$, for $i \in [m]$ and $j \in [n]$, using an $(82, 80) \times (82, 80)$ Product code, where an $(82, 80)$ systematic RKRP code [15] is used as the row and column code. The output symbols from the encoder of the Product code are then encoded using a $(10000, 6724)$ FLT code with right degree distribution $\Omega(x) = 0.013x + 0.5x^2 + 0.1661x^3 + 0.0726x^4 + 0.0826x^5 + 0.0581x^8 + 0.0340x^9 + 0.0576x^{18} + 0.0160x^{66}$. As discussed in Section III-A.1, the number of source symbols is $K = mn = 6400$. For the chosen values of $K = 6400$ and $P = 10000$, the maximum fraction of straggling workers that any scheme can tolerate is given by $\frac{P-K}{P} = 0.36$ [1]. Note that Polynomial codes achieve this threshold, and have zero probability of decoding failure for any $\alpha <= 0.36$ where $\alpha = \frac{P-N}{P}$ denotes the fraction of straggling workers. In our simulations, for each value of α being considered, the αP straggling workers were chosen uniformly at random.

To illustrate the superiority of the encoding scheme in Section III-A.1, we generate coefficient matrix $\mathbf{M}$ in three different ways by altering the second step of encoding as follows: (Scheme I) Fix $d_1 = d$ and $d_2 = 1$; (Scheme II) Define a uniform random variable i which takes value from the set $\mathcal{I} = \{1, 2\}$. Fix $d_i = d$ and $d_{\mathcal{I}\setminus\{i\}} = 1$; (Scheme III) Follow the encoding process in Section III-A.1. For each of these encoding schemes, in Fig. 3, we have plotted the probability that the master node is unable to recover the output matrix $\mathbf{C}$ after receiving computations from αP fraction of workers. We have also plotted the decoding failure of a $(21, 18) \times (22, 19) \times (22, 19)$ Product code in Fig. 3 for comparison. We observe that the encoding scheme described in Section III-A.1 (Scheme III) has better recovery threshold when compared to the other two encoding schemes (Schemes I and II). In addition, the recovery threshold of FR codes (Scheme III) is higher than that of *3-D* Product codes. We have also plotted the error performance of the FR codes under inactivation decoding as described in Section III-B.3. One can see that the recovery threshold under inactivation decoding, instead of using only peeling decoding, is higher.

Next, for the chosen parameters above, we demonstrate the numerical stability of the proposed FR codes by simulating the *average relative error* defined as $\eta_{\text{ave}} := \mathbb{E}\left[||\mathbf{C} - \widehat{\mathbf{C}}||_2/||\mathbf{C}||_2\right]$, where $\widehat{\mathbf{C}}$ is the estimate of the output matrix $\mathbf{C}$. In Fig. 4, we plot the average relative error versus fraction of straggling workers. While computing average relative error, we do not account for instances of decoding failure due to stopping sets. It can be seen that the average relative error of the simulated FR code is around 10^{-9} and does not increase as the fraction of straggling workers increases. It is also observed that *3-D* product codes and FR codes have comparable average relative error, see Fig. 4; whereas the FR codes have a better recovery threshold than *3-D* product codes, see Fig. 3. In Fig. 4, it can also be seen that the average relative error of Polynomial codes is substantially higher than that of product codes and FR codes.

REFERENCES

[1] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017.

[2] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2022–2026.

[3] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1585–1589.

[4] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication," *IEEE International Symposium on Information Theory, ISIT*, 2019.

[5] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[6] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[7] S. Dutta, V. Cadambe, and P. Grover, ""short-dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6171–6193, 2019.

[8] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *2016 IEEE Globecom Workshops*, Dec 2016, pp. 1–6.

[9] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 2418–2422.

[10] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded distributed computing: Straggling servers and multistage dataflows," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2016, pp. 164–171.

[11] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 709–719.

[12] A. B. Das and A. Ramamoorthy, "Distributed matrix-vector multiplication: A convolutional coding approach," *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 3022–3026, 2019.

[13] V. Y. Pan, "How bad are Vandermonde matrices?" *SIAM J. Matrix Analysis Applications*, vol. 37, pp. 676–694, 2015.

[14] M. Fahim and V. Cadambe, "Numerically stable polynomially coded computing," in *to appear in proceedings of the International Symposium on Information Theory*, 2019.

[15] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Random khatri-rao-product codes for numerically-stable distributed matrix multiplication," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019, pp. 253–259.

[16] A. Severinson, A. Graell i Amat, and E. Rosnes, "Block-diagonal and LT codes for distributed computing with straggling servers," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 1739–1753, March 2019.

[17] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1993–1997.

[18] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 8192–8196.

[19] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, Proceedings.*, Nov 2002, pp. 271–280.

[20] S. Wang, J. Liu, and N. B. Shroff, "Coded sparse matrix multiplication," *CoRR*, vol. abs/1802.03430, 2018. [Online]. Available: http://arxiv.org/abs/1802.03430

[21] D. Burshtein and G. Miller, "Efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2837–2844, Nov.