Straggler Mitigation in Distributed Matrix Multiplication: Fundamental Limits and Optimal Coding

Qian Yu, *Student Member, IEEE*, Mohammad Ali Maddah-Ali, *Member, IEEE*, and A. Salman Avestimehr, *Fellow, IEEE*

***Abstract*— We consider the problem of massive matrix multiplication, which underlies many data analytic applications, in a large-scale distributed system comprising a group of worker nodes. We target the stragglers' delay performance bottleneck, which is due to the unpredictable latency in waiting for slowest nodes (or stragglers) to finish their tasks. We propose a novel coding strategy, named *entangled polynomial code*, for designing the intermediate computations at the worker nodes in order to minimize the recovery threshold (i.e., the number of workers that we need to wait for in order to compute the final output). We demonstrate the optimality of entangled polynomial code in several cases, and show that it provides orderwise improvement over the conventional schemes for straggler mitigation. Furthermore, we characterize the optimal recovery threshold among all linear coding strategies within a factor of 2 using *bilinear complexity*, by developing an improved version of the entangled polynomial code. In particular, while evaluating bilinear complexity is a well-known challenging problem, we show that optimal recovery threshold for linear coding strategies can be approximated within a factor of 2 of this fundamental quantity. On the other hand, the improved version of the entangled polynomial code enables further and orderwise reduction in the recovery threshold, compared to its basic version. Finally, we show that the techniques developed in this paper can also be extended to several other problems such as coded convolution and fault-tolerant computing, leading to tight characterizations.**

***Index Terms*— Distributed computing, matrix multiplication, straggler mitigation, coded computing.**

I. Introduction

MATRIX multiplication is one of the key operations underlying many data analytics applications in various fields such as machine learning, scientific computing, and graph processing. Many such applications require processing terabytes or even petabytes of data, which needs massive computation and storage resources that cannot be provided by a single machine. Hence, deploying matrix computation tasks on large-scale distributed systems has received wide interests [2]–[5].

There is, however, a major performance bottleneck that arises as we scale out computations across many distributed nodes: stragglers' delay bottleneck, which is due to the unpredictable latency in waiting for slowest nodes (or stragglers) to finish their tasks [6]. The conventional approach for mitigating straggler effects involves injecting some form of "computation redundancy" such as repetition (e.g., [7]). Interestingly, it has been shown recently that *coding theoretic* concepts can also play a transformational role in this problem, by efficiently creating "computational redundancy" to mitigate the stragglers [8]–[14].

In this paper, we consider a general formulation of distributed matrix multiplication, study information-theoretic limits, and develop optimal coding designs for straggler effect mitigation. We consider the canonical master-worker distributed setting, where a group of N workers aim to collaboratively compute the product of two large matrices A and B, and return the result $C = A^\intercal B$ to the master. As shown in Figure 1, the two input matrices are partitioned (arbitrarily) into p-by-m and p-by-n blocks of submatrices respectively, where all submatrices within the same input are of equal size. Each worker has a local memory that can be used to store any coded function of each matrix, denoted by $\tilde{A}_i$'s and $\tilde{B}_i$'s, each with a size equal to that of the corresponding submatrices. The workers then multiply their two stored (coded) submatrices

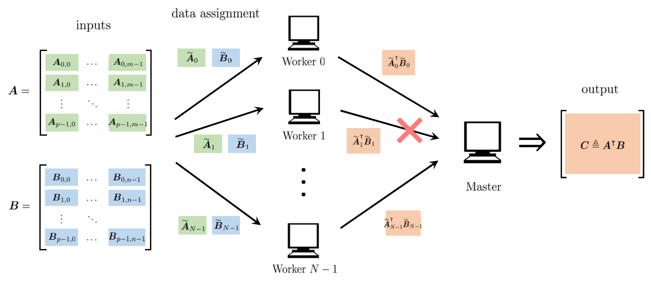
inputs
data assignment
$A =$
$A_{0,0}$
$A_{0,m-1}$
$A_{1,0}$
$A_{1,m-1}$
$A_{p-1,0}$
$A_{p-1,m-1}$
$B =$
$B_{0,0}$
$B_{0,n-1}$
$B_{1,0}$
$B_{1,n-1}$
$B_{p-1,0}$
$B_{p-1,n-1}$
$\tilde{A}_0$ $\tilde{B}_0$
$\tilde{A}_1$ $\tilde{B}_1$
$\tilde{A}_{N-1}$ $\tilde{B}_{N-1}$
Worker 0
Worker 1
Worker $N-1$
$\tilde{A}_0^\intercal \tilde{B}_0$
$\tilde{A}_1^\intercal \tilde{B}_1$
$\tilde{A}_{N-1}^\intercal \tilde{B}_{N-1}$
Master
output
$C \triangleq A^\intercal B$

Fig. 1. Overview of the distributed matrix multiplication problem. Each worker computes the product of the two stored encoded submatrices ($\tilde{A}_i$ and $\tilde{B}_i$) and returns the result to the master. By carefully designing the coding strategy, the master can decode the multiplication result of the input matrices from a subset of workers, without having to wait for stragglers (worker 1 in this example).

Manuscript received January 23, 2018; revised December 12, 2019; accepted December 12, 2019. Date of publication January 3, 2020; date of current version February 14, 2020. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract HR001117C0053, in part by the ONR under Award N000141612189, and in part by the NSF under Grant CCF-1703575 and Grant NeTS-1419632. This article was presented at the ISIT 2018.

Qian Yu and A. Salman Avestimehr are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: qyu880@usc.edu; avestimehr@ee.usc.edu).

Mohammad Ali Maddah-Ali is with the Department of Electrical Engineering, Sharif University of Technology, Tehran 11365, Iran (e-mail: maddah_ali@sharif.edu).

Communicated by K. Narayanan, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2019.2963864

and return the results to the master. By carefully designing the coding functions, the master can decode the final result without having to wait for the slowest workers, which provides robustness against stragglers.

Note that by allowing different values of parameters p, m, and n, we allow flexible partitioning of input matrices, which in return enables different utilization of system resources (e.g., the required amount of storage at each worker and the amount of communication from worker to master).[1] Hence, considering the system constraints on available storage and communication resources, one can choose p, m, and n accordingly. We aim to find optimal coding and computation designs for *any* choice of parameters p, m and n, to provide optimum straggler effect mitigation for various situations.

With a careful design of the coded submatrices $\tilde{A}_i$ and $\tilde{B}_i$ at each worker, the master only needs results from the fastest workers before it can recover the final output, which effectively mitigates straggler issues. To measure the robustness against straggler effects of a given coding strategy, we use the metric *recovery threshold*, defined previously in [11], which is equal to the minimum number of workers that the master needs to wait for in order to compute the output C. Given this terminology, our main problem is as follows: What is the minimum possible recovery threshold and the corresponding coding scheme, for any choice of parameters p, m, n, and N?

We propose a novel coding technique, referred to as *entangled polynomial code*, which achieves the recovery threshold of $pmn + p - 1$ for all possible parameter values. The construction of the entangled polynomial code is based on the observation that when multiplying an m-by-p matrix and a p-by-n matrix, we essentially evaluate a subspace of bilinear functions, spanned by the pairwise product of the elements from the two matrices. Although potentially there are a total of p^2mn pairs of elements, at most pmn pairs are directly related to the matrix product, which is an order of p less. The particular structure of the proposed code entangles the input matrices to the output such that the system almost avoids unnecessary multiplications and achieves a recovery threshold in the order of pmn, while allowing robust straggler mitigation for arbitrarily large systems. This allows orderwise improvement upon conventional uncoded approaches, random linear codes, and MDS-coding type approaches for straggler mitigation [8], [9].

Entangled polynomial code generalizes our previously proposed polynomial code for distributed matrix multiplication [11], which was designed for the special case of $p = 1$ (i.e., allowing only column-wise partitioning of matrices A and B). However, as we move to arbitrary partitioning of the input matrices (i.e., arbitrary values of m, n, and p), a key challenge is to design the coding strategy at each worker such that its computation best *aligns* with the final computation C. In particular, to recover the product C, the master needs mn components that each involve summing p products of submatrices of A and B. Entangled polynomial code effectively aligns the workers' computations with the master's need, which is its key distinguishing feature from polynomial code.

We show that entangled polynomial code achieves the optimal recovery threshold among all linear coding strategies in the cases of $m = 1$ or $n = 1$. It also achieves the optimal recovery threshold among all possible schemes within a factor of 2 when $m = 1$ or $n = 1$.

Furthermore, for *all* partitionings of input matrices (i.e., all values of p, m, n, and N), we characterize the optimal recovery threshold among all linear coding strategies within a factor of 2 of $R(p, m, n)$, which denotes the *bilinear complexity* of multiplying an m-by-p matrix to a p-by-n matrix (see Definition 3 later in the paper). While evaluating bilinear complexity is a well-known challenging problem in the computer science literature (see [15]), we show that the optimal recovery threshold for linear coding strategies can be approximated within a factor of 2 of this fundamental quantity.

We establish this result by developing an improved version of the entangled polynomial code, which achieves a recovery threshold of $2R(p, m, n) - 1$. Specifically, this coding construction exploits the fact that any matrix multiplication problem can be converted into a problem of computing the element-wise product of two arrays of length $R(p, m, n)$. Then we show that this augmented computing task can be optimally handled using a variation of the entangled polynomial code, and the corresponding optimal code achieves the recovery threshold $2R(p, m, n) - 1$.

Finally, we show that the coding construction and converse bounding techniques developed for proving the above results can also be directly extended to several other problems. For example, we show that the converse bounding technique can be extended to the problem of coded convolution, which was originally considered in [16]. We prove that the state-of-the-art scheme we proposed in [11] for this problem is in fact optimal among all linear coding schemes. These techniques can also be applied in the context of fault-tolerant computing, which was first studied in [17] for matrix multiplication. We provide tight characterizations on the maximum number of detectable or correctable errors.

We note that recently, another computation design named PolyDot was also proposed for distributed matrix multiplication, achieving a recovery threshold of $m^2(2p - 1)$ for $m = n$ [18]. Both entangled polynomial code and PolyDot are developed by extending the polynomial codes proposed in [11] to allow arbitrary partitioning of input matrices. Compared with PolyDot, entangled polynomial code achieves a strictly smaller recovery threshold of $pmn + p - 1$, by a factor of 2. More importantly, in this paper we have developed a converse bounding technique that proves the optimality of the entangled polynomial code in several cases. We have also proposed an improved version of the entangled polynomial code and characterized the optimum recovery threshold within a factor of 2 for all parameter values.

II. System Model and Problem Formulation

We consider a problem of matrix multiplication with two input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$, for some integers r, s, t and a sufficiently large field $\mathbb{F}$.[2] We are interested in

[1]A more detailed discussion is provided in Remark 3

[2]Here we consider the general class of fields, which includes finite fields, the field of real numbers, and the field of complex numbers.

computing the product $C \triangleq A^\intercal B$ in a distributed computing environment with a master node and N worker nodes, where each worker can store $\frac{1}{pm}$ fraction of A and $\frac{1}{pn}$ fraction of B, based on some integer parameters p, m, and n (see Fig. 1).

Specifically, each worker i can store two coded matrices $\tilde{A}_i \in \mathbb{F}^{\frac{s}{p} \times \frac{r}{m}}$ and $\tilde{B}_i \in \mathbb{F}^{\frac{s}{p} \times \frac{t}{n}}$, computed based on A and B respectively. Each worker can compute the product $\tilde{C}_i \triangleq \tilde{A}_i^\intercal \tilde{B}_i$, and return it to the master. The master waits only for the results from a subset of workers before proceeding to recover the final output C using certain *decoding functions*.

Given the above system model, we formulate the *distributed matrix multiplication problem* based on the following terminology: We define the *computation strategy* as a collection of $2N$ *encoding functions*, denoted by

$$\boldsymbol{f} = (f_0, f_1, \ldots, f_{N-1}), \qquad \boldsymbol{g} = (g_0, g_1, \ldots, g_{N-1}), \quad (1)$$

that are used by the workers to compute each $\tilde{A}_i$ and $\tilde{B}_i$, and a class of *decoding functions*, denoted by

$$\boldsymbol{d} = \{d_\mathcal{K}\}_{\mathcal{K} \subseteq \{0,1,\ldots,N-1\}}, \quad (2)$$

that are used by the master to recover C given results from any subset $\mathcal{K}$ of the workers. Each worker i stores matrices

$$\tilde{A}_i = f_i(A), \qquad \tilde{B}_i = g_i(B), \quad (3)$$

and the master can compute an estimate $\hat{C}$ of matrix C using results from a subset $\mathcal{K}$ of the workers by computing

$$\hat{C} = d_\mathcal{K}\left(\{\tilde{C}_i\}_{i \in \mathcal{K}}\right). \quad (4)$$

For any integer k, we say a computation strategy is *k-recoverable* if the master can recover C given the computing results from *any* k workers. Specifically, a computation strategy is *k-recoverable* if for any subset $\mathcal{K}$ of k users, the final output $\hat{C}$ from the master equals C for all possible input values. We define the *recovery threshold* of a computation strategy, denoted by $K(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{d})$, as the minimum integer k such that computation strategy $(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{d})$ is k-recoverable.

We aim to find a computation strategy that requires the minimum possible recovery threshold and allows efficient decoding at the master. Among all possible computation strategies, we are particularly interested in a certain class of designs, referred to as the *linear codes* and defined as follows:

Definition 1: For a distributed matrix multiplication problem of computing $A^\intercal B$ using N workers, we say a computation strategy is a *linear code* given parameters p, m, and n, if there is a partitioning of the input matrices A and B where each matrix is divided into the following submatrices of equal sizes

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,m-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p-1,0} & A_{p-1,1} & \cdots & A_{p-1,m-1} \end{bmatrix}, \quad (5)$$

$$B = \begin{bmatrix} B_{0,0} & B_{0,1} & \cdots & B_{0,n-1} \\ B_{1,0} & B_{1,1} & \cdots & B_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{p-1,0} & B_{p-1,1} & \cdots & B_{p-1,n-1} \end{bmatrix}, \quad (6)$$

such that the encoding functions of each worker i can be written as

$$\tilde{A}_i = \sum_{j,k} A_{j,k} a_{ijk}, \qquad \tilde{B}_i = \sum_{j,k} B_{j,k} b_{ijk}, \quad (7)$$

for some tensors a and b, and the decoding function given each subset $\mathcal{K}$ can be written as[3]

$$\hat{C}_{j,k} = \sum_{i \in \mathcal{K}} \tilde{C}_i c_{ijk}, \quad (8)$$

for some tensor c. For brevity, we denote the set of linear codes as $\mathcal{L}$.

The major advantage of linear codes is that they guarantee that both the encoding and the decoding complexities of the scheme scale linearly with respect to the size of the input matrices. Furthermore, as we have proved in [11], linear codes are optimal for $p = 1$. Given the above terminology, we define the following concept.

Definition 2: For a distributed matrix multiplication problem of computing $A^\intercal B$ using N workers, we define the *optimum linear recovery threshold* as a function of the problem parameters p, m, n, and N, denoted by K^*_{linear}, as the minimum achievable recovery threshold among all linear codes. Specifically,

$$K^*_{\text{linear}} \triangleq \min_{(\boldsymbol{f},\boldsymbol{g},\boldsymbol{d}) \in \mathcal{L}} K(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{d}). \quad (9)$$

Our goal is to characterize the optimum linear recovery threshold K^*_{linear}, and to find computation strategies to achieve such optimum threshold. Note that if the number of workers N is too small, obviously no valid computation strategy exists even without requiring straggler tolerance. Hence, in the rest of the paper, we only consider the meaningful case where N is large enough to support at least one valid computation strategy. More concretely, we show that the minimum possible number of workers is given by a fundamental quantity: the bilinear complexity of multiplying an m-by-p matrix and a p-by-n matrix, which is formally introduced in Section III.

We are also interested in characterizing the minimum recovery threshold achievable using general coding strategies (including non-linear codes). Similar to [11], we define this value as the *optimum recovery threshold* and denote it by K^*.

III. Main Results

We state our main results in the following theorems:

Theorem 1: For a distributed matrix multiplication problem of computing $A^\intercal B$ using N workers, with parameters p, m, and n, the following recovery threshold can be achieved by a linear code, referred to as the entangled polynomial code.[4]

$$K_{\text{entangled-poly}} \triangleq pmn + p - 1. \quad (10)$$

Remark 1: Compared to some other possible approaches, our proposed entangled polynomial code provides orderwise

[3]Here $\hat{C}_{j,k}$ denotes the master's estimate of the subblock of C that corresponds to $\sum_\ell A_{\ell,j} B_{\ell,k}$.

[4]For $N < pmn + p - 1$, we define $K_{\text{entangled-poly}} \triangleq N$.

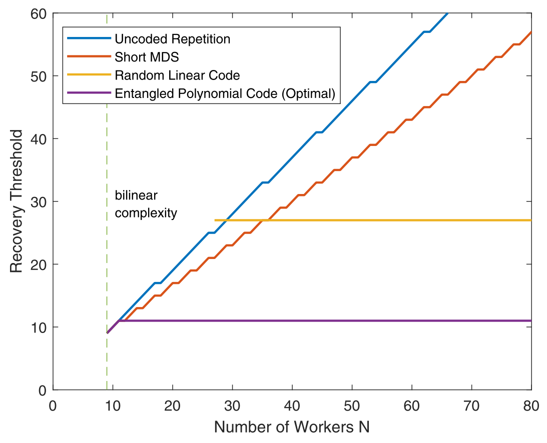

Fig. 2. Comparison of the recovery thresholds achieved by the uncoded repetition scheme, the random linear code, the short-MDS (or short-dot) [8], [9] and our proposed entangled polynomial code, given problem parameters $p = m = 3$, $n = 1$. The entangled polynomial code orderwise improves upon all other approaches. It also achieves the optimum linear recovery threshold in this scenario.

improvement in the recovery threshold (see Fig. 2). One conventional approach (referred to as the *uncoded repetition scheme*) is to let each worker store and multiply uncoded submatrices. With the additional computation redundancy through repetition, the scheme can robustly tolerate some stragglers. However, its recovery threshold, $K_{\text{uncoded}} \triangleq N - \lfloor \frac{N}{pmn} \rfloor + 1$, grows linearly with respect to the number of workers. Another approach is to let each worker store two random linear combinations of the input submatrices (referred to as the *random linear code*). With high probability, this achieves recovery threshold $K_{\text{RL}} \triangleq p^2mn$,[5] which does not scale with N. However, to calculate C, we need the result of at most pmn sub-matrix multiplications. Indeed, the lack of structure in the random coding forces the system to wait for p times more than what is essentially needed. One surprising aspect of the proposed entangled polynomial code is that, due to its particular structure which aligns the workers' computations with the master's need, it avoids unnecessary multiplications of submatrices. As a result, it achieves a recovery threshold that does not scale with N, and is orderwise smaller than that of the random linear code. Furthermore, it allows efficient decoding at the master, which requires at most an almost linear complexity.

Remark 2: There have been several works in prior literature investigating the $p = 1$ case [8], [11], [19]. For this special case, the entangled polynomial code reduces to our previously proposed polynomial code, which achieves the optimum recovery threshold mn and orderwise improves upon other designs. On the other hand, there has been some investigation on matrix-by-vector type multiplication [8], [9], which can be viewed as the special case of $m = 1$ or $n = 1$ in our proposed problem. The short-MDS code (or short-dot) has been proposed, achieving a recovery threshold of $N - \lfloor \frac{N}{p} \rfloor + m$, which scales linearly with N. Our proposed entangled polynomial code also strictly and orderwise improves upon that (see Fig. 2).

Remark 3: By selecting different values of parameters p, m, and n, the entangled polynomial code enables different utilization of the system resources, which allows for balancing the costs due to storage and communication. In particular, one can show that a distributed implementation for multiplying $A^\intercal \in \mathbb{F}^{r\times s}$ and $B \in \mathbb{F}^{s\times t}$ with parameters p, m, and n requires:

- Computation load at each worker (normalized by the cost of a single field operation): $O(\frac{srt}{pmn})$,
- Communication required from each worker (normalized by the size of C): $L \triangleq \frac{1}{mn}$,
- Storage allocated for storing each coded matrix (normalized by the sizes of A, B, respectively): $\mu_A \triangleq \frac{1}{pm}$, $\mu_B \triangleq \frac{1}{pn}$.

If we roughly fix the computation load (specifically, fixing pmn for the cubic matrix multiplication algorithm), the computing scheme requires the following trade-off between storage and communication:

$$L\mu_A\mu_B \sim \text{constant}. \tag{11}$$

By designing the values of p, m, and n, we can operate at different locations on this trade-off to account for the system's requirement,[6] while the entangled polynomial code maintains almost the same recovery threshold.

Our second result is the optimality of the entangled polynomial code when $m = 1$ or $n = 1$. Specifically, we prove that entangled polynomial code is optimal in this scenario among all linear codes. Furthermore, if the base field $\mathbb{F}$ is finite, it also achieves the optimum recovery threshold K^* within a factor of 2, with non-linear coding strategies taken into account.

Theorem 2: For a distributed matrix multiplication problem of computing $A^\intercal B$ using N workers, with parameters p, m, and n, if $m = 1$ or $n = 1$, we have

$$K^*_{\text{linear}} = K_{\text{entangled-poly}}. \tag{12}$$

Moreover, if the base field $\mathbb{F}$ is finite,

$$\frac{1}{2}K_{\text{entangled-poly}} < K^* \leq K_{\text{entangled-poly}}. \tag{13}$$

Remark 4: We prove Theorem 2 by first exploiting the algebraic structure of matrix multiplication to develop a linear algebraic converse for equation (12), and then constructing an information theoretic converse to prove inequality (13). The linear algebraic converse only relies on two properties of the matrix multiplication operation: 1) bilinearity, and

[5]Intuitively, because each worker returns a random linear combination of all p^2mn possible pairwise products, with high probability, the final output can be recovered from any subset of p^2mn results.

[6]For example, letting $p = 1$ minimizes the communication load L, and letting $n = 1$ or $m = 1$ minimizes the storage cost for storing matrix A or matrix B, respectively. Our proposed entangled polynomial code achieves the optimum linear recovery threshold in all these cases. More generally, adjusting the value of p trades communication by storage; then adjusting the ratio between m and n allows for minimizing the overall storage cost, to account for the scenario where the sizes of input matrices are unbalanced. Finally, by scaling p, m, and n without taking the computational constraint into account, we enable the flexibility in terms of level of distribution.

2) uniqueness of zero element. This technique can be extended to any other bilinear operations with similar properties, such as convolution, as mentioned later (see Theorem 4). On the other hand, the information theoretic converse is obtained through a cut-set type argument, which allows a lower bound on the recovery thresholds even for non-linear codes.

Our final result on the main problem is characterizing the optimum linear recovery threshold K^*_{linear} within a factor of 2 for *all* possible p, m, n, and N, by developing an improved version of the entangled polynomial code. This characterization involves the fundamental concept of bilinear complexity [15]:

Definition 3: The *bilinear complexity* of multiplying an m-by-p matrix and a p-by-n matrix, denoted by $R(p, m, n)$, is defined as the minimum number of element-wise multiplications required to complete such an operation. Rigorously, $R(p, m, n)$ denotes the minimum integer R, such that we can find tensors $a \in \mathbb{F}^{R\times p\times m}$, $b \in \mathbb{F}^{R\times p\times n}$, and $c \in \mathbb{F}^{R\times m\times n}$, satisfying

$$\sum_i c_{ijk}\left(\sum_{j',k'} A_{j'k'}a_{ij'k'}\right)\left(\sum_{j'',k''} B_{j''k''}b_{ij''k''}\right) = \sum_{\ell} A_{\ell j}B_{\ell k}. \tag{14}$$

for any input matrices $A \in \mathbb{F}^{p\times m}$, $B \in \mathbb{F}^{p\times n}$.

Using this concept, we state our result as follows.

Theorem 3: For a distributed matrix multiplication problem of computing $A^{\intercal}B$ using N workers, with parameters p, m, and n, the optimum linear recovery threshold is characterized by

$$R(p, m, n) \le K^*_{\text{linear}} \le 2R(p, m, n) - 1, \tag{15}$$

where $R(p, m, n)$ denotes the bilinear complexity of multiplying an m-by-p matrix and a p-by-n matrix.

Remark 5: The key proof idea of Theorem 3 is twofold. We first demonstrate a one-to-one correspondence between linear computation strategies and upper bound constructions[7] for bilinear complexity, which enables converting a matrix multiplication problem into computing the element-wise product of two vectors of length $R(p, m, n)$. Then we show that an optimal computation strategy can be developed for this augmented problem, which achieves the stated recovery threshold. Similarly to this result, factor-of-2 characterization can also be obtained for non-linear codes, as discussed in Section VI.

Remark 6: The coding construction we developed for proving Theorem 3 provides an improved version of the entangled polynomial code. Explicitly, given any upper bound construction for $R(p, m, n)$ with rank R, the coding scheme achieves a recovery threshold of $2R - 1$, while tolerating arbitrarily many stragglers. This improved version further and orderwise reduces the needed recovery threshold on top of its basic version. For example, by simply applying the well-know Strassen's construction [20], which provides an upper bound $R(2^k, 2^k, 2^k) \le 7^k$ for any $k \in \mathbb{N}$, the proposed coding scheme achieves a recovery threshold of $2 \cdot 7^k - 1$, which orderwise improves upon $K_{\text{entangled-poly}} = 8^k + 2^k - 1$ achieved by the entangled polynomial code. Further improvements can be achieved by applying constructions with lower ranks, up to $2R(p, m, n) - 1$.

[7]Formally defined in Section VI.

Remark 7: In parallel to this work, the Generalized PolyDot scheme was proposed in [21] to extend the PolyDot construction [18] to asymmetric matrix-vector multiplication. Generalized PolyDot can be applied to achieve the same recovery threshold of the entangled polynomial code for special case of $m = 1$ or $n = 1$. However, entangled polynomial codes achieve (unboundedly) better recovery thresholds for general values of p, m, and n.

The techniques we developed in this paper can also be extended to several other problems, such as coded convolution [16] and fault-tolerant computing [17], [22], leading to tight characterizations. For coded convolution, we present our result in the following theorem.

Theorem 4: For the distributed convolution problem of computing $\boldsymbol{a}*\boldsymbol{b}$ using N workers that can each store $\frac{1}{m}$ fraction of $\boldsymbol{a}$ and $\frac{1}{n}$ fraction of $\boldsymbol{b}$, the optimum recovery threshold that can be achieved using linear codes, denoted by $K^*_{\text{conv-linear}}$, is exactly characterized by the following equation

$$K^*_{\text{conv-linear}} = K_{\text{conv-poly}} \triangleq m + n - 1. \tag{16}$$

Remark 8: Theorem 4 is proved based on our previously developed coded computing scheme for convolution, which is a variation of the polynomial code [11]. As mentioned before, we extend the proof idea of Theorem 2 to prove the matching converse. This theorem proves the optimality of the computation scheme in [11] among all computation strategies where the encoding functions are linear. For detailed problem formulation and proof, see Appendix A.

Our second extension is in the fault-tolerant computing setting, which was first discussed in [17] for matrix multiplication. Unlike the straggler effects we studied in this paper, fault tolerance considers scenarios where arbitrary errors can be injected into the computation, and the master has no information about which subset of workers are returning errors. We show that the techniques we developed for straggler mitigation can also be applied in this setting to improve robustness against computing failures, and the optimality of any encoding function in terms of recovery threshold also preserves when applied in the fault-tolerant computing setting. As an example, we present the following theorem, demonstrating this connection.

Theorem 5: For a distributed matrix multiplication problem of computing $A^{\intercal}B$ using N workers, with parameters p, m, and n, if $m = 1$ or $n = 1$, the entangled polynomial code can detect up to

$$E^*_{\text{detect}} = N - K_{\text{entangled-poly}} \tag{17}$$

errors, and correct up to

$$E^*_{\text{correct}} = \left\lfloor \frac{N - K_{\text{entangled-poly}}}{2} \right\rfloor \tag{18}$$

errors. This can not be improved using any other linear encoding strategies.

Remark 9: The proof idea for Theorem 5 is to connect the straggler mitigation problem and the fault tolerance problem by extending the concept of Hamming distance to coded computing. Specifically, we map the straggler mitigation problem to the problem of correcting erasure errors, and the fault tolerance problem to the problem of correcting arbitrary errors. The solution to these two communication problems are deeply connected by the Hamming distance, and we show that this result extends to coded computing (see Lemma 3 in Appendix B). Since the concept of Hamming distance is not exclusively defined for linear codes, this connection also holds for arbitrary computation strategies. Furthermore, this approach can be easily extended to the hybrid settings where both stragglers and computing errors exist, and similar results can be proved. The detailed formulation and proof can be found in Appendix B.

In Section IV, we prove Theorem 1 by describing the entangled polynomial code. Then in Section V, we prove Theorem 2 by deriving the converses. Finally, we present the coding construction and converse for proving Theorem 3 in Section VI.

IV. Entangled Polynomial Code

In this section, we prove Theorem 1 by formally describing the entangled polynomial code and its decoding procedure. We start with an illustrating example.

A. Illustrating Example

Consider a distributed matrix multiplication task of computing $A^\intercal B$ using $N = 5$ workers that can each store half of the rows (i.e., $p = 2$ and $m = n = 1$). We evenly divide each input matrix along the row side into 2 submatrices:

$$A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}, \qquad B = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}, \tag{19}$$

Given this notation, we essentially want to compute

$$C = A^\intercal B = \left[A_0^\intercal B_0 + A_1^\intercal B_1\right]. \tag{20}$$

A naive computation strategy is to let the 5 workers compute each $A_i^\intercal B_i$ uncodedly with repetition. Specifically we can let 3 workers compute $A_0^\intercal B_0$ and 2 workers compute $A_1^\intercal B_1$. However, this approach can only robustly tolerate 1 straggler, achieving a recovery threshold of 4. Another naive approach is to use random linear codes, i.e., let each worker store a random linear combination of A_0, A_1, and a combination of B_0, B_1. However, the resulting computation result of each worker is a random linear combination of 4 variables $A_0^\intercal B_0$, $A_0^\intercal B_1$, $A_1^\intercal B_0$, and $A_1^\intercal B_1$, which also results in a recovery threshold of 4.

Surprisingly, there is a simple computation strategy for this example that achieves the optimum linear recovery threshold of 3. The main idea is to instead inject structured redundancy tailored to the matrix multiplication operation. We present this proposed strategy as follows:

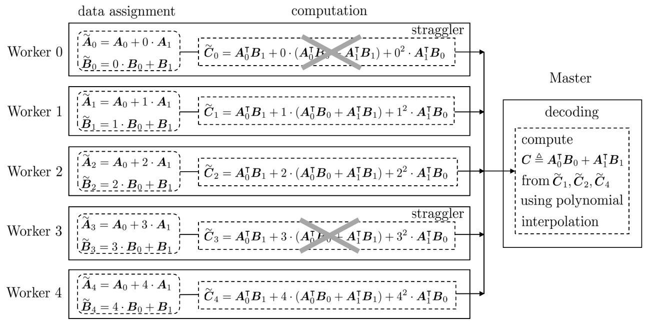
data assignment
computation
straggler
Worker 0
$\tilde{A}_0 = A_0 + 0 \cdot A_1$
$\tilde{B}_0 = 0 \cdot B_0 + B_1$
$\tilde{C}_0 = A_0^\intercal B_1 + 0 \cdot (A_0^\intercal B_0 + A_1^\intercal B_1) + 0^2 \cdot A_1^\intercal B_0$
Worker 1
$\tilde{A}_1 = A_0 + 1 \cdot A_1$
$\tilde{B}_1 = 1 \cdot B_0 + B_1$
$\tilde{C}_1 = A_0^\intercal B_1 + 1 \cdot (A_0^\intercal B_0 + A_1^\intercal B_1) + 1^2 \cdot A_1^\intercal B_0$
Worker 2
$\tilde{A}_2 = A_0 + 2 \cdot A_1$
$\tilde{B}_2 = 2 \cdot B_0 + B_1$
$\tilde{C}_2 = A_0^\intercal B_1 + 2 \cdot (A_0^\intercal B_0 + A_1^\intercal B_1) + 2^2 \cdot A_1^\intercal B_0$
straggler
Worker 3
$\tilde{A}_3 = A_0 + 3 \cdot A_1$
$\tilde{B}_3 = 3 \cdot B_0 + B_1$
$\tilde{C}_3 = A_0^\intercal B_1 + 3 \cdot (A_0^\intercal B_0 + A_1^\intercal B_1) + 3^2 \cdot A_1^\intercal B_0$
Worker 4
$\tilde{A}_4 = A_0 + 4 \cdot A_1$
$\tilde{B}_4 = 4 \cdot B_0 + B_1$
$\tilde{C}_4 = A_0^\intercal B_1 + 4 \cdot (A_0^\intercal B_0 + A_1^\intercal B_1) + 4^2 \cdot A_1^\intercal B_0$
Master
decoding
compute
$C \triangleq A_0^\intercal B_0 + A_1^\intercal B_1$
from $\tilde{C}_1, \tilde{C}_2, \tilde{C}_4$
using polynomial
interpolation

Fig. 3. Example using entangled polynomial code, with 5 workers that can each store half of each input matrix. (a) Computation strategy: each worker i stores $A_0 + iA_1$ and $iB_0 + B_1$, and computes their product. (b) Decoding: master waits for results from *any* 3 workers, and decodes the output using polynomial interpolation.

Suppose elements of A, B are in $\mathbb{R}$. Let each worker $i \in \{0, 1, \ldots, 4\}$ store the following two coded submatrices:

$$\tilde{A}_i = A_0 + iA_1, \qquad \tilde{B}_i = iB_0 + B_1. \tag{21}$$

To prove that this design gives a recovery threshold of 3, we need to find a valid decoding function for any subset of 3 workers. We demonstrate this decodability through a representative scenario, where the master receives the computation results from workers 1, 2, and 4, as shown in Figure 3. The decodability for the other 9 possible scenarios can be proved similarly.

According to the designed computation strategy, we have

$$\begin{bmatrix} \tilde{C}_1 \\ \tilde{C}_2 \\ \tilde{C}_4 \end{bmatrix} = \begin{bmatrix} 1^0 & 1^1 & 1^2 \\ 2^0 & 2^1 & 2^2 \\ 4^0 & 4^1 & 4^2 \end{bmatrix} \begin{bmatrix} A_0^\intercal B_1 \\ A_0^\intercal B_0 + A_1^\intercal B_1 \\ A_1^\intercal B_0 \end{bmatrix}. \tag{22}$$

The coefficient matrix in the above equation is a Vandermonde matrix, which is invertible because its parameters $1, 2, 4$ are distinct in $\mathbb{R}$. So one decoding approach is to directly invert equation (22), of which the returned result includes the needed matrix $C = A_0^\intercal B_0 + A_1^\intercal B_1$. This proves the decodability.

However, as we will explain in the general coding design, directly computing this inverse problem using the classical inversion algorithm might be expensive in some more general cases. Quite interestingly, because of the algebraic structure we designed for the computation strategy (i.e., equation (21)), the decoding process can be viewed as a polynomial interpolation problem (or equivalently, decoding a Reed-Solomon code).

Specifically, in this example each worker i returns

$$\tilde{C}_i = \tilde{A}_i^\intercal \tilde{B}_i = A_0^\intercal B_1 + i(A_0^\intercal B_0 + A_1^\intercal B_1) + i^2 A_1^\intercal B_0, \tag{23}$$

which is essentially the value of the following polynomial at point $x = i$:

$$h(x) \triangleq \tilde{A}_i^\intercal \tilde{B}_i = A_0^\intercal B_1 + x(A_0^\intercal B_0 + A_1^\intercal B_1) + x^2 A_1^\intercal B_0. \tag{24}$$

Hence, recovering C using computation results from 3 workers is equivalent to recovering the linear term coefficient of a quadratic function given its values at 3 points. Later in this section, we will show that by mapping the decoding process to polynomial interpolation, we can achieve almost-linear decoding complexity even for arbitrary parameter values.

B. General Coding Design

Now we present the entangled polynomial code, which achieves a recovery threshold $pmn + p - 1$ for any p, m, n and N as stated in Theorem 1.[8] First of all, we evenly divide each input matrix into pm and pn submatrices according to equations (5) and (6). We then assign each worker $i \in \{0, 1, \ldots, N-1\}$ an element in $\mathbb{F}$, denoted by x_i, and make sure that all x_i's are distinct. Under this setting, we define the following class of computation strategies.

Definition 4: Given parameters $\alpha, \beta, \theta \in \mathbb{N}$, we define the (α, β, θ)-polynomial code as

$$\tilde{A}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1} A_{j,k} x_i^{j\alpha+k\beta},$$
$$\tilde{B}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{n-1} B_{j,k} x_i^{(p-1-j)\alpha+k\theta}, \quad \forall\, i \in \{0, 1, \ldots, N-1\}. \tag{25}$$

In an (α, β, θ)-polynomial code, each worker essentially evaluates a polynomial whose coefficients are fixed linear combinations of the products $A^{\intercal}_{j,k} B_{j',k'}$. Specifically, each worker i returns

$$\tilde{C}_i = \tilde{A}_i^{\intercal} \tilde{B}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1}\sum_{j'=0}^{p-1}\sum_{k'=0}^{n-1} A^{\intercal}_{j,k} B_{j',k'} x_i^{(p-1+j-j')\alpha+k\beta+k'\theta}. \tag{26}$$

Consequently, when the master receives results from enough workers, it can recover all these linear combinations using polynomial interpolation. Recall that we aim to recover

$$C = \begin{bmatrix} C_{0,0} & C_{0,1} & \cdots & C_{0,n-1} \\ C_{1,0} & C_{1,1} & \cdots & C_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m-1,0} & C_{m-1,1} & \cdots & C_{m-1,n-1} \end{bmatrix}, \tag{27}$$

where each submatrix $C_{k,k'} \triangleq \sum_{j=0}^{p-1} A^{\intercal}_{j,k} B_{j,k'}$ is also a fixed linear combination of these products. We design the values of parameters (α, β, θ) such that all these linear combinations appear in (26) separately as coefficients of terms of different degrees. Furthermore, we want to minimize the degree of the polynomial $\tilde{C}_i$, in order to reduce the recovery threshold.

One design satisfying these properties is $(\alpha, \beta, \theta) = (1, p, pm)$, i.e,

$$\tilde{A}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1} A_{j,k} x_i^{j+kp},$$
$$\tilde{B}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{n-1} B_{j,k} x_i^{p-1-j+kpm}. \tag{28}$$

Hence, each worker returns the value of the following degree $pmn + p - 2$ polynomial at point $x = x_i$:

$$h_i(x) \triangleq \tilde{A}_i^{\intercal} \tilde{B}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1}\sum_{j'=0}^{p-1}\sum_{k'=0}^{n-1} A^{\intercal}_{j,k} B_{j',k'} x_i^{(p-1+j-j')+kp+k'pm}, \tag{29}$$

where each $C_{k,k'}$ is exactly the coefficient of the $(p - 1 + kp + k'pm)$-th degree term. Since all x_i's are selected to be distinct, recovering C given results from any $pmn + p - 1$ workers is essentially interpolating $h(x)$ using $pmn + p - 1$ distinct points. Because the degree of $h(x)$ is $pmn + p - 2$, the output C can always be uniquely decoded.

C. Computational Complexities

In terms of complexity, the decoding process of entangled polynomial code can be viewed as interpolating a degree $pmn + p - 2$ polynomial for $\frac{rt}{mn}$ times. It is well known that polynomial interpolation of degree k has a complexity of $O(k \log^2 k \log\log k)$ [23].[9] Therefore, decoding entangled polynomial code only requires at most a complexity of $O(prt \log^2(pmn) \log\log(pmn))$, which is almost linear to the input size of the decoder ($\Theta(prt)$ elements). This complexity can be reduced by simply swapping in any faster polynomial interpolation algorithm or Reed-Solomon decoding algorithm. In addition, this decoding complexity can also be further improved by exploiting the fact that only a subset of the coefficients are needed for recovering the output matrix.

Note that given the presented computation framework, each worker is assigned to multiply two coded matrices with sizes of $\frac{r}{m} \times \frac{s}{p}$ and $\frac{s}{p} \times \frac{t}{n}$, which requires a complexity of $O(\frac{srt}{pmn})$.[10] This complexity is independent of the coding design, indicating that the entangled polynomial code strictly improves other designs without requiring extra computation at the workers. Recall that the decoding complexity of entangled polynomial code grows linearly with respect to the size of the output matrix. The decoding overhead becomes negligible compared to workers' computational load in practical scenarios where the sizes of coded matrices assigned to the workers are sufficiently large. Moreover, the fast decoding algorithms enabled by the Polynomial coding approach further reduces this overhead, compared to general linear coding designs.

Entangled polynomial code also enables improved performances for systems where the data has to encoded online. For instance, if the input matrices are broadcast to the workers and are encoded distributedly, the linearity of entangled polynomial code allows for an in-place algorithm, which does not require addition storage or time complexity. Alternatively, if centralized encoding is required, almost-linear-time

[8]For $N < pmn+p-1$, a recovery threshold of N is achievable by definition. Hence we focus on the case where $N \geq pmn + p - 1$.

[9]When the base field supports FFT, this complexity bound can be improved to $O(k \log^2 k)$.

[10]More precisely, the commonly used cubic algorithm achieves a complexity of $\theta(\frac{srt}{pmn})$ for the general case. Improved algorithms has been found in certain cases (e.g., [20], [24]–[32]), however, all known approaches requires a super-quadratic complexity.

algorithms can also be developed similar to decoding: at most a complexity of $O((\frac{sr}{pm}\log^2(pm)\log\log(pm) + \frac{st}{pn}\log^2(pn)\log\log(pn))N)$ is required using fast polynomial evaluation, which is almost linear with respect to the output size of the encoder ($\Theta((\frac{sr}{pm} + \frac{st}{pn})N)$ elements).

V. Converses

In this section, we provide the proof of Theorem 2. We first prove equation (12) by developing a linear algebraic converse. Then we prove inequality (13) through an information theoretic lower bound.

A. Maching Converses for Linear Codes

To prove equation (12), we start by developing a converse bound on recovery threshold for general parameter values, then we specialize it to the settings where $m = 1$ or $n = 1$. We state this converse bound in the following lemma:

Lemma 1: For a distributed matrix multiplication problem with parameters p, m, n, and N, we have

$$K^*_{\text{linear}} \geq \min\{N,\ \ pm + pn - 1\}. \tag{30}$$

When $m = 1$ or $n = 1$, the RHS of inequality (30) is exactly $K_{\text{entangled-poly}}$. Hence equation (12) directly follows from Lemma 1. So it only suffices to prove Lemma 1, and we prove it as follows:

Proof: To prove Lemma 1, we only need to consider the following two scenarios:

(1) If $K^*_{\text{linear}} = N$, then (30) is trivial.

(2) If $K^*_{\text{linear}} < N$, then we essentially need to show that for any parameter values p, m, n, and N satisfying this condition, we have $K^*_{\text{linear}} \geq pm + pn - 1$. By definition, if such a linear recovery threshold is achievable, we can find a computation strategy, i.e., tensors a, b, and a class of decoding functions $\boldsymbol{d} \triangleq \{d_{\mathcal{K}}\}$, such that

$$d_{\mathcal{K}}\left(\left\{\left(\sum_{j',k'} A^{\intercal}_{j',k'} a_{ij'k'}\right)\left(\sum_{j'',k''} B_{j'',k''} b_{ij''k''}\right)\right\}_{i\in\mathcal{K}}\right) = A^{\intercal}B \tag{31}$$

for any input matrices A and B, and for any subset $\mathcal{K}$ of K^*_{linear} workers.

We choose the values of A and B, such that each $A_{j,k}$ and $B_{j,k}$ satisfies

$$A_{j,k} = \alpha_{jk} A_{\text{c}}, \tag{32}$$

$$B_{j,k} = \beta_{jk} B_{\text{c}}, \tag{33}$$

for some matrices $\alpha \in \mathbb{F}^{p\times m}$, $\beta \in \mathbb{F}^{p\times n}$, and constants $A_{\text{c}} \in \mathbb{F}^{\frac{s}{p}\times\frac{r}{m}}$, $B_{\text{c}} \in \mathbb{F}^{\frac{s}{p}\times\frac{t}{n}}$ satisfying $A_{\text{c}}^{\intercal}B_{\text{c}} \neq 0$. Consequently, we have

$$d_{\mathcal{K}}\left(\left\{\left(\sum_{j',k'} \alpha_{j'k'} a_{ij'k'}\right)\left(\sum_{j'',k''} \beta_{j''k''} b_{ij''k''}\right) A_{\text{c}}^{\intercal}B_{\text{c}}\right\}_{i\in\mathcal{K}}\right) = A^{\intercal}B \tag{34}$$

for all possible values of α, β, and $\mathcal{K}$.

Fixing the value i, we can view each subtensor a_{ijk} as a vector of length pm, and each subtensor b_{ijk} as a vector of length pn. For brevity, we denote each such vector by $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ respectively. Similarly, we can also view matrices α and β as vectors of length pm and pn, and we denote these vectors by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Furthermore, we can define dot products within these vector spaces following the conventions. Using these notations, (34) can be written as

$$d_{\mathcal{K}}\left(\left\{(\boldsymbol{\alpha}\cdot\boldsymbol{a}_i)(\boldsymbol{\beta}\cdot\boldsymbol{b}_i)A_{\text{c}}^{\intercal}B_{\text{c}}\right\}_{i\in\mathcal{K}}\right) = A^{\intercal}B. \tag{35}$$

Given the above definitions, we now prove that within each subset $\mathcal{K}$ of size K^*_{linear}, the vectors $\{\boldsymbol{a}_i\}_{i\in\mathcal{K}}$ span the space $\mathbb{F}^{pm}$. Essentially, we need to prove that for any such given subset $\mathcal{K}$, there does not exist a non-zero $\alpha \in \mathbb{F}^{p\times m}$ such that the corresponding vector $\boldsymbol{\alpha} \in \mathbb{F}^{pm}$ satisfies $\boldsymbol{\alpha}\cdot\boldsymbol{a}_i = 0$ for all $i \in \mathcal{K}$. Assume the opposite that such an α exists, so that $\boldsymbol{\alpha}\cdot\boldsymbol{a}_i$ is always 0, then the LHS of (35) becomes a fixed value. On the other hand, since α is non-zero, we can always find different values of β such that $\alpha^{\intercal}\beta$ is variable. Recalling (32) and (33), the RHS of (35) cannot be fixed if $\alpha^{\intercal}\beta$ is variable, which results in a contradiction.

Now we use this conclusion to prove (30). For any fixed $\mathcal{K}$ with size K^*_{linear}, let $\mathcal{B}$ be a subset of indices in $\mathcal{K}$ such that $\{\boldsymbol{a}_i\}_{i\in\mathcal{B}}$ form a basis. Recall that we are considering the case where $K^*_{\text{linear}} < N$, meaning that we can find a worker $\tilde{k} \notin \mathcal{K}$. For convenience, we define $\mathcal{K}^+ = \mathcal{K}\cup\{\tilde{k}\}$, and $\mathcal{K}^- \triangleq \mathcal{K}^+\backslash\mathcal{B}$. Obviously, $|\mathcal{B}| = pm$, and $|\mathcal{K}^-| = |\mathcal{K}^+| - |\mathcal{B}| = K^*_{\text{linear}} + 1 - pm$. Hence, it suffices to prove that $|\mathcal{K}^-| \geq pn$, which only requires that $\{\boldsymbol{b}_i\}_{i\in\mathcal{K}^-}$ forms a basis of $\mathbb{F}^{pn}$. Equivalently, we only need to prove that any $\beta \in \mathbb{F}^{p\times n}$ such that its vectorized version $\boldsymbol{\beta} \in \mathbb{F}^{pn}$ satisfies $\boldsymbol{\beta}\cdot\boldsymbol{b}_i = 0$ for any $i \in \mathcal{K}^-$ must be zero. For brevity, we let $\mathbb{B}$ denotes the subspace that contains all values of β satisfying this condition.

To prove this statement, we first construct a list of matrices as follows, denoted by $\{\alpha_i\}_{i\in\mathcal{B}}$. Recall that $\{\boldsymbol{a}_i\}_{i\in\mathcal{B}}$ forms a basis. We can find a matrix $\alpha_i \in \mathbb{F}^{p\times m}$ for each $i \in \mathcal{B}$ such that their vectorized version $\{\boldsymbol{\alpha}_i\}_{i\in\mathcal{B}}$ satisfies $\boldsymbol{\alpha}_i\cdot\boldsymbol{a}_{i'} = \delta_{i,i'}$.[11] From elementary linear algebra, the vectors $\{\boldsymbol{\alpha}_i\}_{i\in\mathcal{B}}$ also form a basis of $\mathbb{F}^{pm}$. Correspondingly, their matrix version $\{\alpha_i\}_{i\in\mathcal{B}}$ form a basis of $\mathbb{F}^{p\times m}$.

For any $k \in \mathcal{B}$, we define $\mathcal{K}_k = \mathcal{K}^+\backslash\{k\}$. Note that $|\mathcal{K}_k| = K^*_{\text{linear}}$, equation (35) should also hold for $\mathcal{K}_k$ instead of $\mathcal{K}$. Moreover, note that if we fix $\alpha = \alpha_k$, then the corresponding LHS of (35) remains fixed for any $\beta \in \mathbb{B}$. As a result, $A^{\intercal}B$ must also be fixed. Similar to the above discussion, this requires that the value of $\alpha_k^{\intercal}\beta$ be fixed. This value has to be 0 because $\beta = 0$ satisfies our stated condition.

Now we have proved that any $\beta \in \mathbb{B}$ must also satisfy $\alpha_k^{\intercal}\beta = 0$ for any $k \in \mathcal{B}$. Because $\{\alpha_k\}_{k\in\mathcal{B}}$ form a basis of $\mathbb{F}^{p\times m}$, such β acting on $\mathbb{F}^{p\times m}$ through matrix product has to be the zero operator, so $\beta = 0$. As mentioned above, this results in $K^*_{\text{linear}} \geq pm + pn - 1$, which completes the proof of Lemma 1 and equation (12). ■

[11]Here $\delta_{i,j}$ denotes the discrete delta function, i.e., $\delta_{i,i} = 1$, and $\delta_{i,j} = 0$ for $i \neq j$.

Remark 10: Note that in the above proof, we never used the condition that the decoding functions are linear. Hence, the converse does not require the linearity of the decoder. This fact will be used later in our discussion regarding the fault-tolerant computing in Appendix B.

B. Information Theoretic Converse for Nonlinear Codes

Now we prove inequality (13) through an information theoretic converse bound. Similar to the proof of equation (12), we start by proving a general converse.

Lemma 2: For a distributed matrix multiplication problem with parameters p, m, n, and N, if the base field $\mathbb{F}$ is finite, we have

$$K^* \geq \max\{pm, pn\}. \tag{36}$$

When $m = 1$ or $n = 1$, the RHS of inequality (36) is greater than $\frac{1}{2}K_{\text{entangled-poly}}$. Hence inequality (13) directly results from Lemma 2, which we prove as follows.

Proof: Without loss of generality, we assume $m \geq n$, and aim to prove $K^* \geq pm$. Specifically, we need to show that any computation strategy has a recovery threshold of at least pm, for any possible parameter values. Recall the definition of recovery threshold. It suffices to prove that for any computation strategy $(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{d})$ and any subset $\mathcal{K}$ of workers, if the master can recover C given results from workers in $\mathcal{K}$ (i.e., the decoding function $d_{\mathcal{K}}$ returns C for any possible values of A and B), then we must have $|\mathcal{K}| \geq pm$.

Suppose the condition in the above statement holds. Given each input A, the workers can compute $\{\tilde{A}_i\}_{i\in\mathcal{K}}$ using the encoding functions. On the other hand, for any fixed possible value of B, the workers can compute $\{\tilde{C}_i\}_{i\in\mathcal{K}}$ based on $\{\tilde{A}_i\}_{i\in\mathcal{K}}$. Hence, let $\tilde{C}_{i,\text{func}}$ be a function that returns $\tilde{C}_i$ given B as input, $\{\tilde{C}_{i,\text{func}}\}_{i\in\mathcal{K}}$ is completely determined by $\{\tilde{A}_i\}_{i\in\mathcal{K}}$, without requiring additional information on the value of A. If we view A as a random variable, we have the following Markov chain:

$$A \rightarrow \{\tilde{A}_i\}_{i\in\mathcal{K}} \rightarrow \{\tilde{C}_{i,\text{func}}\}_{i\in\mathcal{K}}. \tag{37}$$

Because the master can decode C as a function of $\{\tilde{C}_i\}_{i\in\mathcal{K}}$, if we define C_{func} similarly as a function that returns C given B as input, C_{func} is also completely determined by $\{\tilde{C}_{i,\text{func}}\}_{i\in\mathcal{K}}$, with no direct dependency on any other variables. Consequently, we have the following extended Markov chain

$$A \rightarrow \{\tilde{A}_i\}_{i\in\mathcal{K}} \rightarrow \{\tilde{C}_i\}_{i\in\mathcal{K}} \rightarrow C_{\text{func}}. \tag{38}$$

Note that by definition, C_{func} has to satisfy $C_{\text{func}}(B) = A^{\mathsf{T}}B$ for any $A \in \mathbb{F}^{s\times r}$ and $B \in \mathbb{F}^{s\times t}$. Hence, C_{func} is essentially a linear operator uniquely determined by A, defined as multiplication by A^{T}. Conversely, one can show that distinct values of A leads to distinct operators, which directly follows from the definition of matrix multiplication. Therefore, the input matrix A can be exactly determined from C_{func}, i.e., $\mathrm{H}(A|C_{\text{func}}) = 0$. Using the data processing inequality, we have $\mathrm{H}(A|\{\tilde{A}_i\}_{i\in\mathcal{K}}) = 0$.

Now let A be uniformly randomly sampled from $\mathbb{F}^{s\times r}$, and we have $\mathrm{H}(A) = sr\log_2|\mathbb{F}|$ bits. On the other hand, each $\tilde{A}_i$ consists of $\frac{sr}{pm}$ elements, which has an entropy of at most $\frac{sr}{pm}\log_2|\mathbb{F}|$ bits. Consequently, we have

$$|\mathcal{K}| \geq \frac{\mathrm{H}(A)}{\max_{i\in\mathcal{K}} \mathrm{H}(\tilde{A}_i)} \geq pm. \tag{39}$$

This concludes the proof of Lemma 2 and inequality (13). ■

VI. Factor of 2 Characterization of Optimum Linear Recovery Threshold

In this section, we provide the proof of Theorem 3. Specifically, we need to provide a computation strategy that achieves a recovery threshold of at most $2R(p, m, n)-1$ for all possible values of p, m, n, and N, as well as a converse result showing that any linear computation strategy requires at least $N \geq R(p, m, n)$ workers for any p, m, and n.

The proof is accomplished in 2 steps. In Step 1, we show that any linear code for matrix multiplication is equivalently an upper bound construction of the bilinear complexity $R(p, m, n)$, and vice versa. This result indicates the equality between $R(p, m, n)$ and the minimum required number of workers, which proves the needed converse. It also converts any matrix multiplication into the computation of element-wise products given two vectors of length $R(p, m, n)$. Then in Step 2, we show that we can find an optimal computation strategy for this augmented computing task. We develop a variation of the entangled polynomial code, which achieves a recovery threshold of $2R(p, m, n) - 1$.

For Step 1, we first formally define upper bound constructions for bilinear complexity.

Definition 5: Given parameters p, m, n, an *upper bound construction for bilinear complexity* $R(p, m, n)$ with *rank* R is a tuple of tensors $a \in \mathbb{F}^{R\times p\times m}$, $b \in \mathbb{F}^{R\times p\times n}$, and $c \in \mathbb{F}^{R\times m\times n}$ such that for any matrices $A \in \mathbb{F}^{p\times m}$, $B \in \mathbb{F}^{p\times n}$,

$$\sum_i c_{ijk}\left(\sum_{j',k'} A_{j'k'}a_{ij'k'}\right)\left(\sum_{j'',k''} B_{j''k''}b_{ij''k''}\right) = \sum_{\ell} A_{\ell j}B_{\ell k}. \tag{40}$$

Recall the definition of linear codes. One can verify that any upper bound construction with rank R is equivalently a linear computing design using R workers when the sizes of input matrices are given by $A \in \mathbb{F}^{p\times m}$, $B \in \mathbb{F}^{p\times n}$. Note that matrix multiplication follows the same rules for any block matrices, this equivalence holds true for any input sizes.[12] Specifically, given an upper bound construction (a, b, c) with rank R, and for general inputs $A \in \mathbb{F}^{s\times r}$, $B \in \mathbb{F}^{s\times t}$, any block of the final output C can be computed as

$$C_{j,k} = \sum_i c_{ijk}\tilde{A}_{i,\text{vec}}^{\mathsf{T}}\tilde{B}_{i,\text{vec}}, \tag{41}$$

where $\tilde{A}_{i,\text{vec}}$ and $\tilde{B}_{i,\text{vec}}$ are linearly encoded matrices stored by R workers, defined as

$$\tilde{A}_{i,\text{vec}} \triangleq \sum_{j,k} A_{j,k}a_{ijk}, \qquad \tilde{B}_{i,\text{vec}} \triangleq \sum_{j,k} B_{j,k}b_{ijk}. \tag{42}$$

[12]Rigorously, it also requires the linear independence of the $A_i^{\mathsf{T}}B_j$'s, which can be easily proved.

Conversely, one can also show that any linear code using N workers is equivalently an upper bound construction with rank N. This equivalence relationship provides a one-to-one mapping between linear codes and upper bound constructions.

Recall the definition of bilinear complexity (provided in Section III), which essentially states that the minimum achievable rank R equals $R(p, m, n)$. We have shown that the minimum number of workers required for any linear code is given by the same quantity, which proves the coverse. In terms of achievability, we have also proved the existence of a linear computing design using $R(p, m, n)$ workers, where the encoding and decoding are characterized by some tensors $a \in \mathbb{F}^{R(p,m,n)\times p\times m}$, $b \in \mathbb{F}^{R(p,m,n)\times p\times n}$, and $c \in \mathbb{F}^{R(p,m,n)\times m\times n}$ satisfying equation (14), following equations (41) and (42). This achievability scheme essentially converts matrix multiplication into a problem of computing the element-wise product of two "vectors" $\tilde{A}_{i,\text{vec}}$ and $\tilde{B}_{i,\text{vec}}$, each of length $R(p, m, n)$. Specifically, the master only needs $\tilde{A}_{i,\text{vec}}^{\intercal}\tilde{B}_{i,\text{vec}}$ for decoding the final output.

Now in Step 2, we develop the optimal computation strategy for this augmented computation task. Given two arbitrary vectors $\tilde{A}_{i,\text{vec}}$ and $\tilde{B}_{i,\text{vec}}$ of length $R(p, m, n)$, we want to achieve a recovery threshold of $2R(p, m, n)-1$ for computing their element-wise product using N workers, each of which can multiply two coded vectors of length 1. As we have explained in Section IV-B, a recovery threshold of N is always achievable, so we only need to focus on the scenario where $N \geq 2R(p, m, n) - 1$.

The main coding idea is to first view the elements in each vector as values of a degree $R(p, m, n) - 1$ polynomial at $R(p, m, n)$ different points. Specifically, given $R(p, m, n)$ distinct elements in the field $\mathbb{F}$, denoted by $x_0, x_1, \ldots, x_{R(p,m,n)-1}$, we find polynomials $\tilde{f}$ and $\tilde{g}$ of degree $R(p, m, n) - 1$, whose coefficients are matrices, such that

$$\tilde{f}(x_i) = \tilde{A}_{i,\text{vec}} \tag{43}$$

$$\tilde{g}(x_i) = \tilde{B}_{i,\text{vec}}. \tag{44}$$

Recall that we want to recover $\tilde{A}_{i,\text{vec}}^{\intercal}\tilde{B}_{i,\text{vec}}$, which is essentially recovering the values of the degree $2R(p, m, n)-2$ polynomial $\tilde{h} \triangleq \tilde{f}^{\intercal}\tilde{g}$ at these $R(p, m, n)$ points. Earlier in this paper, we already developed a coding structure that allows us to recover polynomials of this form. We now reuse the idea in this construction.

Let $y_0, y_1, ..., y_{N-1}$ be distinct elements of $\mathbb{F}$. We let each worker i store

$$\tilde{A}_i = \tilde{f}(y_i), \tag{45}$$

$$\tilde{B}_i = \tilde{g}(y_i), \tag{46}$$

which are linear combinations of the input submatrices. More Specifically,

$$\tilde{A}_i = \sum_j \tilde{A}_{j,\text{vec}} \cdot \prod_{k\neq j} \frac{(y_i - x_k)}{(x_j - x_k)}, \tag{47}$$

$$\tilde{B}_i = \sum_j \tilde{B}_{j,\text{vec}} \cdot \prod_{k\neq j} \frac{(y_i - x_k)}{(x_j - x_k)}. \tag{48}$$

After computing the product, each worker essentially evaluates the polynomial $\tilde{h}$ at y_i. Hence, from the results of any $2R(p, m, n)-1$ workers, the master can recover $\tilde{h}$, which has degree $2R(p, m, n)-2$, and proceed with decoding the output matrix C. This construction achieves a recovery threshold of $2R(p, m, n)-1$, which proves the upper bound in Theorem 3.

Remark 11: The computation strategy we developed in Step 2 provides a tight upper bound on the characterization of the optimum linear recovery threshold for computing element-wise product of two arbitrary vectors using N machines. Its optimality naturally follows from Theorem 2, given that the element-wise product of two vectors contains all the information needed to compute the dot-product, which is a special case of matrix multiplication. We formally state this result in the following corollary.

Corollary 1: Consider the problem of computing the element-wise product of two vectors of length R using N workers, each of which can store a linearly coded element of each vector and return their product to the master. The optimum linear recovery threshold, denoted as $K^*_{\text{e-prod-linear}}$, is given by the following equation:[13]

$$K^*_{\text{e-prod-linear}} = \min\{N, 2R - 1\}. \tag{49}$$

Remark 12: Note that Step 2 of this proof does not require the computation strategy to be linear. Hence, using exactly the same coding approach, we can easily extend this result to non-linear codes, and prove a similar factor-of-2 characterization for the optimum recovery threshold K^*, formally stated in the following corollary.

Corollary 2: For a distributed matrix multiplication problem with parameters p, m, and n, let $N^*(p, m, n)$ denotes the minimum number of workers such that a valid (possibly non-linear) computation strategy exists. Then for all possible values of N, we have

$$N^*(p, m, n) \leq K^* \leq 2N^*(p, m, n) - 1. \tag{50}$$

Remark 13: Finally, note that the computing design provided in this section can be applied any upper bound construction with rank R, achieving a recovery threshold of $2R - 1$, its significance is two-fold. Using constructions that achieves bilinear complexity, it proves the existence of a factor-of-2 optimal computing scheme, which achieves the same recovery threshold while tolerating arbitrarily many stragglers. On the other hand, for cases where $R(p, m, n)$ is not yet known, explicit coding constructions can still be obtained (e.g., using the well know Strassen's result [20], as well as any other known constructions, such as ones presented in [24]–[38]), which enables further improvements upon the basic entangled polynomial code.

A. Computational Complexities

Algorithmically, decoding the improved version of entangled polynomial code can be completed in two steps.

[13]Obviously, we need $N \geq R$ to guarantee the existence of a valid computation strategy.

In step 1, the master can first recover the element-wise products $\{\tilde{A}_{i,\text{vec}}^{\intercal}\tilde{B}_{i,\text{vec}}\}_{i=1}^{R(p,m,n)}$, by Lagruange-interpolating a degree $2R(p,m,n)-1$ polynomial at $R(p,m,n)$ points, for $\frac{rt}{mn}$ times. Similar to the entangled polynomial code, it requires a complexity of at most $O(\frac{rt}{mn}R(p,m,n)\log^2(R(p,m,n))\log\log(R(p,m,n)))$, which is almost linear to the input size of the decoder ($\Theta(\frac{rt}{mn}R(p,m,n))$ elements). Then in Step 2, the master can recover the final results by linearly combining these products, following equation (41). Note that without even exploiting any algebraic properties of the tensor construction, the natural computing approach achieves a complexity of $\Theta(\frac{rt}{mn}R(p,m,n)^2)$ for computing the second step, as well as achieving the same overal decoding complexity. This already achieves a strictly smaller decoding complexity compared with a general linear computing design, which could requires inverting an $R(p,m,n)$-by-$R(p,m,n)$ matrix.[14]

Moreover, note that most commonly used upper bound constructions are based on the sub-multiplicativity of $R(p,m,n)$, further improved decoding algorithms can be designed when these constructions are used instead. As an example, consider Strassen's construction, which achieves a rank of $R = 7^k \geq R(2^k,2^k,2^k)$. The final outputs can essentially be recovered given the intermediate products $\{\tilde{A}_{i,\text{vec}}^{\intercal}\tilde{B}_{i,\text{vec}}\}_{i=1}^{R(p,m,n)}$ by following the last few iterations of Strassen's Algorithm, requiring only a linear complexity $\Theta(\frac{rt}{mn}R(p,m,n))$. This approach achieves an overall decoding complexity of $O(\frac{rt}{mn}R(p,m,n)\log^2(R(p,m,n))\log\log(R(p,m,n)))$, which is almost linear to the input size of the decoder.

Similar to the discussion in Section IV-C, the computational complexity at each worker is $O(\frac{srt}{pmn})$, which is independent of the coding design. Hence, the improved version of the entangled polynomial code also does not require extra computation at the workers, and the decoding overhead becomes negligible when sizes of the coded submatrices are sufficiently large. Improved performances can also be obtained for systems that requires online encoding, following similar approaches used in decoding.

VII. Concluding Remarks

In this paper, we studied the coded distributed matrix multiplication problem and proposed entangled polynomial codes, which allows optimal straggler mitigation and order-wise improves upon the prior arts. Based on our proposed coding idea, we proved a fundamental connection between the optimum linear recovery threshold and the bilinear complexity, which characterizes the optimum linear recovery threshold within a factor of 2 for all possible parameter values. The techniques developed in this paper can be directly applied to many other problems, including coded convolution and fault-tolerant computing, providing matching characterizations. By directly extending entangled polynomial codes to secure [39]–[53], private [45], [47], [52], [54], and batch [50], [55], [56] distributed matrix multiplication, we can also unboundedly improve all other block-partitioning based schemes [43], [44], [52], [55], [56], achieving subcubic recovery threshold while enabling flexible resource tradeoffs. Entangled polynomial codes has also inspired recent development of coded computing schemes for general polynomial computations [57], secure/private computing [58], and secure sharding in blockchain systems [59].

One interesting follow-up direction is to find better characterization of the optimum linear recovery threshold. Although this problem is completely solved for cases including $m=1$, $n=1$, or $p=1$, there is room for improvement in general cases. Another interesting question is whether there exist non-linear coding strategies that strictly out-perform linear codes, especially for the important case where the input matrices are large ($s,r,t \gg p,m,n$), while allowing for efficient decoding algorithms with almost linear complexity. Finally, the main focus of this paper is to provide optimal algorithmic solutions for matrix multiplication on general fields. Although, when the base field is infinite, one can instead embed the computation into finite fields to avoid practical issues such as numerical error and computation overheads (see discussions in [11], [60]). It is an interesting following direction to find new quantization and computation schemes to study optimal tradeoffs between these measures.

Acknowledgment

The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Appendix A
The Optimum Linear Recovery Threshold for Coded Convolution

In this appendix, we first provide the problem formulation for coded convolution, then we prove Theorem 4, which shows the optimality of Polynomial Code for Coded Convolution.

A. System Model and Problem Formulation

Consider a convolution task with two input vectors

$$\boldsymbol{a} = [\boldsymbol{a}_0\ \boldsymbol{a}_1\ \ldots\ \boldsymbol{a}_{m-1}], \qquad \boldsymbol{b} = [\boldsymbol{b}_0\ \boldsymbol{b}_1\ \ldots\ \boldsymbol{b}_{n-1}], \tag{51}$$

where all $\boldsymbol{a}_i$'s and $\boldsymbol{b}_i$'s are vectors of length s over a sufficiently large field $\mathbb{F}$. We want to compute $\boldsymbol{c} \triangleq \boldsymbol{a} * \boldsymbol{b}$ using a master and N workers. Each worker can store two vectors of length s, which are functions of $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively. We refer to these functions as the *encoding functions*, denoted by $(\boldsymbol{f},\boldsymbol{g})$ similar to the matrix multiplication problem.

Each worker computes the convolution of its stored vectors, and returns it to the master. The master only waits for the fastest subset of workers, before proceeding to decode $\boldsymbol{c}$. Similar to the matrix multiplication problem, we define the *recovery threshold* given the *encoding functions*, denoted by $K(\boldsymbol{f},\boldsymbol{g})$, as the minimum number of workers that the master needs to wait that guarantees the existence of valid decoding functions. We aim to characterize the optimum recovery threshold achievable by any linear encoding functions, denoted by $K^*_{\text{conv-linear}}$, and identify an optimal computation strategy that achieves this optimum threshold.

[14]Similar to matrix multiplication, inverting a k-by-k matrix requires a complexity of $O(k^3)$. Faster algorithms has been developed, however, all known results requires super-quadratic complexity.

B. Proof of Theorem 4

Now we prove Theorem 4, which completely solves the above problem. As we have shown in [11], the recovery threshold stated in Theorem 4 is achievable using a variation of polynomial code. This result proves an upperbound of $K^*_{\text{conv-linear}}$. It also identifies an optimal computation strategy. Hence, in this section we focus on proving the matching converse.

Specifically, we aim to prove that given any problem parameters m, n, and N, for any computation strategy, if the encoding functions $(\boldsymbol{f}, \boldsymbol{g})$ are linear, then its recovery threshold is at least $m+n-1$. We prove it by contradiction.

Assume the opposite, then the master can recover $\boldsymbol{c}$ using results from a subset of at most $m+n-2$ workers. We denote this subset by $\mathcal{K}$. Obviously, we can find a partition of $\mathcal{K}$ into two subsets, denoted by $\mathcal{K}_{\text{a}}$ and $\mathcal{K}_{\text{b}}$, such that $|\mathcal{K}_{\text{a}}| \leq m-1$ and $|\mathcal{K}_{\text{b}}| \leq n-1$. Note that the encoding functions of workers in $\mathcal{K}_{\text{a}}$ collaboratively and linearly maps $\mathbb{F}^{ms}$ to $\mathbb{F}^{(m-1)s}$, which has a non-zero kernel. Hence, we can find a non-zero input vector $\boldsymbol{a}$ such that all workers in $\mathcal{K}_{\text{a}}$ returns 0. Similarly, we can find a non-zero $\boldsymbol{b}$ such that all workers in $\mathcal{K}_{\text{b}}$ returns 0. Recall that $\mathcal{K}_{\text{a}} \cup \mathcal{K}_{\text{b}} = \mathcal{K}$. Consequently, when the master receives 0 from all workers in $\mathcal{K}$, the decoding function returns $\boldsymbol{a} * \boldsymbol{b}$.

This convolution product must be the $\mathbf{0}$ vector, given that the workers return the same results under zero inputs. However, note that the convolution operator has no zero-divisor. Either $\boldsymbol{a}$ or $\boldsymbol{b}$ has to be zero, which contradicts the non-zero assumptions. Hence, we have $K(\boldsymbol{f}, \boldsymbol{g}) \geq m+n-1$. This concludes the proof of Theorem 4.

Appendix B
An Equivalence Between Fault Tolerance and Straggler Mitigation

In this appendix, we start by formulating a fault-tolerant computing problem for matrix multiplication, then we prove Theorem 5 by building a connection between straggler mitigation and fault tolerance, by extending the concept of Hamming distance to coded computing.

A. Problem Formulation

We consider a matrix multiplication problem with two input matrices $A \in \mathbb{F}^{s\times r}$ and $B \in \mathbb{F}^{s\times t}$, and we are interested in computing $C \triangleq A^{\intercal}B$ using a master node and N worker nodes, where each worker can store $\frac{1}{pm}$ fraction of A and $\frac{1}{pn}$ fraction of B. Similar to the straggler mitigation problem, each worker i can store two coded matrices $\tilde{A}_i \in \mathbb{F}^{\frac{s}{p}\times\frac{r}{m}}$ and $\tilde{B}_i \in \mathbb{F}^{\frac{s}{p}\times\frac{t}{n}}$, computed based on A and B respectively. Each worker can compute the product $\tilde{C}_i \triangleq \tilde{A}_i^{\intercal}\tilde{B}_i$, and return it to the master. Unlike the straggler setting, the master waits for all workers before proceeding to recover the final output C. However, a subset of workers can return error results, and the master has no information on which subset of results are false. Under this setting, the master wants to: (1) determine if there is an error in the workers' outputs, and (2) try to recover the final output C using the possibly false computing results from the workers.

Given the above system model, we formulate this fault-tolerant computing problem based on the following terminology. Similar to our main problem in this paper, we define the *encoding functions* and denote them by $(\boldsymbol{f}, \boldsymbol{g})$. We also define the *decoding function* for the master, however in this problem it can either return an estimate of C, or report an error. We only consider the *valid* decoding functions, which always correctly decodes C when no worker is making mistakes.

For any integer E, we say the encoding functions can *detect* E errors if we can find a decoding function that either returns the correct value of C or reports an error, when no more than E workers are making mistakes. Moreover, we say the encoding functions can *correct* E errors, if the decoding function always correctly decodes C. We denote the maximum possible integer E given these two criteria by $E_{\text{detect}}(\boldsymbol{f}, \boldsymbol{g})$ and $E_{\text{correct}}(\boldsymbol{f}, \boldsymbol{g})$ respectively.

We aim to find encoding functions that allows detecting and correcting the maximum possible number of errors. Among all possible computation strategies, we are particularly interested in *linear encoding functions*, as defined in Section II. Given the above terminology, we define the following concepts.

Definition 6: For a distributed matrix multiplication problem of computing $A^{\intercal}B$ using N workers, we define the *maximum detectable errors* and the *maximum detectable errors*, denoted by E^*_{detect} and E^*_{correct} respectively, as the maximum possible values of $E_{\text{detect}}(\boldsymbol{f}, \boldsymbol{g})$ and $E_{\text{correct}}(\boldsymbol{f}, \boldsymbol{g})$ over the set of all encoding functions that are linear.

Our goal is to characterize the values of E^*_{detect} and E^*_{correct}, and to find optimal computation strategies to achieve these values. We are also interested in extending these characterizations to non-linear codes.

B. Proof of Theorem 5

We start by defining some concepts, which allows connecting the fault-tolerant computing problem to the straggler mitigation problem.

Definition 7: We define the *Hamming distance* of any encoding functions $(\boldsymbol{f}, \boldsymbol{g})$, denoted by $d(\boldsymbol{f}, \boldsymbol{g})$, as the maximum integer d such that for any two pairs of input matrices whose products C are different, at least d workers compute different values of $\tilde{C}_i$.

Definition 8: We define the *Recovery threshold* of any encoding functions $(\boldsymbol{f}, \boldsymbol{g})$, denoted by $K(\boldsymbol{f}, \boldsymbol{g})$, as the minimum possible recovery threshold given any decoding functions.

We prove that all these three mentioned criteria for designing encoding functions are directly connected by the Hamming distance, which is formally stated as follows.

Lemma 3: For any (possibly non-linear) computation strategy, we have

$$K(\boldsymbol{f}, \boldsymbol{g}) = N - d(\boldsymbol{f}, \boldsymbol{g}) + 1, \tag{52}$$

$$E_{\text{detect}}(\boldsymbol{f}, \boldsymbol{g}) = d(\boldsymbol{f}, \boldsymbol{g}) - 1, \tag{53}$$

$$E_{\text{correct}}(\boldsymbol{f}, \boldsymbol{g}) = \left\lfloor \frac{d(\boldsymbol{f}, \boldsymbol{g}) - 1}{2} \right\rfloor. \tag{54}$$

Remark 14: Lemma 3 essentially indicates that optimizing the straggler mitigation performance over any class of encoding designs is equivalently optimizing its performance in the fault tolerance setting. Furthermore, all these previously mentioned metrics can be simultaneously optimized by the codes with the maximum possible Hamming distance. Hence, there is no tension among these metrics. This result bridges the rich literature of coding theory and distributed computing.

Remark 15: In terms of achievability, Lemma 3 also provides a large class of coding designs for fault-tolerant computing. Specifically, it indicates that given any computing scheme (e.g., the entangled polynomial code, or its improved version) that achieves a certain recovery threshold, denoted by K. Using the same encoding functions, we can obtain a fault-tolerant scheme that detects up to $N-K$ errors, or correct up to $\lfloor \frac{N-K}{2} \rfloor$ errors.

Proof of Lemma 3: Lemma 3 is a direct consequence of the classical coding theory, given that mitigating straggler effects is essentially correcting erasure errors, and tolerating false results in computing is essentially correcting arbitrary error. Hence, we only provide the proof of (52), where equations (53) and (54) can be proved using similar approaches.

Specifically, we want to prove that for any integer K, a recovery threshold of K is achievable by some encoding functions if and only if their Hamming distance is greater or equal to $N-K+1$. If K is achievable, it means that we can find decoding functions that uniquely determines the value of C given results from any K workers. Equivalently, for distinct values of C, at least $N-K+1$ workers has to return distinct results. Recall that the recovery threshold is the minimum of such integer K, and the Hamming distance is the maximum integer that corresponds to $N-K+1$. We have $K(\boldsymbol{f}, \boldsymbol{g}) = N - d(\boldsymbol{f}, \boldsymbol{g}) + 1$. ■

Now we continue to prove Theorem 5 using Lemma 3. As mentioned in Remark 10, the proof of Theorem 2 essentially completely characterizes the optimum recovery threshold over all linear encoding functions for $m=1$ or $n=1$, which is given by $K_{\text{entangled-poly}}$. Hence, using Lemma 3, we directly obtain that if $m=1$ or $n=1$, we have

$$E^*_{\text{detect}} = N - K_{\text{entangled-poly}}, \tag{55}$$

$$E^*_{\text{correct}} = \left\lfloor \frac{N - K_{\text{entangled-poly}}}{2} \right\rfloor. \tag{56}$$

This concludes the proof of Theorem 5.

References

[1] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 2022–2026.

[2] L. E. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Dept. Elect. Eng., Montana State Univ., Bozeman, MT, USA, 1969.

[3] J. Choi, D. W. Walker, and J. J. Dongarra, "PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers," *Concurrency: Pract. Exper.*, vol. 6, no. 7, pp. 543–570, Oct. 1994.

[4] R. A. Van De Geijn and J. Watts, "SUMMA: Scalable universal matrix multiplication algorithm," *Concurrency: Pract. Exper.*, vol. 9, no. 4, pp. 255–274, Apr. 1997.

[5] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms," in *Proc. 17th Int. Eur. Conf. Parallel Process.* Berlin, Germany: Springer-Verlag, 2011, pp. 90–109.

[6] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[7] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. OSDI*, vol. 8, Dec. 2008, p. 7.

[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," 2015, *arXiv:1512.02673*. [Online]. Available: https://arxiv.org/abs/1512.02673

[9] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2092–2100.

[10] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," 2016, *arXiv:1612.03301*. [Online]. Available: https://arxiv.org/abs/1612.03301

[11] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Adv. Neural Inf. Process. Syst.*, 2017, pp. 4406–4416.

[12] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," 2016, *arXiv:1609.01690*. [Online]. Available: https://arxiv.org/abs/1609.01690

[13] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 34–40, Apr. 2017.

[14] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.

[15] M. Bläser, "Fast matrix multiplication," *Graduate Surveys, Theory of Computing Library*, no. 5, 2013. [Online]. Available: https://theoryofcomputing.org/articles/gs005/

[16] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2403–2407.

[17] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vols. C-33, no. 6, pp. 518–528, Jun. 1984.

[18] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput.*, Oct. 2017, pp. 1264–1270.

[19] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2418–2422.

[20] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, no. 4, pp. 354–356, Aug. 1969.

[21] 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 *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1585–1589.

[22] J.-Y. Jou and J. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proc. IEEE*, vol. 74, no. 5, pp. 732–741, May 1986.

[23] J. Von Zur Gathen and J. Gerhard, *Modern Computer Algebra*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[24] V. Y. Pan, "Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations," in *Proc. 19th Annu. Symp. Found. Comput. Sci.*, Oct. 1978, pp. 166–176.

[25] D. Bini, "Relations between exact and approximate bilinear algorithms. Applications," *CALCOLO*, vol. 17, pp. 87–97, Jan. 1980.

[26] A. Schönhage, "Partial and total matrix multiplication," *SIAM J. Comput.*, vol. 10, no. 3, pp. 434–455, Aug. 1981.

[27] F. Romani, "Some properties of disjoint sums of tensors related to matrix multiplication," *SIAM J. Comput.*, vol. 11, no. 2, pp. 263–267, May 1982.

[28] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," in *Proc. 22nd Annu. Symp. Found. Comput. Sci.*, Washington, DC, USA, Oct. 1981, pp. 82–90.

[29] V. Strassen, "The asymptotic spectrum of tensors and the exponent of matrix multiplication," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, Washington, DC, USA, 1986, pp. 49–54.
[30] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symbolic Comput.*, vol. 9, no. 3, pp. 251–280, Mar. 1990.
[31] A. J. Stothers, "On the complexity of matrix multiplication," Ph.D. dissertation, School Math., Univ. Edinburgh, Edinburgh, Scotland, 2010.
[32] V. V. Williams, "Multiplying matrices faster than Coppersmith-Winograd," in *Proc. 44th Symp. Theory Comput. (STOC)*, 2012, pp. 887–898.
[33] J. E. Hopcroft and L. R. Kerr, "On minimizing the number of multiplications necessary for matrix multiplication," *SIAM J. Appl. Math.*, vol. 20, no. 1, pp. 30–36, Jan. 1971.
[34] J. D. Laderman, "A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications," *Bull. Amer. Math. Soc.*, vol. 82, no. 1, pp. 126–128, 1976.
[35] C.-É. Drevet, M. N. Islam, and É. Schost, "Optimization techniques for small matrix multiplication," *Theor. Comput. Sci.*, vol. 412, no. 22, pp. 2219–2236, May 2011.
[36] A. V. Smirnov, "The bilinear complexity and practical algorithms for matrix multiplication," *Comput. Math. Math. Phys.*, vol. 53, no. 12, pp. 1781–1795, Dec. 2013.
[37] A. Sedoglavic, "A non-commutative algorithm for multiplying 5×5 matrices using 99 multiplications," 2017, *arXiv:1707.06860*. [Online]. Available: https://arxiv.org/abs/1707.06860
[38] A. Sedoglavic, "A non-commutative algorithm for multiplying (7×7) matrices using 250 multiplications," Aug. 2017, *arXiv:1712.07935*. [Online]. Available: https://arxiv.org/abs/1712.07935
[39] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," 2018, *arXiv:1806.00469*. [Online]. Available: https://arxiv.org/abs/1806.00469
[40] H. Yang and J. Lee, "Secure distributed computing with straggling servers using polynomial codes," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 1, pp. 141–150, Jan. 2019.
[41] J. Kakar, S. Ebadifar, and A. Sezgin, "Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation," 2018, *arXiv:1810.13006*. [Online]. Available: https://arxiv.org/abs/1810.13006
[42] R. G. D'oliveira, S. El Rouayheb, and D. Karpuk, "GASP codes for secure distributed matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 1107–1111.
[43] H. A. Nodehi, S. R. H. Najarkolaei, and M. A. Maddah-Ali, "Entangled polynomial coding in limited-sharing multi-party computation," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.
[44] M. Aliasgari, O. Simeone, and J. Kliewer, "Distributed and private coded matrix computation with flexible communication load," 2019, *arXiv:1901.07705*. [Online]. Available: https://arxiv.org/abs/1901.07705
[45] M. Kim and J. Lee, "Private secure coded computation," 2019, *arXiv:1902.00167*. [Online]. Available: https://arxiv.org/abs/1902.00167
[46] J. Kakar, S. Ebadifar, and A. Sezgin, "On the capacity and straggler-robustness of distributed secure matrix multiplication," *IEEE Access*, vol. 7, pp. 45783–45799, 2019.
[47] W.-T. Chang and R. Tandon, "On the upload versus download cost for secure and private matrix multiplication," 2019, *arXiv:1906.10684*. [Online]. Available: https://arxiv.org/abs/1906.10684
[48] S. Ebadifar, J. Kakar, and A. Sezgin, "The need for alignment in rate-efficient distributed two-sided secure matrix computation," in *Proc. IEEE Int. Conf. Commun.(ICC)*, May 2019, pp. 1–6.
[49] H. A. Nodehi and M. A. Maddah-Ali, "Secure coded multi-party computation for massive matrix operations," 2019, *arXiv:1908.04255*. [Online]. Available: https://arxiv.org/abs/1908.04255
[50] Z. Jia and S. A. Jafar, "On the capacity of secure distributed matrix multiplication," 2019, *arXiv:1908.06957*. [Online]. Available: https://arxiv.org/abs/1908.06957
[51] J. Kakar, A. Khristoforov, S. Ebadifar, and A. Sezgin, "Uplink-downlink tradeoff in secure distributed matrix multiplication," 2019, *arXiv:1910.13849*. [Online]. Available: https://arxiv.org/abs/1910.13849
[52] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," 2019, *arXiv:1909.00407*. [Online]. Available: https://arxiv.org/abs/1909.00407
[53] R. G. D'Oliveira, S. El Rouayheb, D. Heinlein, and D. Karpuk, "Degree tables for secure distributed matrix multiplication," in *Proc. ITW*, 2019. [Online]. Available: http://eceweb1.rutgers.edu/~csi/docs/Degree_Tables_for_Secure_Distributed_Matrix_Multiplication_ITW2019.pdf
[54] M. Kim, H. Yang, and J. Lee, "Private coded matrix multiplication," *IEEE Trans. Inf. Forensics Security*, to be published.
[55] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch matrix multiplication," 2019, *arXiv:1909.13873*. [Online]. Available: https://arxiv.org/abs/1909.13873
[56] Z. Jia and S. A. Jafar, "Generalized cross subspace alignment codes for coded distributed batch matrix multiplication," Center Pervasive Commun. Comput., Univ. California Irvine, Irvine, CA, USA, Tech. Rep., 2019. [Online]. Available: https://escholarship.org/uc/item/8fx6x83p
[57] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. Mach. Learn. Res.*, vol. 89, K. Chaudhuri and M. Sugiyama, eds., Apr. 2019, pp. 1215–1225.
[58] J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, "Coded-privateml: A fast and privacy-preserving framework for distributed machine learning," 2019, *arXiv:1902.00641*. [Online]. Available: https://arxiv.org/abs/1902.00641
[59] S. Li, M. Yu, S. Avestimehr, S. Kannan, and P. Viswanath, "Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously," 2018, *arXiv:1809.10361*. [Online]. Available: https://arxiv.org/abs/1809.10361
[60] Q. Yu and A. S. Avestimehr, "Harmonic coding: An optimal linear code for privacy-preserving gradient-type computation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 1102–1106.

Qian Yu (Student Member, IEEE) received the B.S. degree in EECS and physics and the M.Eng. degree in electrical engineering from the Massachusetts Institute of Technology (MIT). He is currently pursuing the Ph.D. degree in electrical and computer engineering with the Viterbi School of Engineering, University of Southern California (USC).

His research interests span information theory, distributed computing, and many other math-related problems. He was a recipient of the Google Ph.D. Fellowship in 2018. He received the Jack Keil Wolf ISIT Student Paper Award in 2017. He also received the Annenberg Graduate Fellowship in 2015, and Honorable Mention in the William Lowell Putnam Mathematical Competition in 2013.

Mohammad Ali Maddah-Ali (Member, IEEE) received the B.Sc. degree in electrical engineering from the Isfahan University of Technology and the M.A.Sc. degree in electrical engineering from the University of Tehran. He is currently pursuing the Ph.D. degree with the Coding and Signal Transmission Laboratory (CST Lab), Department of Electrical and Computer Engineering, University of Waterloo.

From 2002 to 2007, he was with the Coding and Signal Transmission Laboratory (CST Lab), Department of Electrical and Computer Engineering, University of Waterloo, Canada. From 2007 to 2008, he was with the Wireless Technology Laboratories, Nortel Networks, Ottawa, ON, Canada. From 2008 to 2010, he was a Post-Doctoral Fellow with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. Then, he joined Bell Labs, Holmdel, NJ, USA, as a Communication Research Scientist. Recently, he started working at the Sharif University of Technology as a Faculty Member. He was a recipient of NSERC Postdoctoral Fellowship in 2007, the Best Paper Award from the IEEE International Conference on Communications (ICC) in 2014, the IEEE Communications Society and IEEE Information Theory Society Joint Paper Award in 2015, and the IEEE Information Theory Society Joint Paper Award in 2016.

A. Salman Avestimehr (Fellow, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology in 2003, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 2005 and 2008, respectively.

He is currently a Professor with the Electrical and Computer Engineering Department, University of Southern California. His research interests include information theory, coding theory, large-scale distributed computing, machine learning, and secure/private computing. He has received a number of awards, including the IEEE Information Theory Society James L. Massey Research and Teaching Award, the Communications Society and Information Theory Society Joint Paper Award, the Presidential Early Career Award for Scientists and Engineers (PECASE) for pushing the frontiers of information theory through its extension to complex wireless information networks, the Young Investigator Program (YIP) Award from the U.S. Air Force Office of Scientific Research, the National Science Foundation CAREER Award, and the David J. Sakrison Memorial Prize. He is currently a general Co-Chair of the 2020 IEEE International Symposium on Information Theory.