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

Qian Yu*, Mohammad Ali Maddah-Ali†, and A. Salman Avestimehr*
* EE Department, University of Southern California † Nokia Bell Labs

***Abstract*—Consider massive matrix multiplication, a problem that underlies many data analytic applications, in a large-scale distributed system comprising a group of workers. 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*, designing intermediate computations at the workers 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 prove the optimality of entangled polynomial code in several cases, and show that it provides order-wise 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.**

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 can not be provided by a single machine. Hence, deploying matrix computation tasks on large-scale distributed systems has received wide interests [1]–[4].

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 [5]. The conventional approach for mitigating stragglers involves injecting some form of "computation redundancy" such as repetition (e.g., [6]). 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 [7]–[11].

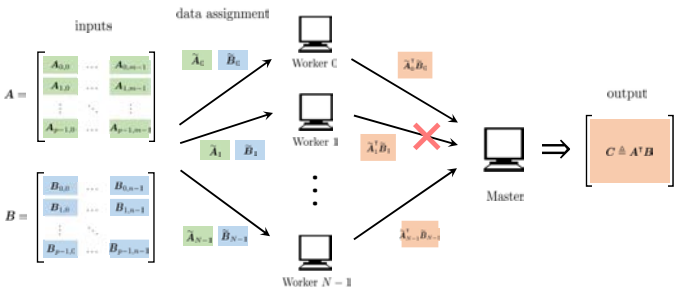

Fig. 1: Overview of the distributed matrix multiplication problem. Each worker computes the product of the two stored encoded submatrices ($\tilde{A}_i$'s and $\tilde{B}_i$'s) and returns the result to the master. By carefully designing the coding strategy, the master can decode the multiplication result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

In this paper, we consider a general formulation of distributed matrix multiplication, and study information-theoretic limits and optimal coding designs for straggler 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 Fig. 1, the two input matrices are partitioned (arbitrarily) into p-by-m and p-by-n blocks of submatrices respectively, where all submatrices within each input are of equal size. Each worker has 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 multiply their two stored (coded) submatrices 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, providing robustness against stragglers.

Note that by selecting different values of parameters p, m, and n, we allow flexible partitioning of input matrices, which in return enables different utilization of system resources (i.e., the required amount of storage at each worker and the amount of communication from worker to master). Hence, considering the system constraints on available storage and communication resources, one can choose desired values for p, m, and n. Thus, 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 [9], 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,

978-1-5386-4780-6/18/$31.00©2018 IEEE

Authorized licensed use limited to: University of Southern California. Downloaded on October 18,2024 at 22:41:13 UTC from IEEE Xplore. Restrictions apply.

which is an order of p less. The particular structure of the proposal 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. This allows orderwise improvement upon conventional uncoded approaches and MDS-coding type approaches for straggler mitigation [7], [8].

Entangled polynomial code generalizes the previously proposed polynomial code for distributed matrix multiplication [9], 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 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 the summation of the product of p submatrices of A and B. Entangled polynomial code effectively aligns the computation of the workers with 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* input matrix partitionings (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 is the *bilinear complexity* of multiplying an m-by-p matrix to a p-by-n matrix (see Definition 3). This is achieved by developing an improved version of the entangled polynomial code. While evaluating bilinear complexity is a well-known challenging problem [12], we show that the optimal recovery threshold for linear coding strategies can be approximated within a factor of 2 of this fundamental quantity.

We note that recently, another computation design named PolyDot was also proposed for this problem, achieving a recovery threshold of $m^2(2p - 1)$ for $m = n$ [13], [14]. Both entangled polynomial code and PolyDot are developed by extending the polynomial codes proposed in [9] to allow arbitrary partitioning of input matrices. Compared with PolyDot, entangled polynomial code achieves a smaller recovery threshold 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. Furthermore, 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 $\mathbb{F}$.[1] We are interested in 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

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

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

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

that are used by the master to recover C given results from any subset $\mathcal{K}$ of the workers. Specifically, each worker i stores matrices $\tilde{A}_i = f_i(A)$ and $\tilde{B}_i = g_i(B)$, and the master can compute an estimate $\hat{C}$ of matrix C using results from a subset $\mathcal{K}$ of the workers by computing $\hat{C} = d_{\mathcal{K}}(\{\tilde{C}_i\}_{i\in\mathcal{K}})$.

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 size

$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,m-1} \\ \vdots & \ddots & \vdots \\ A_{p-1,0} & \cdots & A_{p-1,m-1} \end{bmatrix}, B = \begin{bmatrix} B_{0,0} & \cdots & B_{0,n-1} \\ \vdots & \ddots & \vdots \\ B_{p-1,0} & \cdots & B_{p-1,n-1} \end{bmatrix}, \quad (3)$$

such that the encoding functions of each worker i can be written as $\tilde{A}_i = \sum_{j,k} A_{j,k} a_{ijk}$ and $\tilde{B}_i = \sum_{j,k} B_{j,k} b_{ijk}$ for some tensors a and b, and the decoding function given each subset $\mathcal{K}$ can be written as $\hat{C}_{j,k} = \sum_{i\in\mathcal{K}} \tilde{C}_i c_{ijk}$ for some tensor c.[2] For brevity, we denote the set of linear codes as $\mathcal{L}$.

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

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

Authorized licensed use limited to: University of Southern California. Downloaded on October 18,2024 at 22:41:13 UTC from IEEE Xplore. Restrictions apply.

The major advantage of linear codes is that they guarantee both the encoding and the decoding complexities scale linearly with respect to the size of the input matrices. Furthermore, as we have proved in [9], 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,

$$K^*_{\text{linear}} \triangleq \min_{(\boldsymbol{f},\boldsymbol{g},\boldsymbol{d})\in\mathcal{L}} K(\boldsymbol{f},\boldsymbol{g},\boldsymbol{d}). \tag{4}$$

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. Equivalently, N has to be at least the bilinear complexity of multiplying an m-by-p matrix and a p-by-n matrix.

We are also interested in characterizing the minimum recovery threshold achievable using general coding strategies (including non-linear codes). Similar to [9], 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 3 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.[3]

$$K_{\text{entangled-poly}} \triangleq pmn + p - 1. \tag{5}$$

Furthermore, the entangled polynomial code can be decoded at the master node with at most the complexity of polynomial interpolation given $pmn + p - 1$ points.

Remark 1. Compared to possible alternatives, our proposed entangled polynomial code provides orderwise 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. The scheme tolerates stragglers by adding redundancy through repetition, but its recovery threshold grows linearly with N. Another approach is to let each worker store two random linear combinations of input submatrices (referred to as the *random linear code*). With high probability, this achieves recovery threshold $K_{\text{RL}} \triangleq p^2mn$, which does not scale with N. However, to calculate C, we need at most pmn submatrix multiplication results. Indeed, the lack of structure in the random coding forces the system to wait for p times more than what is essentially needed. Perhaps one surprising aspect of the proposed entangled polynomial code is that, due to its particular structure which aligns the computation of the workers with 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.

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

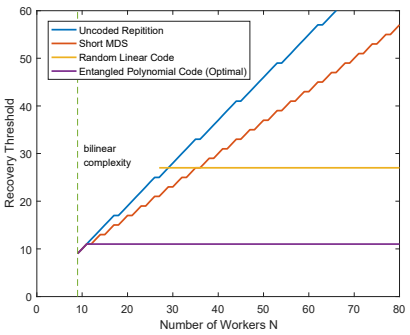

Fig. 2: Comparison of the recovery thresholds achieved by the uncoded repetition scheme, the random linear code, the short-MDS (or short-dot) [7], [8] and our proposed entangled polynomial code, given problem parameters $p = m = 3$, $n = 1$. The entangled polynomial code orderwise improves upon all other approaches.

Remark 2. There have been several works in prior literature investigating the $p = 1$ case [7], [9], [15]. For this special case, entangled polynomial code reduces to our previously proposed polynomial code, achieving the optimum recovery threshold mn and orderwise improvement upon other designs. On the other hand, there has been some investigation on matrix-by-vector type multiplication [7], [8], 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).

Our second result is the optimality of the entangled polynomial code for $m = 1$ or $n = 1$.

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*

$$K^*_{\text{linear}} = K_{\text{entangled-poly}}. \tag{6}$$

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

$$\frac{1}{2}K_{\text{entangled-poly}} < K^* \leq K_{\text{entangled-poly}}. \tag{7}$$

Remark 3. We omit the proof of Theorem 2 for brevity, which can be found in the long version [16]. The key proof idea is to first exploit the algebraic structure of matrix multiplication and develop a linear algebraic converse, which proves equation (6). Then we construct an information theoretic converse which also holds for non-linear codes, to prove inequality (7).

Our final result 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 [12]:

Authorized licensed use limited to: University of Southern California. Downloaded on October 18,2024 at 22:41:13 UTC from IEEE Xplore. Restrictions apply.

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 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

$$\sum_i c_{ijk}\Big(\sum_{j',k'} A_{j'k'}a_{ij'k'}\Big)\Big(\sum_{j'',k''} B_{j''k''}b_{ij''k''}\Big) = \sum_i A_{ij}B_{ik}$$

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*

$$R(p,m,n) \le K^*_{\text{linear}} \le 2R(p,m,n) - 1, \tag{8}$$

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

In this paper, we provide proof sketches for Theorems 1 and 3. Other proofs and extensions (e.g., coded convolution, fault tolerance computing) can be found in the long version [16].

IV. Entangled Polynomial Code

We prove Theorem 1 by describing the entangled polynomial code and its decoding procedure, starting with an 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:

$$A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}, \qquad B = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}, \tag{9}$$

Given this notation, we essentially want to compute

$$C = A^\intercal B = \begin{bmatrix} A_0^\intercal B_0 + A_1^\intercal B_1 \end{bmatrix}. \tag{10}$$

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, a simple computation strategy achieves the optimum linear recovery threshold of 3, by injecting structured redundancy tailored to matrix multiplication operations:

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

$$\tilde{A}_i = A_0 + iA_1, \qquad \tilde{B}_i = iB_0 + B_1. \tag{11}$$

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 an example 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.

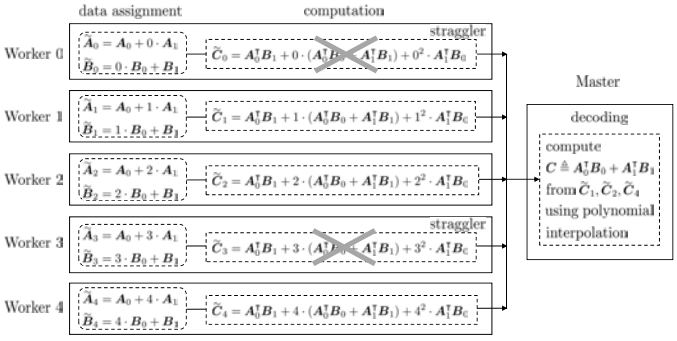

Fig. 3: Example using entangled polynomial code, with 5 workers that can each store half of each input matrix. (a) Computation strategy: each worker i stores $A_0 + iA_1$ and $iB_0 + B_1$, and computes their product. (b) Decoding: master waits for results from *any* 3 workers, and decodes the output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

According to the designed computation strategy, we have

$$\begin{bmatrix} \tilde{C}_1 \\ \tilde{C}_2 \\ \tilde{C}_4 \end{bmatrix} = \begin{bmatrix} 1^0 & 1^1 & 1^2 \\ 2^0 & 2^1 & 2^2 \\ 4^0 & 4^1 & 4^2 \end{bmatrix} \begin{bmatrix} A_0^\intercal B_1 \\ A_0^\intercal B_0 + A_1^\intercal B_1 \\ A_1^\intercal B_0 \end{bmatrix}. \tag{12}$$

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 (12), of which the returned result includes the needed matrix $C = A_0^\intercal B_0 + A_1^\intercal B_1$. This proves the decodability.

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.[4] First of all, we evenly divide each input matrix into pm and pn submatrices according to equation (3). We then assign each worker $i \in \{0,1,...,N-1\}$ a distinct element in $\mathbb{F}$, denoted by x_i. 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

$$\tilde{A}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{m-1} A_{j,k}x_i^{j\alpha+k\beta}, \tag{13}$$

$$\tilde{B}_i = \sum_{j=0}^{p-1}\sum_{k=0}^{n-1} B_{j,k}x_i^{(p-1-j)\alpha+k\theta}. \tag{14}$$

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

$$\tilde{C}_i = \tilde{A}_i^\intercal \tilde{B}_i = \sum_{j,k,j',k'} A_{j,k}^\intercal B_{j',k'} x_i^{(p-1+j-j')\alpha+k\beta+k'\theta}. \tag{15}$$

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

Authorized licensed use limited to: University of Southern California. Downloaded on October 18,2024 at 22:41:13 UTC from IEEE Xplore. Restrictions apply.

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 $C_{k,k'} \triangleq \sum_{j=0}^{p-1} A_{j,k}^{\intercal} B_{j,k'}$, which are also fixed linear combinations of these products. We design parameters (α, β, θ) such that all these linear combinations appear in (15) as coefficients of terms of different degrees. Furthermore, we want to minimize the degree of polynomial $\tilde{C}_i$, in order to reduce the recovery threshold.

One design satisfying these properties is $(\alpha, \beta, \theta) = (1, p, pm)$. Hence, each worker returns the value of the following degree $pmn + p - 2$ polynomial at point $x = x_i$:

$$h_i(x) \triangleq \tilde{C}_i = \sum_{j,k,j',k'} A_{j,k}^{\intercal} B_{j',k'} x_i^{(p-1+j-j')+kp+k'pm}, \quad (16)$$

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.

As of complexity, the decoding can be viewed as interpolating a degree $pmn+p-2$ polynomial for $\frac{rt}{mn}$ times, requiring at most a complexity of $O(prt \log^2(pmn) \log\log(pmn))$, which is almost linear to the input size of the decoder. This complexity can be further improved, as discussed in the long version [16].

V. Factor of 2 characterization of K^*_{linear}

We now provide a proof sketch for the upper bound in Theorem 3. The achievability can be proved in 2 steps.

First, we show that any matrix multiplication is essentially computing the element-wise product of two vectors of length $R(p, m, n)$. Recall the definition of bilinear complexity in Section III; we can always find 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 $c \in \mathbb{F}^{R(p,m,n)\times m\times n}$ such that any block of the final output C can be computed as

$$C_{j,k} = \sum_i c_{ijk} \tilde{A}_{i,\text{vec}}^{\intercal} \tilde{B}_{i,\text{vec}}, \quad (17)$$

where $\tilde{A}_{i,\text{vec}}$ and $\tilde{B}_{i,\text{vec}}$ are linear combinations of the blocks of A and B defined as

$$\tilde{A}_{i,\text{vec}} \triangleq \sum_{j,k} A_{j,k} a_{ijk}, \qquad \tilde{B}_{i,\text{vec}} \triangleq \sum_{j,k} B_{j,k} b_{ijk}. \quad (18)$$

This essentially converts matrix multiplication to 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.

Secondly, we solve this augmented problem by developing an improved version of the entangled polynomial code. This computation strategy encodes any given pair of vectors of length $R(p, m, n)$ for N workers, such that from the computing results of any subset of $2R(p, m, n) - 1$ workers, the master can recover all $R(p, m, n)$ element-wise products.

The main 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 $R(p, m, n) - 1$, such that $\tilde{f}(x_i) = \tilde{A}_{i,\text{vec}}$, and $\tilde{g}(x_i) = \tilde{B}_{i,\text{vec}}$. We want to recover the values of polynomial $\tilde{h} \triangleq \tilde{f}^{\intercal}\tilde{g}$ at these $R(p, m, n)$ points.

Recall that in the entangled polynomial code, we have developed a coding structure that allows us to recover polynomials of this form while achieving a recovery threshold that equals the degree plus one. We now reuse the idea in this construction.

Let y_0, y_1, ..., y_{N-1} be distinct elements of $\mathbb{F}$. We make each worker i store $\tilde{A}_i = \tilde{f}(y_i)$ and $\tilde{B}_i = \tilde{g}(y_i)$. By computing the product, each worker essentially evaluates $\tilde{h}(y_i)$. Hence, from the results of any $2R(p, m, n) - 1$ workers, the master can recover $\tilde{h}$ of degree $2R(p, m, n) - 2$ and decode the output matrix C. This construction achieves a recovery threshold of $2R(p, m, n) - 1$, proving the upper bound in Theorem 3.

VI. Acknowledgement

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053. 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. This work is also in part supported by ONR award N000141612189 and NSF Grants CCF-1703575 and NeTS-1419632.

References

[1] L. E. Cannon, *A Cellular Computer to Implement the Kalman Filter Algorithm*. PhD thesis, Bozeman, MT, USA, 1969.

[2] J. Choi, D. W. Walker, and J. J. Dongarra, "Pumma: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers," *Concurrency: Practice and Experience*, 1994.

[3] R. A. Van De Geijn and J. Watts, "Summa: scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, 1997.

[4] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms," in *ICPP*, 2011.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, 2013.

[6] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," *OSDI*, 2008.

[7] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *e-print arXiv:1512.02673*, 2015.

[8] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *NIPS*, 2016.

[9] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *NIPS*, 2017.

[10] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," *arXiv preprint arXiv:1609.01690*, 2016.

[11] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Commun. Mag.*, vol. 55, pp. 34–40, April 2017.

[12] M. Bläser, *Fast Matrix Multiplication*. No. 5 in Graduate Surveys, Theory of Computing Library, 2013.

[13] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Allerton*, 2017.

[14] S. Dutta, Z. Bai, P. Grover, and T. M. Low, "Coded training of model parallel deep neural networks under soft-errors," *Submitted to ISIT 2018*.

[15] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *IEEE ISIT*, 2017.

[16] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv preprint arXiv:1801.07487*, 2018.

Authorized licensed use limited to: University of Southern California. Downloaded on October 18,2024 at 22:41:13 UTC from IEEE Xplore. Restrictions apply.