Coded Matrix Chain Multiplication

Xiaodi Fan*, Angel Saldivia†, Pedro Soto*, Jun Li‡
*Graduate Center, City University of New York
†School of Computing and Information Sciences, Florida International University
‡Queens College and Graduate Center, City University of New York

***Abstract*—The matrix multiplication is a fundamental building block in many machine learning models. As the input matrices may be too large to be multiplied on a single server, it is common to split input matrices into multiple submatrices and execute the multiplications on different servers. However, in a distributed infrastructure it is common to observe stragglers whose performance is lower than other servers at some time. In order to mitigate the adversarial effects of potential stragglers, various coding schemes for the distributed matrix multiplication have been recently proposed. While most existing works have only considered the simplest case where only two matrices are multiplied, we investigate a more general case in this paper where multiple matrices are multiplied, and propose a coding scheme that the result can be directly decoded in one round, instead of in multiple rounds of computation. Compared to completing the matrix chain multiplication in multiple rounds, our coding scheme can achieve significant savings of completion time by up to 90.3%.**

I. Introduction

The matrix multiplication is a fundamental operation for solving various learning-based problems. With the ever growing sizes of learning models and datasets, the sizes of the matrix multiplication in the models are also increasing. It has become challenging to execute the matrix multiplication on a single server when input matrices are from large datasets. Therefore, it is common to split the job of matrix multiplication to multiple tasks which can be executed on different servers in parallel.

However, it is well known that servers in a distributed infrastructure, *e.g.*, in a cloud, can exhibit faulty behaviors [1] due to load imbalance, resource contention, or hardware issues, *etc*. Therefore, if some tasks are running on such servers, *i.e.*, stragglers, they will become the bottleneck of the job. Even one single straggler can significantly slow down the overall progress of the whole job, as the completion of the whole job depends on the completion of all of its tasks.

A naive method that mitigates the adversarial effects of stragglers is to replicate each task on multiple servers, so that the job can be completed as long as one of them runs on a non-straggling server. However, it incurs an excessive amount of resource consumption. To tolerate any r stragglers, all tasks need to be replicated on $r+1$ servers. On the other hand, coding-based methods have been proposed where the result of the job can be decoded from a certain number of *coded* tasks [2], [3], [4]. As illustrated in Fig. 1a, in order to calculate AX, we first split A into two submatrices A_0 and A_1 so that $AX = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} X = \begin{bmatrix} A_0X \\ A_1X \end{bmatrix}$. Then one additional coded task can be created as $(A_0 + A_1)X$, such that any two of the three tasks can recover the result of AX. Compared to replication which needs two additional tasks to tolerate one single straggler, we save the number of workers by 25% in Fig. 1b. Hence, the coded matrix multiplication enjoys a higher level of straggler tolerance with much fewer additional tasks.

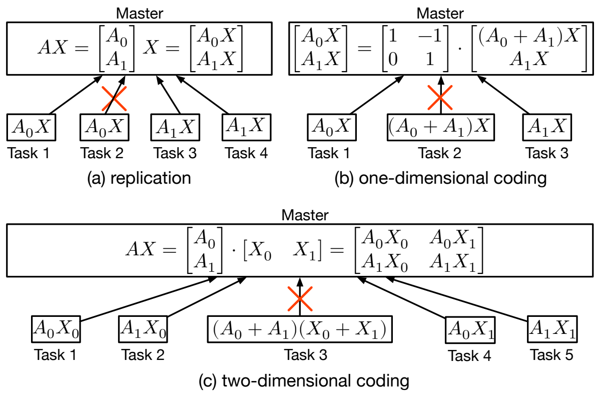
Master
$AX = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} X = \begin{bmatrix} A_0X \\ A_1X \end{bmatrix}$
A_0X Task 1
A_0X Task 2
A_1X Task 3
A_1X Task 4
(a) replication
Master
$\begin{bmatrix} A_0X \\ A_1X \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} (A_0+A_1)X \\ A_1X \end{bmatrix}$
A_0X Task 1
$(A_0+A_1)X$ Task 2
A_1X Task 3
(b) one-dimensional coding
Master
$AX = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} \cdot [X_0 \quad X_1] = \begin{bmatrix} A_0X_0 & A_0X_1 \\ A_1X_0 & A_1X_1 \end{bmatrix}$
A_0X_0 Task 1
A_1X_0 Task 2
$(A_0+A_1)(X_0+X_1)$ Task 3
A_0X_1 Task 4
A_1X_1 Task 5
(c) two-dimensional coding

Fig. 1. Examples of the coded matrix multiplication.

In order to reduce the size of the task, existing works for the coded matrix multiplication have been evolved from one-dimensional coding (*e.g.*, [2], [3], [5]) where only one matrix is split as shown in Fig. 1b, to two-dimensional coding (*e.g.*,[6], [7], [8]) where both two matrices can be split, one vertically and the other horizontally, as illustrated in Fig. 1c. Comparing the tasks in Fig. 1b and those in Fig. 1c, we can see that the size of the tasks in Fig. 1c is further reduced by 50%. On the other hand, we need to have five tasks to tolerate one straggler in Fig. 1c, as the overall result needs to be decoded from any four tasks.

More generally, the two input matrices can be split both horizontally and vertically into $p_0 \times p_1$ and $p_1 \times p_2$ submatrices. Fig. 1c, for example, corresponds to a special case of $p_0 = p_2 = 2$ and $p_1 = 1$. A coded task will then multiply two coded matrices that are linear combinations of submatrices in the two input matrices. Therefore, we can see that the input matrices can be divided into more and more submatrices of smaller sizes, and then each task can be computed with less time. To the best of our knowledge, the *recovery threshold*, *i.e.*, the number of tasks required for the recovery of the result of the matrix multiplication is $p_0p_1p_2 + p_1 - 1$ [4], [9].

However, existing works on the coded matrix multiplication have been focusing on the multiplication of only two matrices, while in practice there are various learning-based algorithms requiring the result of the matrix chain multiplication, *i.e.*, multiplying multiple matrices together.

With existing coding techniques above, the matrix chain multiplication can only be completed by multiple rounds of matrix multiplications. At least one input matrix in each round must be based on the result of the previous round. Although the sequence of multiplication can be determined using dynamic programming to minimize the overall computational complexity [10], the result of each round still needs to be decoded and encoded again for the next round, making the job completion time increase linearly with the number of rounds.

In this paper, we propose a general coding framework for the matrix chain multiplication where the job can be finished with just one round of tasks. Assume that there are m matrices M_i, $i = 0, \dots, m-1$, and we aim to calculate their multiplication $\prod_{i=0}^{m-1} M_i$. Although Dutta *et al.* [9] have also discussed coding for the matrix chain multiplication, the input matrices must be partitioned with specific patterns. Our coding scheme, instead, supports to split the matrix in a more general way, where M_i can be split into any p_i partitions vertically and p_{i+1} partitions horizontally, and hence M_i will be divided into p_ip_{i+1} submatrices, *i.e.*,

$$M_i = \begin{bmatrix} M_i^{0,0} & \cdots & M_i^{0,p_{i+1}-1} \\ \vdots & \ddots & \vdots \\ M_i^{p_i-1,0} & \cdots & M_i^{p_i-1,p_{i+1}-1} \end{bmatrix}.$$

Each task will then be a chain multiplication of m coded matrices encoded from the submatrices in $M_0, \dots, M_{m-1}$.[1] We prove that with our coding scheme, the recovery threshold to recover the overall result of $\prod_{i=0}^{m-1} M_i$ is $\prod_{i=0}^{m+1} p_i + \prod_{i=1}^{m} p_i - 1$. In particular, we will see that the coding scheme proposed in [4] can be considered as a special case of $m = 2$.

II. Background: Entangled Polynomial Code ($m = 2$)

Before demonstrating our coding scheme for the matrix chain multiplication, we first give a brief review of entangled polynomial codes [4], a special case of our coding scheme with $m = 2$. We will construct our code for the matrix chain multiplication based on entangled polynomial codes. In this case, the multiplication of M_0 and M_1 can be written as

$$M_0M_1 = \begin{bmatrix} \sum_{x_1=0}^{p_1-1} M_0^{0,x_1}M_1^{x_1,0} & \cdots & \sum_{x_1=0}^{p_1-1} M_0^{0,x_1}M_1^{x_1,p_2-1} \\ \vdots & \ddots & \vdots \\ \sum_{x_1=0}^{p_1-1} M_0^{p_0-1,x_1}M_1^{x_1,0} & \cdots & \sum_{x_1=0}^{p_1-1} M_0^{p_0-1,x_1}M_1^{x_1,p_2-1} \end{bmatrix},$$

where we can see that there are p_0p_2 submatrices. With an entangled polynomial code, coded tasks are constructed to obtain such p_0p_2 submatrices. Each server runs a task that calculates $f_0(X)f_1(X)$, where the value of X is different from that in any other tasks. In particular,

$$f_0(X) = \sum_{x_0=0}^{p_0-1}\sum_{x_1=0}^{p_1-1} M_0^{x_0,x_1} X^{p_1p_2x_0+x_1},$$

and

$$f_1(X) = \sum_{x_2=0}^{p_2-1}\sum_{x_1=0}^{p_1-1} M_1^{p_1-1-x_1,x_2} X^{p_1x_2+x_1}.$$

Therefore, we have

$$f_0(X)f_1(X) = \sum_{x_0=0}^{p_0-1}\sum_{x_2=0}^{p_2-1}\sum_{s_1=0}^{2p_1-2}\left(\sum_{x_1=\max\{0,s_1-p_1+1\}}^{\min\{p_1-1,s_1\}} M_0^{x_0,x_1}M_1^{p_1-1-s_1+x_1,x_2}\right)X^{p_1(p_2x_0+x_2)+s_1}. \quad (1)$$

From (1), we can see that $f_0(X)f_1(X)$ is a polynomial function of X of degree $p_0p_1p_2 + p_1 - 2$. Therefore, we can decode the coefficients of $f_0(X)f_1(X)$ with $p_0p_1p_2 + p_1 - 1$ such tasks, by a polynomial interpolation algorithm or Gaussian elimination. In particular, in (1), the coefficients of $X^{p_1(p_2x_0+x_2)+s_1}$ with $s_1 = p_1 - 1$ are $\sum_{x_1=0}^{p_1-1} M_0^{x_0,x_1}M_1^{x_1,x_2}$. Therefore, we can obtain the p_0p_2 desired submatrices in M_0M_1 after decoding.

It is interesting to note that when $s_1 \neq p_1 - 1$, the corresponding coefficients are *noise* coefficients, *i.e.*, they are not needed after decoding. As shown in Fig. 2, we cover the exponents of the terms in $f_0(X)f_1(X)$ with some specific values of x_0 and x_1. We can see that such exponents range between $p_1p_2p_0 + p_1x_2$ and $p_1p_2x_0 + p_1x_2 + 2p_1 - 2$. In particular, the term with the exponent $p_1p_2x_0 + p_1x_2 + p_1 - 1$, which corresponds to $s_1 = p_1 - 1$, has its coefficient as a desired submatrix $\sum_{x_1=0}^{p_1-1} M_0^{x_0,x_1}M_1^{x_1,x_2}$.

In $f_0(X)f_1(X)$, the exponents of noise coefficients will not interfere with the other desired coefficients with $s_1 = p_1 - 1$, although the values of s_1 can range between 0 and $2p_1 - 2$. If we compare the terms of (x_0, x_2) with those of $(x_0, x_2 - 1)$, we can see that the exponents of $(x_0, x_2 - 1)$ are all smaller than the exponent that corresponds to the desired submatrix, if $x_2 > 0$. On the other hand, the exponents of $(x_0, x_2 + 1)$ are all larger than that of the desired submatrix if $x_2 < p_2 - 1$. Hence, no other terms in $f_0(X)f_1(X)$ will have the same

[1] Since a coded task still calculates the matrix chain multiplication, dynamic programming can also be applied on each task to minimize its complexity, regardless of the coding scheme. Therefore, we focus on the coding scheme only in this paper, instead of the order of multiplication in each task.

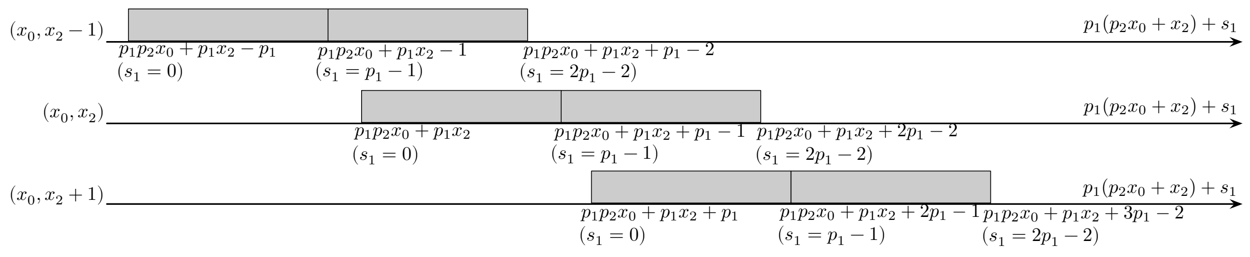

Fig. 2. Entangled exponents of X in $f_0(X)f_1(X)$.

exponent as $p_1p_2x_0 + p_1x_2 + p_1 - 1$. We can also get the same result if $x_2 = 0$ or $x_2 = p_2 - 1$, *i.e.*, when $(x_0, p_2 - 1)$ goes to $(x_0+1, 0)$. Therefore, all desired submatrices with all possible values of (x_0, x_2) will also have unique exponents, making sure that their values can be correctly obtained after decoding.

Moreover, the exponents of noise coefficients can overlap so that the degree of the polynomial can be reduced. In Fig. 2, we can see that except the exponent corresponding to $s_1 = p_1-1$, all the other exponents around can be matched with the same exponent above with (x_0, x_2-1) or lower with (x_0, x_2+1). As their corresponding coefficients are noise, they can be added together without affecting the overall result after decoding. Therefore, compared to making all terms with different values of (x_0, x_2) have different exponents, entangled polynomial codes save the overall degree of $f_0(X)f_1(X)$ and thus also helps to achieve a low recovery threshold. In this paper, we will further utilize this property in order to achieve a low recovery threshold in the coded matrix chain multiplication.

III. Coded Chain Multiplication of Three Matrices ($m = 3$)

We now start to construct our coding scheme for the matrix chain multiplication. For simplicity, we first present the construction for a special case of multiplying three matrices, *i.e.*, $m = 3$. We will present the code construction for a general value of m in Sec. IV.

We show that the code for the multiplication of three matrices can be extended from the entangled polynomial code. Considering the case of $M_0M_1M_2$, a coded task will then be constructed as $f_0(X)f_1(X)f_2(X)$. Here, $f_0(X)$ and $f_1(X)$ remain the same as constructed with the corresponding entangled polynomial code constructed for M_0M_1, and we will now present how to construct $f_2(X)$.

Similar to the case of $m = 2$ in Sec. II, there will be p_0p_3 submatrices in $M_0M_1M_2$. In particular, if we define $\hat{M}_2 = M_0M_1$, we can divide $\hat{M}_2$ into p_0p_2 submatrices and have

$$M_0M_1M_2 = \hat{M}_2M_2 =$$
$$\begin{bmatrix} \sum_{x_2=0}^{p_2-1} \hat{M}_2^{0,x_2} M_2^{x_2,0} & \cdots & \sum_{x_2=0}^{p_2-1} \hat{M}_2^{0,x_2} M_2^{x_2,p_3-1} \\ \vdots & \ddots & \vdots \\ \sum_{x_2=0}^{p_1-1} \hat{M}_2^{p_0-1,x_2} M_2^{x_2,0} & \cdots & \sum_{x_2=0}^{p_2-1} \hat{M}_2^{p_0-1,x_2} M_2^{x_2,p_3-1} \end{bmatrix},$$

where $\hat{M}_2^{x_0,x_2} = \sum_{x_1=0}^{p_1-1} M_0^{x_0,x_1} M_1^{x_1,x_2}$, $0 \le x_0 \le p_0-1$ and $0 \le x_2 \le p_2 - 1$.

We will now discuss how to construct $f_2(X)$ to obtain such p_0p_3 submatrices. Considering (1), we can rewrite $f_0(X)f_1(X)$ as:

$$\begin{aligned} & f_0(X)f_1(X) \\ &= \sum_{x_0=0}^{p_0-1} \sum_{x_2=0}^{p_2-1} \sum_{s_1=0}^{2p_1-2} C_2(x_0, x_2, s_1) X^{p_1(p_2x_0+x_2)+s_1} \\ &= \sum_{s_1=0}^{2p_1-2} \left(\sum_{x_0=0}^{p_0-1} \sum_{x_2=0}^{p_2-1} C_2(x_0, x_2, s_1) X^{p_1(p_2x_0+x_2)+s_1} \right), \end{aligned}$$

where $C_2(x_0, x_2, s_1) \triangleq \sum_{x_1=\max\{0,s_1-p_1+1\}}^{\min\{p_1-1,s_1\}} M_0^{x_0,x_1} M_1^{p_1-1-s_1+x_1,x_2}$. From Sec. II, we know that we are interested in the value of $C_2(x_0, x_2, s_1)$ if $s_1 = p_1 - 1$. Therefore, the submatrices in $\hat{M}_2$ have been encoded in $f_0(X)f_1(X)$ as

$$\begin{aligned} \hat{f}_2(X) &= \sum_{x_0=0}^{p_0-1} \sum_{x_2=0}^{p_2-1} C_2(x_0, x_2, p_1 - 1) X^{p_1(p_2x_0+x_2)+p_1-1} \\ &= \sum_{x_0=0}^{p_0-1} \sum_{x_2=0}^{p_2-1} \hat{M}_2^{x_0,x_2} X^{p_1(p_2x_0+x_2)+p_1-1}. \end{aligned}$$

Now we reapply entangled polynomial codes to $\hat{M}_2M_2$, and encode M_2 as $f_2(X) = \sum_{x_3=0}^{p_3-1} \sum_{x_2=0}^{p_2-1} M_2^{p_2-1-x_2,x_3} X^{p_0p_1p_2x_3+p_1x_2}$. We will then get

$$\begin{aligned} \hat{f}_2(X)f_2(X) = \sum_{x_0=0}^{p_0-1} \sum_{x_3=0}^{p_3-1} \sum_{s_2=0}^{2p_2-2} & \left(\sum_{x_2=\max\{0,s_2-p_2+1\}}^{\min\{p_2-1,s_2\}} \right. \\ & \left. \hat{M}_2^{x_0,x_2} M_2^{p_2-1-s_2+x_2,x_3} \right) X^{p_1(p_0p_2x_3+p_2x_0+s_2)+(p_1-1)}. \end{aligned}$$

Similarly, we are interested in the coefficients where $s_2 = p_2 - 1$. Therefore, the desired coefficients in $f_0(X)f_1(X)f_2(X)$ are those with $s_1 = p_1-1$ and $s_2 = p_2-1$:

$$\begin{aligned} f_0(X)f_1(X)f_2(X) = \sum_{x_0=0}^{p_0-1} \sum_{x_3=0}^{p_3-1} \sum_{s_2=0}^{2p_2-2} \sum_{s_1=0}^{2p_1-2} & \\ \left(\sum_{x_2=\max\{0,s_2-p_2+1\}}^{\min\{p_2-1,s_2\}} C_2(x_0, x_2, s_1) M_3^{p_2-1-s_2+x_2,x_3} \right) & \\ X^{p_1(p_0p_2x_3+p_2x_0+s_2)+s_1}. & \quad (2) \end{aligned}$$

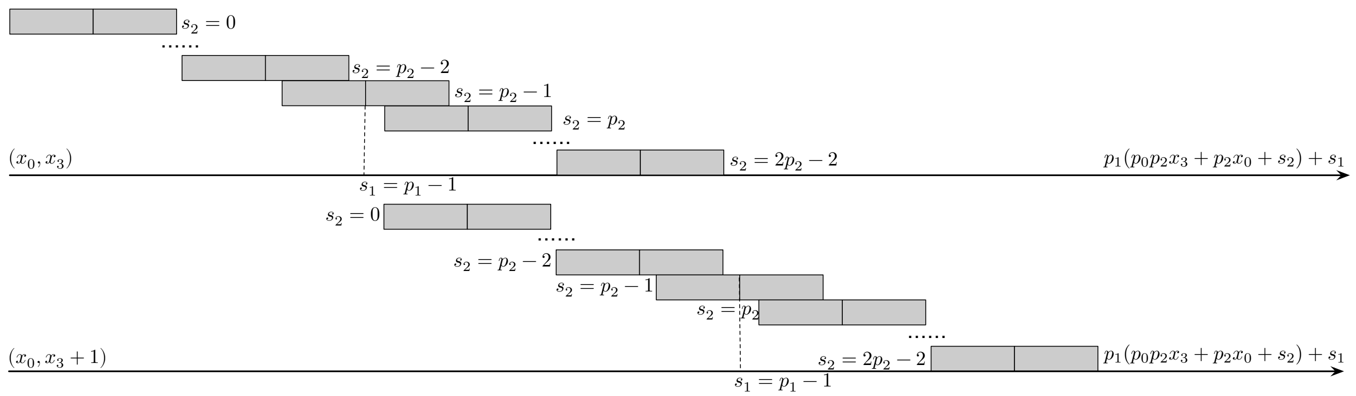

Fig. 3. Entangled exponents of X in $f_0(X)f_1(X)f_2(X)$.

We show how exponents of $f_0(X)f_1(X)f_2(X)$ are entangled in Fig. 3. Given a fixed (x_0, x_3), we can see that the exponents of the corresponding terms are entangled in the same way as entangled polynomial codes. In this case, however, the exponents are also further entangled with the exponents with $(x_0, x_3 + 1)$. Similar to entangled polynomial codes, although $p_1s_2 + s_1$ can vary from 0 to $2p_1p_2 - 2$, and the exponents of X in (2) will increase by p_1p_2 when (x_3, x_0) goes to $(x_3, x_0 + 1)$, we can still see that the desired coefficients in the middle will not interfere with noise coefficients around. As shown in Fig. 3, given x_0 and x_3, the only desired coefficient has $s_1 = p_1 - 1$ and $s_2 = p_2 - 1$, while all other noise coefficients can overlap with each other with different values of s_2. When we change x_3 to $x_3 + 1$, the exponents of coefficients with $s_2 \neq p_2 - 1$ will further be entangled with those of previous coefficients of (x_0, x_3), while the desired coefficient still enjoys its unique exponent.

Given the entangled exponents above, we can see that the degree of $f_0(X)f_1(X)f_3(X)$ is $p_0p_1p_2p_3 + p_1p_2 - 2$, as the exponents of X range from 0 to $p_0p_1p_2(p_3 - 1) + p_1p_2(p_0 - 1) + p_1(2p_2 - 2) + (2p_1 - 2) = p_0p_1p_2p_3 + p_1p_2 - 2$. Therefore, the recovery threshold is $p_0p_1p_2p_3 + p_1p_2 - 1$, and the desired submatrices can be found in p_0p_3 of its coefficients with $s_1 = p_1 - 1$ and $s_2 = p_2 - 1$.

IV. General Matrix Chain Multiplication

We now generalize the code construction for matrix chain multiplication with any m matrices, $m \geq 2$. We define a coding function Ω_m that generates $f_0(X), \ldots, f_{m-1}(X)$, *i.e.*, $(f_0(X), \ldots, f_{m-1}(X)) = \Omega_m(M_0, \ldots, M_{m-1})$.

Following the method in Sec. III, the general coding function Ω_m can be constructed recursively in Alg. 1. We define $P_a^b = \prod_{i=a}^{b} p_i$. In particular, if $a > b$, we define $P_a^b = 1$.

Note that $\Omega_m(M_0, \ldots, M_{m-1}) = f_0(X), \ldots, f_{m-1}(X)$ can be constructed before encoding and then the encoding process will be directly evaluating the value of such m polynomials with a unique value of X, and we will obtain a coded task $F_m(X) = \prod_{i=0}^{m-1} f_i(X)$. Moreover, the coefficients in $F_m(X)$ can also be decoded by interpolation or Gaussian elimination, as in the entangled polynomial code.

Algorithm 1 Construction of $\Omega_m(M_0, \ldots, M_{m-1})$

```
1: **if** $m = 2$ **then**
2:     $f_0(X) = \sum_{x_0=0}^{p_0-1} \sum_{x_1=0}^{p_1-1} M_1^{x_0,x_1} X^{p_1p_2x_0+x_1}$
3:     $f_1(X) = \sum_{x_2=0}^{p_2-1} \sum_{x_1=0}^{p_1-1} M_2^{p_1-1-x_1,x_2} X^{p_1x_2+x_1}$
4:     **return** $(f_0(X), f_1(X))$
5: **else if** $m$ is odd **then**
6:     $(f_0(X), \ldots, f_{m-2}(X)) = \Omega_{m-1}(M_0, \ldots, M_{m-2})$
7:     $f_{m-1}(X) = \sum_{x_m=0}^{p_m-1} \sum_{x_{m-1}=0}^{p_{m-1}-1} M_{m-1}^{p_{m-1}-1-x_{m-1},x_m}$
8: $X^{P_0^{m-1}x_m+P_1^{m-2}x_{m-1}}$
9:     **return** $(f_0(X), \ldots, f_{m-1}(X))$
10: **else if** $m$ is even **then**
11:     $(f_1(X), \ldots, f_{m-1}(X)) = \Omega_{m-1}(M_1, \ldots, M_{m-1})$
12:     $f_0(X) = \sum_{x_0=0}^{p_0-1} \sum_{x_1=0}^{p_1-1} M_0^{p_1-1-x_1,x_0} X^{P_1^m x_0+P_2^{m-1}x_1}$
13:     **return** $(f_0(X), \ldots, f_{m-1}(X))$
```

From Alg. 1, we can see that $F_m(X)$ is still a polynomial function of X, and thus we can write $F_m(X)$ as

$$F_m(X) = \sum_{x_0=0}^{p_0-1} \sum_{x_m=0}^{p_m-1} \sum_{s_1=0}^{2p_1-2} \cdots \sum_{s_{m-1}=0}^{2p_{m-1}-2} C_m(x_0, x_m, s_1, \ldots, s_{m-1}) X^{R_m(x_0,x_m,s_1,\ldots,s_{m-1})},$$

where $C_m(x_0, x_m, s_1, \ldots, s_{m-1})$ and $R_m(x_0, x_m, s_1, \ldots, s_{m-1})$ denote the coefficient and exponent of each term, respectively. We now analyze the recovery threshold and the correctness of Alg. 1 in the two theorems below.

Theorem 1. *In $F_m(X)$, $0 \leq R_m(x_0, x_m, s_1, \ldots, s_{m-1}) \leq P_0^m + P_1^{m-1} - 2$.*

Theorem 2. *For any $m \geq 2$, the p_0p_m submatrices in $\prod_{i=0}^{m-1} M_i$ can be uniquely found in $C_m(x_0, x_m, s_1, \ldots, s_{m-1})$ when $s_i = p_i - 1$, $i = 1, \ldots, m-1$.*

From Theorem 1, we can see that the recovery threshold of $F_m(X)$ is $P_0^m + P_1^{m-1} - 1$. Furthermore, by Threorem 2, the results of the chain matrix multiplication can be correctly obtained from p_0p_m desired coefficients in $F_m(X)$. The proof of the two theorems above can be found in the full version of the paper [11].

V. Evaluation

In this section, we present our empirical results of running the coded matrix multiplication in a cluster of virtual machines hosted on Microsoft Azure. All coded tasks run on virtual machines of type B1. The job is controlled by another virtual machine of type B4 as a master, which also decodes the results of tasks as decoding requires more memory than each task.

We implement our coding scheme (chain) for the matrix chain multiplication with OpenMPI. The m coded matrices in $\Omega_m(M_0, \dots, M_{m-1})$ are initially stored on each worker. Each worker multiplies such m coded matrices and uploads the result to the master. The master keeps polling if there is any new result sent from a worker, terminates all remaining tasks once the number of received results reaches the recovery threshold, and then decodes the results. In our experiments, we decode the results with Gaussian elimination. Note that we only need to obtain the $p_0 p_m$ desired coefficients in $F_m(X)$, and thus the decoding will be stopped once we get such desired coefficients in order to save time.

As a comparison, we implement another scheme EP (partition) which completes coded chain multiplication in multiple rounds, each of which is encoded with an entangled polynomial code. To make a fair comparison, we also first store each m matrices encoded with entangled polynomial codes on each worker. In the first round, each worker multiplies the two coded matrices in $\Omega_2(M_0, M_1)$, and the master will obtain $P_1 = M_0 M_1$. Then the master will only encode P_1 as in $\Omega_2(P_1, M_2)$, as M_2 has already been encoded, and sends coded matrices to each worker. In this round, $P_2 = P_1 M_2$ will be calculated and the master will also encode P_2, and so on until all m input matrices have been multiplied at the end of the $(m-1)$-th round.

We run jobs with the two schemes above, which multiply m random matrices of the same size with $m = 3$, 4, and 5, respectively. In these jobs, the sizes of the m matrices are 2000×2000 and 4000×4000. Each matrix is split both vertically and horizontally into 2 partitions, *i.e.*, $p_0 = \cdots = p_m = 2$. Therefore, the recovery thresholds for the jobs in chain are 19 ($m = 3$), 39 ($m = 4$), and 79 ($m = 5$). The number of workers is then chosen as the sum of the corresponding recovery threshold and 5 additional workers, such that at most 5 stragglers can always be tolerated. When running the job with entangled polynomial codes in multiple rounds, the number of workers in each round will be chosen such that the same number of stragglers can be tolerated.

Although EP (partition) maintains the same partitions as chain, the entangled polynomial code has a much lower recovery threshold and thus require much fewer workers if the same number of stragglers need to be tolerated. Hence, we run the same jobs with one more scheme EP (worker), which is also based on the entangled polynomial code, by increasing the number of partitions of input matrices in each round so that the same number of workers will be required as chain. The numbers of partitions in the two input matrices in each round will be 2×4 and 4×2 ($m = 3$), 3×4 and 4×3 ($m = 4$), and 3×8 and 8×3 ($m = 5$). If the rows or columns of a matrix cannot be equally divisible by the number of partitions, we will add additional zero rows or columns at the end of the matrix.

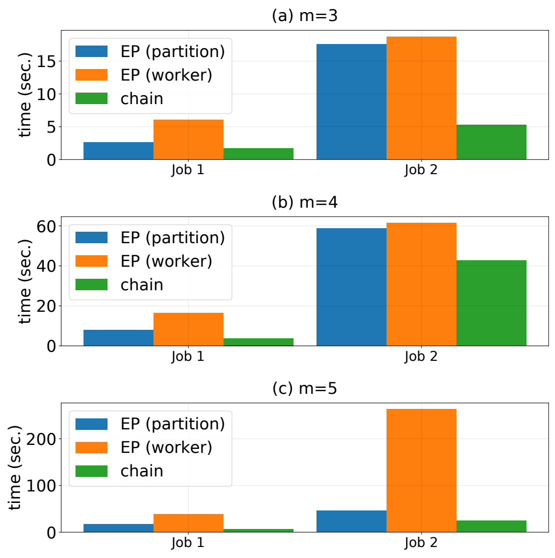

Fig. 4. Comparison of the job completion time of the matrix chain multiplications.

In Fig. 4, we present the job completion time of coded chain multiplication in the two jobs with the three schemes above, *i.e.*, chain, EP (partition), and EP (worker). We run each job 60 times, and show the average of its job completion time in Fig. 4. We can see that our coding scheme can significantly save the overall job completion time by at most 90.3%. Compared to EP (partition), although more tasks are allowed with our coding scheme, making the parallelism of the job increase, the most saving of time comes from the communication overhead, as with entangled polynomial codes the intermediate results should be uploaded to the master and then be encoded for the next round. Moreover, although EP (worker) enjoys the same level of parallelism as chain, its high communication overhead actually becomes its bottleneck, and we can see that its job completion time is the worst. Due to the space limit, the experiment results about the communication overhead can be found in the full version [11].

We now use the coded matrix chain multiplication to solve a linear regression problem in a distributed manner. The problem is modeled as $\min_x f(x) \triangleq \min_x \frac{1}{2}||Ax - y||^2$, where $y \in \mathbb{R}^q$ is the label vector, $A \in \mathbb{R}^{q \times r}$ is the matrix of the dataset, and $x \in \mathbb{R}^r$ is the unknown weight vector to be trained. We solve the linear regression problem with gradient descent. After initializing the weight vector as $x^{(0)}$, we update it iteratively as $x^{(t+1)} = x^{(t)} - \gamma \nabla f(x^{(t)}) = x^{(t)} - \gamma A^T(Ax^{(t)} - y)$, $t \geq 0$. We can then observe that each step can be completed by two matrix multiplications, *i.e.*, $g^{(t)} \triangleq Ax^{(t)}$, and $A^T(g^{(t)} - y)$. To tolerate potential stragglers, we can use entangled polynomial codes to encode A and $x^{(t)}$ in the first matrix multiplication, and A^T and $g^{(t)} - y$ in the second matrix multiplication, and then proceed to the next step. As A and A^T do not change in each step, we can place their coded matrices on each worker before the job starts, and hence only $x^{(t)}$ and $g^{(t)} - y$ need to be encoded and sent to all workers.

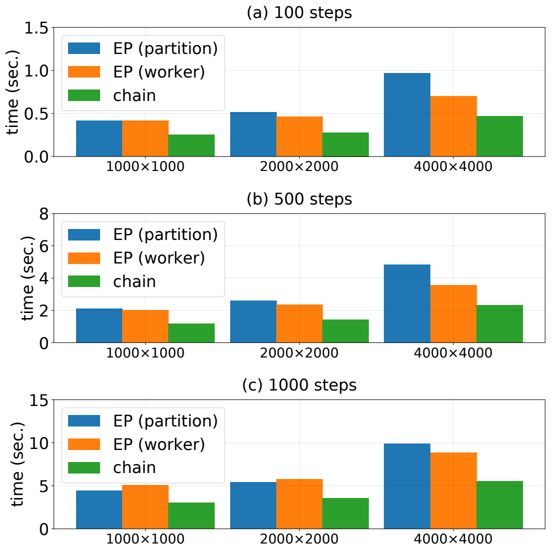

Fig. 5. Comparison of the job completion time of the linear regression.

The algorithm above was also used in [2], [12], requiring two rounds of matrix multiplications in each step. However, we observe that $x^{(t+1)}$ can be written as $x^{(t+1)} = x^{(t)} - \gamma A^T A x^{(t)} + \gamma A^T y$. Since A and y are constant, we only need to compute $A^T y$ once. In each step, we only need to compute a matrix chain multiplication $A^T A x^{(t)}$. Compared to existing solvers based on the distributed matrix multiplication, the number of matrix multiplications in each step is saved from two to one. Still, A and A^T are encoded and placed on each worker before the job starts, and only $x^{(t)}$ need to be encoded and sent to all worker per step.

In our experiment, we run the same jobs using the two methods above, *i.e.*, EP and chain. We use coded matrix chain multiplication in chain. In chain, A and A^T are both partitioned into 2×2 submatrices. As $x^{(t)}$ is a vector, we just split it into 2 partitions horizontally. The same to Fig. 4, we split the matrices in two ways in EP. EP (partition) partitions the input matrices in the same way as in chain, and EP (worker) increases the input matrices so that the recovery threshold equals that in chain, *i.e.*, A and A^T are partitioned into 4×2 submatrices and $x^{(t)}$ is still split into 2 partitions. We repeat each job 20 times, and obtain their average time of completion. The sizes of the dataset matrix A are randomly generated in three different sizes, 1000×1000, 2000×2000, and 4000×4000, as well as the label vector y with the corresponding sizes.

Fig. 5 illustrates the completion time of the jobs running with the three schemes above. Comparing to EP (partition), chain is faster by up to 51.7%. EP (worker) is also a bit faster than EP (partition), by up to 27.29%, due to its higher parallelism. Although EP (worker) enjoys the same recovery threshold as chain, chain is still faster since each step only requires one chain multiplication, leading to a higher level of parallelism since the two matrix multiplications in each step in EP (worker) can only be done sequentially.

VI. Conclusion

Coded computing for the distributed matrix multiplication have been demonstrated to efficiently tolerate stragglers. However, existing coding schemes proposed so far have only considered the multiplication of two matrices, and we consider the matrix chainmultiplication in this paper. As the existing coded matrix multiplication can only multiply two matrices each time, with which the chain matrix multiplication needs to be completed in multiple rounds, we propose a coding scheme for the matrix chain multiplication with a general number of matrices multiplied, which allows to complete the chain multiplication in one single round.

Acknowledgment

This paper is based on work supported by the National Science Foundation under Grant No. CCF-2101388.

References

[1] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray Failure: The Achilles' Heel of Cloud-Scale Systems," in *USENIX Conference on Hot Topics in Operating Systems (HotOS)*, 2017.
[2] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.
[3] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 2100–2108.
[4] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler Mitigation in Distributed Matrix Multiplication: Fundamental Limits and Optimal Coding," in *IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2022–2026.
[5] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded Computation over Heterogeneous Clusters," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2408–2412.
[6] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing Massive-scale Distributed Matrix Multiplication with d-dimensional Product Codes," in *IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1993–1997.
[7] H. Park, K. Lee, J.-Y. Sohn, C. Suh, and J. Moon, "Hierarchical Coding for Distributed Computing," in *IEEE International Symposium on Information Theory (ISIT)*, 2018.
[8] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," *Advances in Neural Information Processing Systems (NIPS)*, 2017.
[9] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the Optimal Recovery Threshold of Coded Matrix Multiplication," *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2019.
[10] S. S. Godbole, "On Efficient Computation of Matrix Chain Products," *IEEE Transactions on Computers*, vol. 22, no. 9, pp. 864–866, 1973.
[11] X. Fan, A. Saldivia, P. Soto, and J. Li, "Coded Matrix Chain Multiplication." [Online]. Available: https://boole.cs.qc.cuny.edu/li/papers/xiaodi-iwqos21-full.pdf
[12] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded Elastic Computing," in *IEEE International Symposium on Information Theory*, 2019.