Dual Entangled Polynomial Code: Three-Dimensional Coding for Distributed Matrix Multiplication

Pedro Soto [1] **Jun Li** [1] **Xiaodi Fan** [1]

Abstract

Matrix multiplication is a fundamental building block in various machine learning algorithms. When the matrix comes from a large dataset, the multiplication can be split into multiple tasks which calculate the multiplication of submatrices on different nodes. As some nodes may be stragglers, coding schemes have been proposed to tolerate stragglers in such distributed matrix multiplication. However, existing coding schemes typically split the matrices in only one or two dimensions, limiting their capabilities to handle large-scale matrix multiplication. Three-dimensional coding, however, does not have any code construction that achieves the optimal number of tasks required for decoding, with the best result achieved by entangled polynomial (EP) codes. In this paper, we propose dual entangled polynomial (DEP) codes that require around 25% fewer tasks than EP codes by executing two matrix multiplications on each task. With experiments in a real cloud environment, we show that DEP codes can also save the decoding overhead and memory consumption of tasks.

1. Introduction

Modern machine learning algorithms play a critical role in various cognitive problems such as computer vision, natural language processing, robotics, *etc*. To achieve high performance when running a machine learning algorithm whose input is a large dataset, it is common to run such algorithms on large-scale distributed infrastructure, *e.g.*, in the cloud. By dividing a job of a machine learning algorithm into multiple tasks, modern distributed computing frameworks, such as MapReduce (Had, 2018) and Spark (Zaharia et al., 2010), have been able to process a high volume of data at the scale of tens of terabytes or more, on a cluster of nodes with limited power of CPU and memory space. Matrix multiplication, for example, is a fundamental and pervasive operation in many machine learning algorithms, which can be split into multiple tasks where each task calculates a submatrix of the result.

However, it is well known that nodes in a distributed infrastructure are subject to various faulty behaviors. For example, nodes may experience temporary performance degradation (Huang et al., 2017) due to resource contention or load imbalance, and we call such nodes *stragglers*. For example, it can be observed that virtual machines on Amazon EC2 may be $5\times$ slower than others of the same type (Tandon et al., 2017). Therefore, the performance of distributed matrix multiplication may not necessarily be improved by simply scaling out the job to more nodes, as the overall progress is more likely to be affected by unavailable or straggling nodes (Lee et al., 2018).

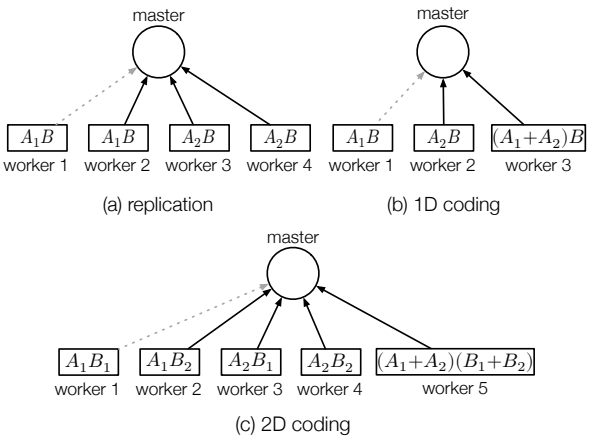


Figure 1: Examples of distributed matrix multiplication with replication, 1D coding and 2D coding.

A naive method to tolerate stragglers in distributed matrix multiplication is adding replicated tasks. Fig. 1a shows an example where the multiplication of AB is split into two tasks A_1B and A_2B and the two tasks are replicated on two nodes, respectively. Therefore, we can disregard the result of any single node when it becomes a straggler. This method,

[1]School of Computing and Information Sciences, Florida International University, Miami, Florida, USA. Correspondence to: Jun Li <junli@cs.fiu.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

however, suffers from high resource overhead. In order to tolerate any r stragglers, each task must be replicated on $r+1$ nodes. Therefore, Lee *et al.* (Lee et al., 2018) proposed the first coding scheme for distributed matrix multiplication. As show in Fig. 1b, only one additional task $(A_1 + A_2)B$ is needed, and we can still disregard the result of any single straggler.

The problem of the code in Fig. 1b is that it can only work well when the size of B is small. When the size of B is large, the overhead to compute A_iB may still be infeasible to be executed on an individual node. In order to solve this problem, Yu *et al.* (Yu et al., 2017) proposed polynomial codes, which allow both A and B to be split. Fig. 1c illustrates an example of polynomial codes where A is split horizontally into A_1 and A_2, and B is split vertically into B_1 and B_2. The multiplication of AB can then be split from such two dimensions into four tasks $A_iB_j, i \in \{1, 2\}, j \in \{1, 2\}$. Polynomial codes encode the four tasks and generate coded tasks such as $(A_1 + A_2)(B_1 + B_2)$. In this way, we can see that the resource requirement of each task is reduced to $\frac{1}{4}$ of the job, and we can also tolerate any single straggler among the five nodes, as $(A_1 + A_2)(B_1 + B_2) = \sum_{i=1}^{2}\sum_{j=1}^{2} A_iB_j$. Its number of tasks required to recover the result of the job is also proved to be optimal (Yu et al., 2017).

Polynomial codes can reduce the workload of tasks if the number of A's rows or the number of B's columns is large. However, when the number of A's columns (or equivalently the number of B's rows) is large, dividing the two matrices in two dimensions may still not make tasks small enough for individual nodes. In this case, it is only possible to make tasks small enough by dividing both A and B in all their three dimensions.[1] Assuming that the rows of A, the columns of A (or the rows of B), and the columns of B are divided into x, y, and z partitions, entangled polynomial (EP) codes (Yu et al., 2018), to the best of our knowledge, are the state-of-the-art codes that support such three-dimensional division of matrices and require $xyz + z - 1$ tasks to recover the result of AB. In this paper, we propose dual entangled polynomial (DEP) codes that require only $\frac{3}{4}xyz + \frac{z}{2} - 1$ tasks to recover the job. Because of this, we can also demonstrate that the memory requirement of each task, and the decoding complexity of DEP codes will also be lower than EP codes, when they are set to tolerate the same number of stragglers.

2. Related Work

Lee *et al.* (Lee et al., 2018), for the first time, leveraged ideas from coding theory to tolerate stragglers in distributed matrix multiplication. In their work, only one input matrix is split and encoded with an MDS (maximum distance separable) code, as shown in Fig. 1b. As the code is MDS, it achieves the optimal *recovery threshold*, *i.e.*, the result of the job can be decoded from a theoretical minimum number of *any* tasks. We name this approach as one-dimensional coding (1D coding). Following this direction, different coding schemes, such as rateless coding (Mallick et al., 2018) and sparse coding (Dutta et al., 2016) have been proposed. As nodes with slower performance in a heterogeneous cluster will always be considered as stragglers, 1D coding with heterogeneous nodes has also been discussed where different workload will be assigned to nodes based on their performance (Kiani et al., 2018; Reisizadeh et al., 2017).

As tasks with 1D coding may also be too large to be executed on individual nodes, two-dimensional coding (2D coding) makes it possible to split both two matrices in the multiplication. Initially, 2D coding schemes were constructed by reapplying 1D coding to the tasks encoded from 1D coding. In other words, a task A_iB can be further encoded into subtasks A_iB_j. In this way, a 2D coding scheme can be constructed based on product codes (Lee et al., 2017; Gupta et al., 2018; Park et al., 2018). The problem of product codes is that the optimal recovery threshold cannot be achieved. Given any i, if the results from subtasks encoded from A_iB cannot be decoded, the result of the whole job cannot be decoded as well. Yu *et al.*, on the other hand, proposed polynomial codes, a family of two-dimensional codes based on the Vandermonde matrix (Yu et al., 2017). Different from product codes, polynomial codes achieve the optimal recovery threshold. Besides MDS codes, Wang *et al.* proposed 2D coding based on sparse coding that significantly saves the decoding overhead with a near-optimal recovery threshold (Wang et al., 2018).

In a matrix multiplication AB, with 2D coding we can split A's rows and B's columns. The only dimension missed is the columns of A (or equivalently the rows of B). Yu *et al.* showed that convolution codes extended from polynomial codes can be solely applied to this dimension (Yu et al., 2017). Although three-dimensional coding (3D coding) based on product codes can be constructed combining polynomial codes and convolution codes, it suffers from a similar problem of high recovery threshold (Baharav et al., 2018). Entangled polynomial (EP) codes, on the other hand, are the first three-dimensional codes that are not based on product codes (Yu et al., 2018). The recovery threshold of EP codes is proved to be less than twice of the optimal recovery threshold of 3D coding when $x = 1$ or $y = 1$. In this paper, we propose dual entangled polynomial (DEP) codes. DEP codes require about 25% fewer tasks to recover the job than EP codes, making it far closer to the optimal recovery threshold.

[1] The columns of A and the rows of B are considered as the same dimension, as they should always be split in the same way to make the multiplication feasible.

3. Background: Entangled Polynomial Codes

In this section, we give a brief introduction of entangled polynomial (EP) codes (Yu et al., 2018), which will be useful for the construction of our dual entangled polynomial (DEP) codes. Assume that we have a job to calculate the multiplication of an $X \times Z$ matrix A and a $Z \times Y$ matrix B. Given three integers x, y, and z, where $x|X$, $y|Y$, and $z|Z$, we evenly split the rows and columns of A into x and z partitions and split the rows and columns of B into z and y partitions. Therefore, we can split A and B into xz and yz partitions. We use $A_{i,l}, 0 \le i \le x-1, 0 \le l \le z-1$, to denote the submatrix of A whose rows and columns are from the $(i+1)$-th partition of its rows and $(l+1)$-th partitions of its columns, respectively. We also define submatrices $B_{l,j}$ in the same way, $0 \le j \le y-1, 0 \le l \le z-1$. Meanwhile, the result $C = AB$ can be split into x and y partitions by its rows and columns. Similarly, we use $C_{i,j}$, $0 \le i \le x-1, 0 \le j \le y-1$, to represent its submatrices, where $C_{i,j} = \sum_{l=0}^{z-1} A_{i,l}B_{l,j}$.

We now demonstrate a property that plays an important role in the construction of EP codes. We define two functions $\tilde{A}_i(\delta) = \sum_{l=0}^{z-1} A_{i,l}\delta^l$ and $\tilde{B}_j(\delta) = \sum_{l=0}^{z-1} B_{z-1-l,j}\delta^l$. Therefore,

$$\tilde{A}_i(\delta)\tilde{B}_j(\delta) = \sum_{t=0}^{2z-2} \left(\sum_{l=\max(0,t-z+1)}^{\min(z-1,t)} A_{i,l}B_{z-1-t+l,j} \right) \delta^t.$$

In general, the coefficient of δ^t in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$ is denoted as $M(i,j,t)$, and then we can see that the coefficient of δ^{z-1}, *i.e.*, $M(i,j,z-1)$, is $\sum_{l=0}^{z-1} A_{i,l}B_{l,j} = C_{i,j}$. With this property, it is no longer needed to calculate each individual product of $A_{i,l}B_{l,j}$, as we have already got their sum. This property will also be extended in the construction of DEP codes.

In order to get $C_{i,j}$ for all feasible values of i and j, we can define a coded task $T(\delta)$, which is the multiplication of $\sum_{i=0}^{x-1} \tilde{A}_i(\delta)\delta^{\alpha i}$ and $\sum_{j=0}^{y-1} \tilde{B}_j(\delta)\delta^{\beta j}$. Therefore, we have all possible values of i and j in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$. In $T(\delta)$, the coefficients where $t = z-1$ are desirable while other coefficients are considered as *noise*. In order to decode the results, we need to choose values of α and β such that the desirable coefficients appear in different terms in the polynomial of $T(\delta)$. A natural and naive choice of α and β is $\alpha = y(2z-1)$ and $\beta = 2z-1$. In Fig. 2a, we show the corresponding exponents of δ whose coefficients are $M(i,j,t)$ with all values of i and j. We can see that the exponents of δ in $T(\delta)$ range from 0 to $xy(2z-1)-1$, and the exponents in the terms associated with all coefficients do not coincide. As long as we have the result of $T(\delta)$ from any $xy(2z-1)$ tasks with different values of δ, the results can be decoded by interpolating $T(x)$ using $xy(2z-1)$ distinct points.

In fact, the choices of α and β in Fig. 2a are not optimal, as only the exponents associated with $M(i,j,z-1)$ should be distinct. For other coefficients that do not appear in C and are considered as noise, their coefficients can be the same, in order to reduce the degree of $T(\delta)$ and the recovery threshold. EP codes, hence, choose better values of α and β where $\alpha = yz$ and $\beta = z$, and then $T(\delta) = \sum_{i,j} \tilde{A}_i(\delta)\tilde{B}_j(\delta)\delta^{yzi+zj}$ in which the exponents range from 0 to $xyz+z-2$, as shown in Fig. 2b. Moreover, we can see that when $t = z-1$, for all feasible values of i and j, the exponents of δ whose coefficients are $M(i,j,t)$ will not coincide, but most other exponents are shared by two noise coefficients. Yu *et al.* proved that when $x = 1$ or $y = 1$, the recovery threshold of EP codes $K_{\text{EP}} = xyz + z - 1$ is no more than twice the optimal number K_{opt}, *i.e.*, $K_{\text{EP}} < 2K_{\text{opt}}$. In this paper, we show that this number can be further reduced to $\frac{3}{4}xyz + \frac{z}{2} - 1$, which is less than $\frac{3}{2}K_{\text{opt}}$ when $x = 1$ or $y = 1$.

When $x = 2$, $y = 1$, and $z = 2$, we get an example of EP codes. In this case,

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix}, \quad B = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}.$$

With EP codes, we have $T(\delta) = (A_{0,0}\delta^0 + A_{0,1}\delta^1 + A_{1,0}\delta^2 + A_{1,1}\delta^3)(B_1\delta^0 + B_0\delta^1)$. If we have five tasks finished with different values of $\delta = \delta_i$, $i = 0, \ldots, 4$, we will have

$$\begin{bmatrix} \delta_0^0 & \delta_0^1 & \delta_0^2 & \delta_0^3 & \delta_0^4 \\ \delta_1^0 & \delta_1^1 & \delta_1^2 & \delta_1^3 & \delta_1^4 \\ \delta_2^0 & \delta_2^1 & \delta_2^2 & \delta_2^3 & \delta_2^4 \\ \delta_3^0 & \delta_3^1 & \delta_3^2 & \delta_3^3 & \delta_3^4 \\ \delta_4^0 & \delta_4^1 & \delta_4^2 & \delta_4^3 & \delta_4^4 \end{bmatrix} \begin{bmatrix} A_{0,0}B_1 \\ A_{0,0}B_0 + A_{0,1}B_1 \\ A_{0,1}B_0 + A_{1,0}B_1 \\ A_{1,1}B_1 + A_{1,0}B_0 \\ A_{1,1}B_0 \end{bmatrix}. \tag{1}$$

The left matrix in (1) is a Vandermonde matrix which is invertible, and thus we can decode it by multiplying its inverse on the left. After decoding, we will be able to get $AB = \begin{bmatrix} A_{0,0}B_0 + A_{0,1}B_1 \\ A_{1,1}B_1 + A_{1,0}B_0 \end{bmatrix}$.

4. Dual Entangled Polynomial Code: A Special Case

In this section, we start from a special case of DEP codes where $2|xy$ and $2|z$. The idea of DEP codes, in general, is to make the terms in the multiplication of $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$ with different values of i and j share more noise coefficients. From Fig. 2b, we can see that at most two coefficients can share the same exponent with different values of i and j. For example, the corresponding exponents of $M(0,0,z)$ in $\tilde{A}_0(\delta)\tilde{B}_0(\delta)$ and $M(0,1,0)$ in $\tilde{A}_0(\delta)\tilde{B}_1(\delta)$ are the same with EP codes, which is z as shown in Fig. 2b. In DEP codes, however, we increase this number by having at most four coefficients sharing the same exponent. Therefore, we

	$(0,0)$	$(0,1)$	$\cdots$	(i,j)	$\cdots$	$(x-1,y-1)$
$M(i,j,0)$	0	$2z-1$	$\cdots$	$(iy+j)(2z-1)$	$\cdots$	$(xy-1)(2z-1)$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,z-1)$	$z-1$	$3z-2$	$\cdots$	$(iy+j)(2z-1)+z-1$	$\cdots$	$(xy-1)(2z-1)+z-1$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,2z-2)$	$2z-2$	$4z-3$	$\cdots$	$(iy+j+1)(2z-1)-1$	$\cdots$	$xy(2z-1)-1$

(a) A naive design ($\alpha = y(2z-1), \beta = 2z-1$.)

	$(0,0)$	$(0,1)$	$\cdots$	(i,j)	$\cdots$	$(x-1,y-1)$
$M(i,j,0)$	0	z	$\cdots$	$(iy+j)z$	$\cdots$	$(xy-1)z$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,z-1)$	$z-1$	$2z-1$	$\cdots$	$(iy+j)z+z-1$	$\cdots$	$xyz-1$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,2z-2)$	$2z-2$	$3z-2$	$\cdots$	$(iy+j)z+2z-2$	$\cdots$	$xyz+z-2$

(b) Entangled polynomial codes ($\alpha = yz, \beta = z$)

Figure 2: Two examples of choices of α and β.

can further reduce the degree in $T(\delta)$ in each task, and thus achieve an even lower recovery threshold.

In order to achieve this objective, we define

$$\tilde{A}_i(\delta) = \sum_{l=0}^{\frac{z}{2}-1} A_{i,l}\delta^l + \sum_{l=\frac{z}{2}}^{z-1} A_{x-1-i,l}\delta^l,$$

and

$$\tilde{B}_j(\delta) = \sum_{l=0}^{\frac{z}{2}-1} B_{\frac{z}{2}-1-l,j}\delta^l + \sum_{l=\frac{z}{2}}^{z-1} B_{\frac{3}{2}z-1-l,y-1-j}\delta^l.$$

We also define $M(i,j,t)$ as the coefficient of δ^t in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$. In this way, we can find that the terms in $C_{i,j}$ will appear in $M(i,j,\frac{z}{2}-1)$ and $M(x-1-i,y-1-j,\frac{3}{2}z-1)$, *i.e.*,

$$M(i,j,\frac{z}{2}-1) = \sum_{l=0}^{\frac{z}{2}-1} A_{i,l}B_{l,j},$$

and

$$M(x-1-i,y-1-j,\frac{3}{2}z-1) = \sum_{l=\frac{z}{2}}^{z-1} A_{i,l}B_{l,j}.$$

In order to obtain $C_{i,j} = M(i,j,\frac{z}{2}-1) + M(z-1-i,z-1-j,\frac{3}{2}z-1)$, we need to make the exponents of these two coefficients the same. As shown in Fig. 3, the coefficients in the table can be divided into four areas. The coefficients in the two areas with the same colors will coincide so that the whole $C_{i,j}$ will eventually appear as a coefficient in some term. Meanwhile, the coefficients in the left half and the right half will also coincide. The coefficients in each half

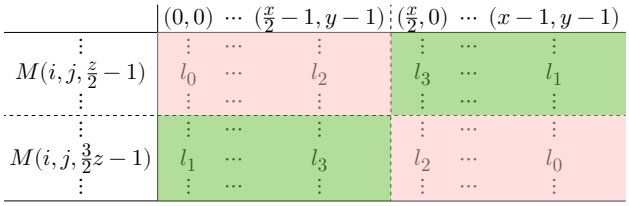


Figure 3: The general design of coefficients in DEP codes.

can be generated following the way in EP, and thus most other terms will have four noise coefficients added together.

In this way, a task will calculate a function $T(\delta) = T_1(\delta) + T_2(\delta^{-1})$, where

$$T_1(\delta) = \sum_{i=0}^{\frac{x}{2}-1} \tilde{A}_i(\delta)\delta^{\alpha i} \cdot \sum_{j=0}^{y} \tilde{B}_j(\delta)\delta^{\beta j}, \tag{2}$$

and

$$T_2(\delta) = \left(\sum_{i=\frac{x}{2}}^{x-1} \tilde{A}_i(\delta)\delta^{\alpha i} \cdot \sum_{j=0}^{y} \tilde{B}_j(\delta)\delta^{\beta j}\right) \delta^{-(\frac{x}{2}-1)i-(y-1)j}. \tag{3}$$

Note that $(\frac{x}{2}-1)i + (y-1)j$ in (3) is actually the highest exponent in (2). Moreover, in $T_2(\delta^{-1})$ the exponents will decrease as we replace δ with δ^{-1}. Therefore, each exponent in the left half in Fig. 3 can be replicated in the right half. If we choose $\alpha = \frac{3}{2}yz$ and $\beta = \frac{3}{2}z$, we will have exponents in $T(\delta)$ as shown in Fig. 4. We can see that the exponents range from 0 to $\frac{3}{4}xyz + \frac{1}{2}z - 2$. Meanwhile they are point symmetric across their center, *i.e.*, the exponent of the term whose coefficient is $M(i,j,t)$ equals that whose coefficient is $M(x-1-i,y-1-j,2z-2-t)$, $0 \le i \le x-1, 0 \le j \le y-1, 0 \le t \le 2z-2$. Especially, when

	$(0,0)$	$\cdots$	(i,j)	$\cdots$	$(\frac{x}{2}-1,y-1)$	$(\frac{x}{2},0)$	$\cdots$	$(\frac{x}{2}-1-i,y-1-i)$	$\cdots$	$(x-1,y-1)$
$M(i,j,0)$	0	$\cdots$	$\frac{3}{2}z(iy+j)$	$\cdots$	$\frac{3}{4}xyz-\frac{3}{2}z$	$\frac{3}{4}xyz+\frac{z}{2}-2$	$\cdots$	$\frac{3}{2}z(iy+j)+2z-2$	$\cdots$	$2z-2$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,\frac{1}{2}z-1)$	$\frac{z}{2}-1$	$\cdots$	$\frac{3}{2}z(iy+j)+\frac{z}{2}-1$	$\cdots$	$\frac{3}{4}xyz-z-1$	$\frac{3}{4}xyz-1$	$\cdots$	$\frac{3}{2}z(iy+j)+\frac{3}{2}z-1$	$\cdots$	$\frac{3}{2}z-1$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,\frac{3}{2}z-1)$	$\frac{3}{2}z-1$	$\cdots$	$\frac{3}{2}z(iy+j)+\frac{3}{2}z-1$	$\cdots$	$\frac{3}{4}xyz-1$	$\frac{3}{4}xyz-z-1$	$\cdots$	$\frac{3}{2}z(iy+j)+\frac{z}{2}-1$	$\cdots$	$\frac{z}{2}-1$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,2z-2)$	$2z-2$	$\cdots$	$\frac{3}{2}z(iy+j)+2z-2$	$\cdots$	$\frac{3}{4}xyz+\frac{z}{2}-2$	$\frac{3}{4}xyz-\frac{3}{2}z$	$\cdots$	$\frac{3}{2}z(iy+j)$	$\cdots$	0

Figure 4: Exponents in DEP codes ($0 \le i \le \frac{x}{2}-1, 0 \le j \le y$).

$t = \frac{z}{2}-1$ or $t = \frac{3}{2}z-1$, the corresponding exponents will also not coincide with any other exponents in the same half of the table, except their matching entry in the other half of the table. In other words, the coefficient of such term eventually will be $M(i,j,\frac{1}{2}z)+M(x-1-i,y-1-j,\frac{3}{2}z) = C_{i,j}$. Therefore, with the results of $T(\delta)$ from $\frac{3}{4}xyz+\frac{1}{2}z-1$ tasks with different values of δ, we can decode them and get each $C_{i,j}$, $0 \le i \le x, 0 \le j \le y$.

We now show an example with $x = 2, y = 1$, and $z = 2$. Following (2) and (3), we have

$$T_1(\delta) = (A_{0,0}\delta^0 + A_{1,1}\delta^1)(B_0\delta^0 + B_1\delta^1),$$

and

$$T_2(\delta) = (A_{1,0}\delta^0 + A_{0,1}\delta^1)(B_0\delta^0 + B_1\delta^1)\delta^{-2}.$$

Therefore, if we have three tasks finished where $\delta = \delta_i$, $i = 0, \ldots, 2$, we have

$$\begin{bmatrix} \delta_0^0 & \delta_0^1 & \delta_0^2 \\ \delta_1^0 & \delta_1^1 & \delta_1^2 \\ \delta_2^0 & \delta_2^1 & \delta_2^2 \end{bmatrix} \begin{bmatrix} A_{0,0}B_0 + A_{0,1}B_1 \\ N \\ A_{1,1}B_1 + A_{1,0}B_0 \end{bmatrix}, \quad (4)$$

where $N = A_{0,0}B_1 + A_{1,1}B_0 + A_{1,0}B_1 + A_{0,1}B_0$. The coefficient matrix in (4) is a Vandermonde matrix and thus (4) can be decoded in the same way as (1). Compared to the example of EP codes with the same values of x, y, and z in Sec. 3, we can add all noise coefficients together with DEP codes. Hence, we only need to have 3 tasks to complete the job instead of 5 tasks with EP codes, saving 40% nodes. Although with DEP codes each node needs to calculate two multiplications, each multiplication requires the same amount of memory as the task with EP codes, and thus DEP codes will not increase the memory requirement.

5. Generalization

We now present the general construction of DEP codes. From this general construction, we will see that the special case in Sec. 4 is not just one special case, but also the optimal case in terms of the recovery threshold.

5.1. The choice of exponents

Different from Sec. 4, we no longer require that z is even, but just a positive integer. We first study the exponents of terms in the polynomial of $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$. We already know that the exponents in this polynomial range in $[0, 2z-2]$. In the previous examples, we should place $C_{i,j} = \sum_{l=0}^{z-1} A_{i,l}B_{l,j}$, or a part of the summation into coefficients with the same exponents such that eventually they can be added back together. In EP codes, the exponent chosen for $C_{i,j}$ is $z-1$, and the special case of DEP codes in Sec. 4 chooses $\frac{z}{2}-1$ and $\frac{3}{2}z-1$.

$\times$	δ^0	δ^1	δ^2	δ^3
δ^0	0	1	2	3
δ^1	1	2	3	4
δ^2	2	3	4	5
δ^3	3	4	5	6

Figure 5: Exponents in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$ where $z = 4$.

We show an illustration of such choices in Fig. 5, where we use different colors to highlight the choices in these two case. We can observe that the entries chosen in EP codes fall in the antidiagonal, and those in DEP codes in Sec. 4 also fall in lines parallel to the antidiagonal. In fact, such choices can be generalized to all exponents as long as they are congruent modulo z. In Fig. 5 ($z = 4$), coefficients whose corresponding exponents are 2 and 6, as well as 0 and 4, can also be chosen to place the terms in $C_{i,j}$.

Therefore, there can be z choices of exponents: 0 and z, 1 and $z+1$, ..., $z-2$ and $2z-2$, and $z-1$. If the choice is $z-1$, the constructed code becomes EP codes. If the choice is $\frac{z}{2}-1$ and $\frac{3}{2}-1$, given $2|z$, the corresponding code is already presented in Sec. 4. In this section, we will present the construction with any choices of exponents for $C_{i,j}$.

If the exponents chosen for $C_{i,j}$ is p and $p+z$, $0 \le p \le z-2$,

	$(0,0)$	$(0,1)$	$\cdots$	(i,j)	$\cdots$	$(\frac{x}{2}-1,y-1)$
$M(i,j,0)$	0	$2z-p-1$	$\cdots$	$(iy+j)(2z-p-1)$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,p)$	p	$2z-1$	$\cdots$	$(iy+j)(2z-p-1)+p$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)+p$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,p+z)$	$p+z$	$3z-1$	$\cdots$	$(iy+j)(2z-p-1)+p+z$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)+p+z$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,2z-2)$	$2z-2$	$4z-p-3$	$\cdots$	$(iy+j)(2z-p-1)+2z-2$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)+2z-2$

(a) $0 \le p \le \frac{z}{2}-1$.

	$(0,0)$	$(0,1)$	$\cdots$	(i,j)	$\cdots$	$(\frac{x}{2}-1,y-1)$
$M(i,j,0)$	0	$p+z+1$	$\cdots$	$(iy+j)(p+z+1)$	$\cdots$	$(\frac{xy}{2}-1)(p+z+1)$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,p)$	p	$2p+z+1$	$\cdots$	$(iy+j)(p+z+1)+p$	$\cdots$	$(\frac{xy}{2}-1)(p+z+1)+p$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,p+z)$	$p+z$	$2p+2z+1$	$\cdots$	$(iy+j)(p+z+1)+p+z$	$\cdots$	$(\frac{xy}{2}-1)(p+z+1)+p+z$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$M(i,j,2z-2)$	$2z-2$	$p+3z-1$	$\cdots$	$(iy+j)(p+z+1)+2z-2$	$\cdots$	$(\frac{xy}{2}-1)(p+z+1)+2z-2$

(b) $\frac{z}{2}-1 \le p \le z-1$.

Figure 6: Exponents in DEP codes with general values of z.

we define

$$\tilde{A}_i(\delta) = \sum_{l=0}^{p} A_{i,l}\delta^l + \sum_{l=p+1}^{z-1} A_{x-1-i,l}\delta^l, \tag{5}$$

and

$$\tilde{B}_j(\delta) = \sum_{l=0}^{p} B_{p-l,j}\delta^l + \sum_{l=p+1}^{z-1} B_{p+z-l,y-1-j}\delta^l, \tag{6}$$

Therefore, in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$, the coefficients $M(i,j,t)$ when $t=p$ and $t=p+z$ are

$$M(i,j,p) = \sum_{l=0}^{p} A_{i,l}B_{l,j}$$

and

$$M(i,j,p+z) = \sum_{l=p+1}^{z-1} A_{x-1-i,l}B_{l,y-1-j}.$$

Hence, $C_{i,j} = M(i,j,p) + M(x-1-i, y-1-j, p+z)$.

Specifically, we have $M(i,j,p) = 0$ when $p=-1$, or $M(i,j,p+z)=0$ when $p=z-1$. Then $\tilde{A}_i(\delta)$ and $\tilde{B}_j(\delta)$ become the same as in EP codes, and hence we can directly have $M(i,j,p) = C_{i,j}$.

5.2. Construction

Now we present how to construct a task $T(\delta)$ with a general value of p. Similar to Sec. 4, we can still construct a coded task $T(\delta) = T_1(\delta) + T_2(\delta^{-1})$ from (2) and (3), with different values of α and β.

In order to minimize the recovery threshold, the value of j should also be minimized. However, the value of j should also not be too small so that the exponent of $M(i,j,p)$ is the same as that of other noise coefficients. We show in Fig. 6 the optimal choice of j. We only show the left half of the exponents, *i.e.*, when $i \le \frac{x}{2}$, as the right half will be point symmetric to the left half.

When $p \le \frac{z}{2}-1$, the number of exponents above $M(i,j,p)$ is $p-1$, no more than that below $M(i,j,p+z)$ which is $z-p-2$. Hence, j should be no less than $2z-p-1$. For example, when $(i,j)=(0,1)$, the exponent of $M(0,1,p)$ should be no less than $2z-1$. Otherwise, it will be the same with the exponent of some noise coefficient below $M(0,0,p+z)$. Therefore, $j = 2z-p+1$ when $p \le \frac{z}{2}-1$, and then $i=(2z-p+1)y$. In this way, the degree of the polynomial in $T_1(\delta)$ is $(\frac{xy}{2}-1)(2z-p-1)+2z-2$.

Similarly, when $p \ge \frac{z}{2}-1$, the number of exponents above $M(i,j,p)$ is the same with or higher than that below $M(i,j,p+z)$. Hence, the exponent of $M(0,1,0)$ can directly start from $p+z+1$, as the value of $2p+z+1$ must be higher than p, making this exponent different from $M(0,0,p)$ and $M(0,0,p+z)$. In this case, $j = p+z+1$ and $i=(p+z+1)y$. The degree of $T_1(\delta)$ is $(\frac{xy}{2}-1)(p+z+1)+2z-2$.

Moreover, if we choose $p=-1$ or $p=z-1$ in (5) and (6), all terms in $C_{i,j}$ are already added together in $M(i,j,z-1)$,

as $M(i,j,-1)$ and $M(i,j,2z-1)$ do not exist. Therefore, the dual construction cannot be applied to these two choices of p, and we have EP codes in such cases.

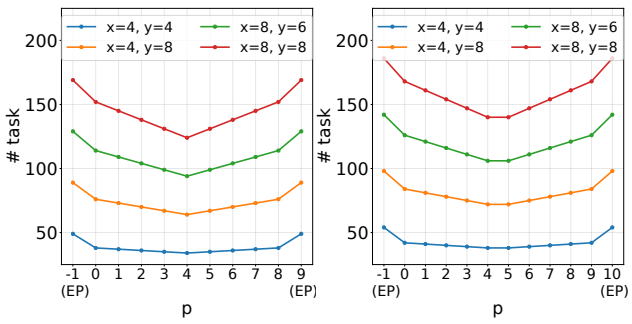
5.3. Recovery Threshold

From the general construction above, we can see that if $p \leq \frac{z}{2}-1$, the degree of $T_1(\delta)$ is minimized when $p = \frac{z}{2}-1$. This degree is also minimized to the same value when $p = \frac{z}{2}-1$ if $p \geq \frac{z}{2}-1$. The two cases above indicate that the degree of $T_1(x)$ and thus the recovery threshold is minimized, when $p = \frac{z}{2}-1$, $2|p$. We can see that this is exactly the case in Sec. 4.

We can also notice that the recovery threshold of DEP codes is symmetric to the values of p around $\frac{z}{2}-1$. Let $p_1 \leq \frac{z}{2}-1$ and $p_2 \geq \frac{z}{2}-1$. We can see that if $p_1+p_2 = z-2$, we have $2z-p_1-1 = p_2+z+1$, and thus the degrees of $T_1(\delta)$ are also the same. Therefore, if p is odd, the minimum recovery threshold is achieved when $p = \frac{z+1}{2}-1$ or $p = \frac{z-1}{2}-1$, which equals $\frac{3}{4}xyz + \frac{xy}{4} + \frac{z}{2} + 1$.

On the other hand, the highest degree is achieved when $p = 0$ or $p = z-2$, which equals $xyz - \frac{1}{2}xy$. Even in such worst cases, DEP codes still outperform EP codes.

We compare the actual number of tasks required for decoding in Fig. 7. We assume $z = 10$ in Fig. 7a, and $z = 11$ in Fig. 7b. When $p = -1$ or $p = z-1$, the corresponding data are for EP codes. With different values of x and y, we can see that DEP codes always require fewer tasks for decoding than EP codes, and save up to 30.6% tasks with the optimal value of p.



(a) $z = 10$. (b) $z = 11$.

Figure 7: Comparison of the recovery threshold.

6. Evaluation

In this section, we present our empirical results of running distributed matrix multiplication in clusters of virtual machines hosted on Microsoft Azure. All virtual machines are of type B1s, with 1 vcpu and 1 GB of memory.

We implement the coded distributed matrix multiplication with OpenMPI (ope, 2017). Coded matrices inside each task $T(\delta)$ are placed on one of the n virtual machines, *i.e.*, workers. We use one more virtual machine as a master, which controls the job and decodes results received from workers. Each worker will send its result to the master. The master, at the same time, keeps polling to check if there is any new result from a worker. If the number of received results meets the recovery threshold of the corresponding coding scheme, the master will stop receiving results of any other unfinished tasks and start to decode received results.

We first run a job to multiply two matrices AB, where the sizes of A and B are 3600×200 and 200×3600. Each job is repeated for 20 times and we demonstrate the average results. We measure three metrics: the total job completion time, the decoding time, and the number of tolerable stragglers. In each job, we launch $n = 24$ virtual machines as workers. The corresponding 24 tasks are encoded with EP codes and DEP codes, respectively. We also add one more scheme EP X2, where each worker will execute two tasks of EP codes. In this way, a worker in EP X2 will also have two multiplications, the same as DEP codes, except that the results of two tasks are not added together but directly uploaded to the master.

In Fig. 8a, we first show the number of tolerable stragglers with three configurations of (x,y,z). As DEP codes require fewer tasks to recover the job, it is natural to observe that the number of stragglers tolerable with DEP codes is higher than EP codes by up to 120%. Although EP X2 can tolerate slightly more stragglers, it does not improve the performance of the job, as it doubles the amount of data each worker needs to upload. We can see in Fig. 8b that its job completion time is even higher than EP codes. Although each worker also needs to calculate two multiplications with DEP codes, fewer tasks are required for decoding. Hence, we observe that the job completion time with DEP codes can be lower than EP codes, even though in a task with DEP codes two multiplications need to be calculated. We observe that the major bottleneck of workers is sending their results to the master. Hence, DEP codes can finish the data transmission faster, while the jobs in EP and EP X2 will have to wait for more tasks. The decoding overhead at the master, similarly, can also be significantly reduced by roughly 50%.

Another interesting observation in Fig. 8 is that increasing the number of partitions in all three schemes does not necessarily save the time of the job, especially when the value of z is increased. This is because with a higher z, the size of the result in each task (only depending on x and y) will not change, but we need to accept results from more tasks to complete the job. The decoding complexity will also be increased with more partitions. Therefore, when making tasks small enough for workers, we should always increase

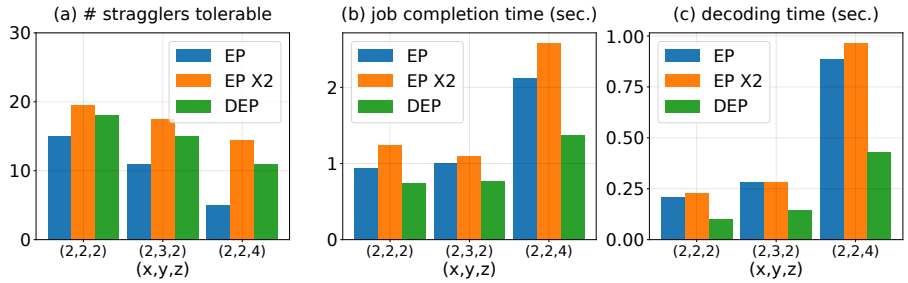


Figure 8: Performance comparison between EP and DEP codes with a fixed number of workers ($n = 24$).

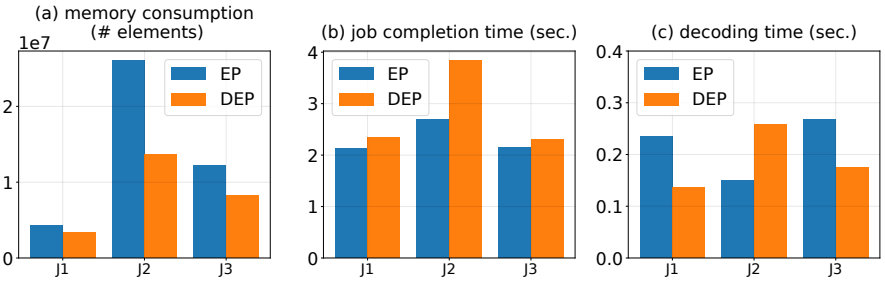


Figure 9: Performance comparison between EP and DEP codes of three different jobs.

the values of x and y before z.

We now run three more jobs encoded with EP and DEP codes. This time, the number of workers and the number of stragglers to tolerate are fixed. Specifically, we have n workers and s stragglers to tolerate. The configurations of the three jobs are shown in Fig. 10. We only compare EP and DEP in this experiment, as EP X2 will require the same amount of memory with higher job completion time.

	n	s	EP			DEP			X	Y	Z
			x	y	z	x	y	z			
J1	12	3	2	2	2	2	3	2	2.4k	2.4k	2.4k
J2	17	4	2	3	2	2	2	4	10k	0.3k	10k
J3	24	5	2	2	4	2	3	4	0.2k	9.6k	9.6k

Figure 10: Configurations of jobs.

With the same values of n and s, DEP codes allow us to split the input matrix into more partitions, lowering the memory consumption of each task which can then be fit into workers with less memory. The memory overhead is measured in terms of the elements in the input and output matrices of each task. In the three jobs, the memory overhead can be saved by 22.2%, 47.6%, and 32.7%, respectively. With saved memory, the job completion time still remains similar to EP codes, except when the value of z is increased in the job J2 because we need to tolerate the same number of stragglers. In practice, however, the choices of z can be more flexible so that the job completion time can be better controlled. Finally, we find that even though the number of tasks required for decoding remains the same in each job for the two coding schemes, DEP codes can still save the decoding time by up to 42.0% in the job J1 and J3, thanks to its higher number of partitions which also makes the size of results smaller for decoding. In the job J2, however, its value of y is even smaller with DEP codes, making the size of results larger. Again, the choices of (x, y, z) can be more flexible in practice, as we do not need to have exactly the same values of n and s.

7. Conclusions

In the large-scale matrix multiplication, the input matrices are split into partitions so that they can be calculated on multiple resource-limited nodes. Although coding in distributed matrix multiplication can efficiently tolerate a high number of stragglers, most existing coding schemes are limited in only one or two dimensions. In this paper, we propose dual entangled polynomial (DEP) codes, a three-dimensional coding scheme that requires significantly fewer tasks to complete the job than entangled polynomial (EP) codes, the state-of-the-art three-dimensional codes for distributed matrix multiplication, by running one more matrix multiplication than EP codes in each task. Achieving lower or similar job completion time, DEP codes can also lower the memory requirement of tasks and the decoding overhead.

References

OpenBLAS: An Optimized BLAS library, 2017. URL https://www.openblas.net.

Apache Hadoop, 2018. URL http://hadoop.apache.org.

Baharav, T., Lee, K., Ocal, O., and Ramchandran, K. Straggler-proofing Massive-scale Distributed Matrix Multiplication with d-dimensional Product Codes. In *IEEE International Symposium on Information Theory (ISIT)*, pp. 1993–1997, 2018.

Dutta, S., Cadambe, V., and Grover, P. Short-Dot: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products. In *Advances in Neural Information Processing Systems*, pp. 2100–2108. 2016.

Gupta, V., Wang, S., Courtade, T., and Ramchandran, K. OverSketch: Approximate Matrix Multiplication for the Cloud. In *IEEE International Conference on Big Data*, 2018.

Huang, P., Guo, C., Zhou, L., Lorch, J. R., Dang, Y., Chintalapati, M., and Yao, R. Gray Failure: The Achilles' Heel of Cloud-Scale Systems. In *USENIX Conference on Hot Topics in Operating Systems (HotOS)*, 2017.

Kiani, S., Ferdinand, N., and Draper, S. C. Exploitation of Stragglers in Coded Computation. In *IEEE International Symposium on Information Theory (ISIT)*, 2018.

Lee, K., Suh, C., and Ramchandran, K. High-dimensional Coded Matrix Multiplication. In *IEEE International Symposium on Information Theory (ISIT)*, 2017.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Speeding Up Distributed Machine Learning Using Codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2018.

Mallick, A., Chaudhari, M., and Joshi, G. Rateless Codes for Near-Perfect Load Balancing in Distributed Matrix-Vector Multiplication. Technical report, 2018. URL https://arxiv.org/pdf/1804.10331.pdf.

Park, H., Lee, K., Sohn, J.-Y., Suh, C., and Moon, J. Hierarchical Coding for Distributed Computing. In *IEEE International Symposium on Information Theory (ISIT)*, 2018.

Reisizadeh, A., Prakash, S., Pedarsani, R., and Avestimehr, S. Coded Computation over Heterogeneous Clusters. In *IEEE International Symposium on Information Theory (ISIT)*, pp. 2408–2412, 2017.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient Coding: Avoiding Stragglers in Distributed Learning. In *International Conference on Machine Learning (ICML)*, pp. 3368–3376, 2017.

Wang, S., Liu, J., and Shroff, N. Coded Sparse Matrix Multiplication. In *International Conference on Machine Learning (ICML)*, 2018.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Straggler Mitigation in Distributed Matrix Multiplication: Fundamental Limits and Optimal Coding. In *IEEE International Symposium on Information Theory (ISIT)*, pp. 2022–2026. IEEE, 2018.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster Computing with Working Sets. In *USENIX Conference on Hot Topics in Cloud Computing (HotStorage)*, 2010.