Chebyshev Polynomial Codes: Task Entanglement-based Coding for Distributed Matrix Multiplication

Sangwoo Hong* [1] **Heecheol Yang*** [2] **Yeongseok Yoon** [1] **Taehyeon Cho** [1] **Jungwoo Lee** [1]

Abstract

Distributed computing has been a prominent solution to efficiently process massive datasets in parallel. However, the existence of stragglers is one of the major concerns that slows down the overall speed of distributed computing. To deal with this problem, we consider a distributed matrix multiplication scenario where a master assigns multiple tasks to each worker to exploit stragglers' computing ability (which is typically wasted in conventional distributed computing). We propose Chebyshev polynomial codes, which can achieve order-wise improvement in encoding complexity at the master and communication load in distributed matrix multiplication using task entanglement. The key idea of task entanglement is to reduce the number of encoded matrices for multiple tasks assigned to each worker by intertwining encoded matrices. We experimentally demonstrate that, in cloud environments, Chebyshev polynomial codes can provide significant reduction in overall processing time in distributed computing for matrix multiplication, which is a key computational component in modern deep learning.

1. Introduction

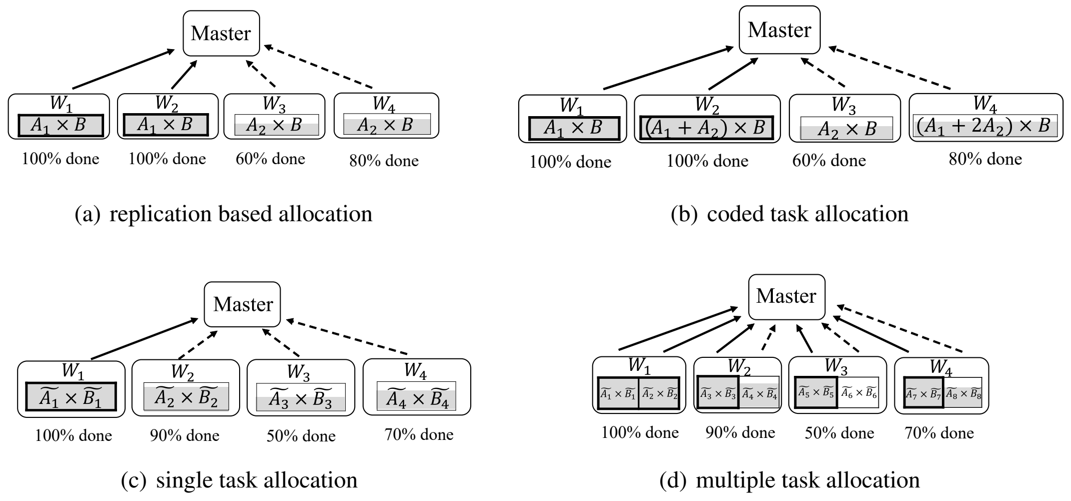
Matrix multiplication is one of the most basic building blocks in machine learning and deep learning. As the size of data required for deep learning grows, distributed computing is receiving significant attention due to its ability to handle the large dataset in a parallel manner. However, many studies (Dean & Barroso, 2013), (Huang et al., 2017), (Tandon et al., 2017) have reported that stragglers, which compute a given task much slower than other workers, deteriorate the computation capability of the system. Accordingly, the existence of stragglers has arose as an important issue in distributed computing since it is a major bottleneck in overall processing time.

To handle straggler issue in distributed computing, several methods have been proposed, which are depicted in Fig. 1. One of the conventional methods (Wang et al., 2014) is to replicate the computational tasks and allocate them to several workers. On the other hand, (Lee et al., 2018) were the first to suggest coded distributed computing, which can effectively utilize redundancy in task allocation by a coding theoretic approach, in order to alleviate the straggler problem. Fig. 1(a) and Fig. 1(b) show examples of distributed matrix multiplication on $C = A \times B$, where a master uses replication-based task allocation and coded task allocation schemes with four workers. In Fig. 1(a), two tasks are replicated and allocated to two workers under the assumption that A is divided into two matrices A_1 and A_2 of the same size, i.e., $A = [A_1; A_2]$. In this scheme, the master can tolerate one straggler per task to get the computation result C. However, if both workers responsible for the same task compute and return their results slowly as in Fig. 1(a), the master cannot decode the final computation result. On the other hand, in Fig. 1(b), the master utilizes a coded computation scheme which uses maximum distance separable (MDS) codes for task allocation. In this case, the master can decode the final computation result C from two of the fastest computation returns, thus it can tolerate any two stragglers among four workers.

For matrix multiplication, a coded computation scheme is expanded into various ways with different matrix partitioning parameters. Polynomial codes have been suggested in (Yu et al., 2017), which are designed to encode both input matrices by polynomial functions. In (Dutta et al., 2019), the authors have proposed a different matrix partitioning scheme for distributed matrix multiplication, achieving significant reduction in memory overhead at workers. Furthermore, in (Dutta et al., 2019), polydot codes that generalize the previously proposed schemes was introduced, which show trade-offs between recovery threshold, communication load, and computational load at workers. Finally, the authors

*Equal contribution [1]Communications and Machine Learning Lab., Department of Electrical and Computer Engineering, Seoul National University, Seoul, 08826, South Korea [2]Division of Computer Convergence, Chungnam National University, Daejeon, 34134, South Korea. Correspondence to: Jungwoo Lee <junglee@snu.ac.kr>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).



(a) replication based allocation

(b) coded task allocation

(c) single task allocation

(d) multiple task allocation

Figure 1. Comparison between task allocation schemes

in (Yu et al., 2020) have suggested entangled polynomial codes, which achieve better trade-off curve than polydot.

Although coded distributed computing do handle the straggler issue effectively, one of the major drawbacks is that the unfinished tasks at stragglers are completely ignored at the master. This results in inefficient utilization of computing resources in distributed computing system. In (Kiani et al., 2018), the authors have introduced a new coded computation scheme, in which the master assigns multiple small tasks to each worker, instead of a single large task. By assigning multiple small tasks to each worker and allowing them to return their computation results separately, the master can exploit the partial computation results from slow workers. We call this approach a **"straggler-exploiting coded computation"** in this paper. As shown in Fig. 1(c), when a single task is assigned to each worker, the master can only utilize computation results of worker W_1, while being unable to utilize unfinished computations of workers W_2, W_3 and W_4. On the other hand, in Fig. 1(d), the master allocates two small tasks with half of the size to each worker. In this case, the master can utilize two computation results of worker W_1 and one of the two computation results of workers W_2, W_3, and W_4 once they finish either of the two assigned tasks. Consequently, this approach can speed up overall processing time in distributed computing by fully leveraging computing resources. In addition, this approach has been extended into various distributed computing scenarios such as preserving data privacy and considering sparse matrix inputs (Kim et al., 2019), (Das & Ramamoorthy, 2020).

In this paper, we consider coded distributed computing in a straggler-exploiting scenario for matrix multiplication. Particularly, we propose a novel task encoding scheme to speed up overall processing time, referred to as *Chebyshev polynomial codes*. With Chebyshev polynomial codes, we perform task entanglement for matrix multiplication. By task entanglement, the results of L tasks allocated to each worker can be produced by multiplication between L_1 and L_2 encoded matrices, where $L_1 L_2 = L$. In a conventional straggler-exploiting distributed matrix multiplication, the master sends $2L$ encoded matrices to each worker to assign L tasks. In contrast, in Chebyshev polynomial codes, the master sends $L_1 + L_2$ encoded matrices to each worker. This results in **i) reducing encoding complexity** at the master, **ii) decreasing communication load** from the master to workers, and **iii) lowering memory usage** at workers to store encoded matrices, and **iv) having resiliency to straggler** as well. In the following, we will introduce how to perform task entanglement using Chebyshev polynomial codes, and how the master decodes the final product from the results returned from workers. We will also provide the performance of Chebyshev polynomial codes on the important metrics of distributed computing systems such as recovery threshold at the master, memory usage at workers, and overall process time. Finally, we will demonstrate the performance of Chebyshev polynomial codes by making comparison with existing schemes.

2. System Model

We consider a straggler-exploiting distributed computing scenario for matrix multiplication. In our setup, the master performs a matrix multiplication $C = AB$ with input matrices $A \in \mathbb{F}^{a \times b}$ and $B \in \mathbb{F}^{b \times c}$ in parallel, allocating WL tasks to W workers.

To be specific, the master generates encoded matrices $\tilde{A}_{i,j}$

and $\tilde{B}_{i,j}$ for $i \in [1:W]$, $j \in [1:L]$ using encoding functions $\mathbf{p_A}$, $\mathbf{p_B}$ on input matrices A and B. After generating encoded matrices, the master assigns L tasks by sending $\tilde{A}_{i,j}$ and $\tilde{B}_{i,j}$ for $j \in [1:L]$ to ith worker W_i for $i \in [1:W]$. Based on the received encoded matrices for L tasks, each worker W_i computes the assigned tasks $\tilde{C}_{i,j} = \tilde{A}_{i,j}\tilde{B}_{i,j}$, and sends the completed computation results to the master asynchronously. This implies that each worker sends completed results whenever each task is finished.[1]

After sending the encoded matrices, the master waits for the results of the tasks from workers, until being able to decode C from the results by using decoding functions. Once the master can decode C from the computation results, it stops receiving them and starts decoding the final product.

The main issue is how to design encoding functions and determine evaluation points, which are used for the generation of encoded matrices for matrix multiplication $C = AB$. Let us define important factors that mainly affect the performance in distributed computing as follows.

Encoding $\&$ **Decoding Cost**: Computational complexities to generate encoded matrices, and to decode the final product at the master.

Communication Load: Size of the transmitted data between the master and workers. It includes encoded matrices from the master to workers (task-allocation communication load)[2] and computation results returned from workers to the master (computation-return communication load).

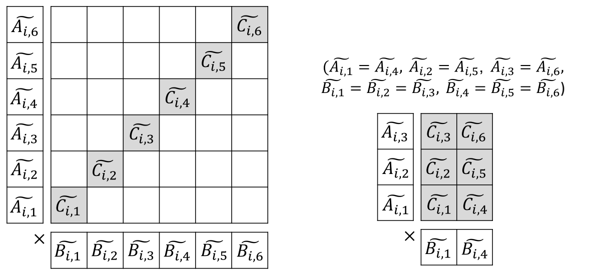
Computation Load: Computational complexity to calculate assigned tasks at each worker.

Recovery Threshold: Minimum number of computation results $\tilde{C}_{i,j}$ that the master requires to obtain the final result C in the worst-case scenario.

3. Task Entanglement

In this section, we provide task entanglement in a straggler-exploiting distributed matrix multiplication and suggest the required conditions. Mentioned earlier in Section 1, we reduce the number of encoded matrices sent to each worker from $2L$ to $L_1 + L_2$ by facilitating L_1 and L_2 encoded matrices to produce L computation results of the tasks when $L = L_1L_2$.

Fig. 2 shows the relationship between encoded matrices and 6 tasks assigned to each worker in conventional coding schemes and task entanglement-based coding scheme. In conventional coding schemes (Fig. 2(a)), 12 encoded matrices are sent to each worker to compute 6 tasks $\tilde{C}_{i,j} = \tilde{A}_{i,j}\tilde{B}_{i,j}$ for $j \in [1:6]$. Thus, the master needs to generate and send 12 encoded matrices to each worker. On the other hand, in the entanglement-based coding scheme (Fig. 2(b)), a total of 5 encoded matrices are generated and sent to W_i for 6 tasks ($L_1 = 2$ and $L_2 = 3$ in this case). This can be obtained by designing the matrix multiplications between each of $\{\tilde{A}_{i,1}, \tilde{A}_{i,3}, \tilde{A}_{i,5}\}$ and $\{\tilde{B}_{i,1}, \tilde{B}_{i,2}\}$ to produce required 6 computation results, $\tilde{C}_{i,j}$ for $j \in [1:6]$ as shown in Figure 2(b), which we call task entanglement.



$\widetilde{A_{i,6}}$
$\widetilde{A_{i,5}}$
$\widetilde{A_{i,4}}$
$\widetilde{A_{i,3}}$
$\widetilde{A_{i,2}}$
$\widetilde{A_{i,1}}$
$\widetilde{C_{i,1}}$ $\widetilde{C_{i,2}}$ $\widetilde{C_{i,3}}$ $\widetilde{C_{i,4}}$ $\widetilde{C_{i,5}}$ $\widetilde{C_{i,6}}$
$\times$
$\widetilde{B_{i,1}}$ $\widetilde{B_{i,2}}$ $\widetilde{B_{i,3}}$ $\widetilde{B_{i,4}}$ $\widetilde{B_{i,5}}$ $\widetilde{B_{i,6}}$
$(\widetilde{A_{i,1}} = \widetilde{A_{i,4}}, \widetilde{A_{i,2}} = \widetilde{A_{i,5}}, \widetilde{A_{i,3}} = \widetilde{A_{i,6}},$
$\widetilde{B_{i,1}} = \widetilde{B_{i,2}} = \widetilde{B_{i,3}}, \widetilde{B_{i,4}} = \widetilde{B_{i,5}} = \widetilde{B_{i,6}})$
$\widetilde{A_{i,3}}$ $\widetilde{C_{i,3}}$ $\widetilde{C_{i,6}}$
$\widetilde{A_{i,2}}$ $\widetilde{C_{i,2}}$ $\widetilde{C_{i,5}}$
$\widetilde{A_{i,1}}$ $\widetilde{C_{i,1}}$ $\widetilde{C_{i,4}}$
$\times$
$\widetilde{B_{i,1}}$ $\widetilde{B_{i,4}}$

(a) conventional coding scheme

(b) task entanglement-based coding scheme

Figure 2. Example of conventional coding scheme and task entanglement-based coding scheme

To accomplish task entanglement, we suggest the condition for encoded matrices that will be sent to each worker to assign L tasks.

Lemma 1. *In a straggler-exploiting distributed computing for matrix multiplication, one can reduce the number of encoded matrices sent to each worker by task entanglement, from* $2L$ *to* $L_1 + L_2$, *if encoded matrices satisfy the following conditions, where* $L = L_1L_2$, *for* $i \in [1:W]$,

$$\tilde{A}_{i,x} = \tilde{A}_{i,x+L_2\times(y-1)}, \quad \forall x \in [1:L_2], \forall y \in [1:L_1],$$
$$\tilde{B}_{i,y} = \tilde{B}_{i,x+L_2\times(y-1)}, \quad \forall x \in [1:L_2], \forall y \in [1:L_1].$$

To compute the given tasks $\{\tilde{C}_{i,j}\}_{j=1}^{L}$, each worker W_i requires encoded matrices $\{\tilde{A}_{i,j}\}_{j=1}^{L}$ and $\{\tilde{B}_{i,j}\}_{j=1}^{L}$. However, by the above condition, $\{\tilde{A}_{i,j}\}_{j=1}^{L}$ are compressed into L_2 matrices since each of a set of L_1 matrices has the same elements, and $\{\tilde{B}_{i,j}\}_{j=1}^{L}$ are compressed into L_1 matrices in the same way. Thus, the number of encoded matrices for each worker is equal to $L_1 + L_2$ and this implies that multiplication of L_1 and L_2 encoded matrices produce required results for L tasks $\{\tilde{C}_{i,j}\}_{j=1}^{L}$. Accordingly, the master encode and send only $L_1 + L_2$ encoded matrices to each worker, instead of $2L$ encoded matrices, and this results in reducing encoding complexity at the master and communication load for task assignment from the master to workers.

[1]We do not restrict the computation order of the L tasks at each worker.

[2]We note that this metric corresponds to the memory usage at each worker, since it can also represent the required memory size at each worker to store encoded matrices for the assigned tasks.

From now on, we will introduce our coding scheme, and show how task entanglement is realized. In our scheme, the master divides input matrices A and B into sub-matrices of equal size $A_w \in \mathbb{F}^{\frac{a}{m} \times b}$ for $w \in [1:m]$, and $B_z \in \mathbb{F}^{b \times \frac{c}{n}}$ for $z \in [1:n]$, which are given by

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix}, \quad B = \begin{bmatrix} B_1 & \cdots & B_n \end{bmatrix}. \tag{1}$$

Encoding functions $\mathbf{p_A}$ and $\mathbf{p_B}$ are constructed by using the divided sub-matrices of A and B as coefficients, which are given by

$$\mathbf{p_A}(x) = \sum_{w=1}^{m} A_w f(x)^{w-1}, \tag{2}$$
$$\mathbf{p_B}(x) = \sum_{z=1}^{n} B_z g(x)^{z-1},$$

where x represents the variable of polynomials of encoding functions $\mathbf{p_A}$ and $\mathbf{p_B}$, and $f(x)$ and $g(x)$ denote the polynomial basis for $\mathbf{p_A}$ and $\mathbf{p_B}$, respectively. Using these encoding functions, $\tilde{A}_{i,j} \in \mathbb{F}^{\frac{a}{m} \times b}$ and $\tilde{B}_{i,j} \in \mathbb{F}^{b \times \frac{c}{n}}$ are encoded from $\mathbf{p_A}(x)$ and $\mathbf{p_B}(x)$ at evaluation points $x = x_{i,j}$, which implicates $\tilde{A}_{i,j} = \mathbf{p_A}(x_{i,j})$ and $\tilde{B}_{i,j} = \mathbf{p_B}(x_{i,j})$. The tasks assigned to W_i are represented as $\tilde{C}_{i,j} = \tilde{A}_{i,j}\tilde{B}_{i,j} \in \mathbb{F}^{\frac{a}{m} \times \frac{c}{n}}$ for $j \in [1:L]$. These results represent the values of objective function $\mathbf{p_C}(x)$, which is given by

$$\mathbf{p_C}(x) = \mathbf{p_A}(x) \times \mathbf{p_B}(x), \tag{3}$$

at $x = x_{i,j}$. In this coding scheme, because $\mathbf{p_C}$ is also a polynomial function as $\mathbf{p_A}$ and $\mathbf{p_B}$, the master aims to decode the final product $C = AB$ by interpolating objective function $\mathbf{p_C}(x)$ from received results on evaluation points at $x = x_{i,j}$.

To achieve task entanglement, we need to carefully choose a pair of polynomial bases $f(x)$ and $g(x)$, that makes encoded matrices satisfying Lemma 1. We now provide the conditions on $f(x)$ and $g(x)$ for task entanglement in our scheme.

Lemma 2. *For encoding functions $\mathbf{p_A}(x)$ and $\mathbf{p_B}(x)$ in (2), to realize task entanglement, the polynomial bases $f(x)$ and $g(x)$ need to satisfy the following conditions.*

***Condition I**: There exist W sets of L points $x_{1,1}, x_{1,2}, ..., x_{W,L}$ that satisfy $f(x_{i,j+L_2(k-1)}) = \alpha_{i,j}$ and $g(x_{i,j+L_2(k-1)}) = \beta_{i,k}$ for all $i \in [1:W], j \in [1:L_2], k \in [1:L_1]$, where $L = L_1 L_2$.*

***Condition II**: Each of $f(x)$ and $g(x)$ has at least $L_1 - 1$ and $L_2 - 1$ distinct local extremum points, respectively.*

To satisfy Condition I in Lemma 2, $f(x)$ and $g(x)$ need to satisfy Condition II. In addition, to satisfy Condition II, the degrees of $f(x)$ and $g(x)$ need not to be less than L_1 and L_2, respectively.

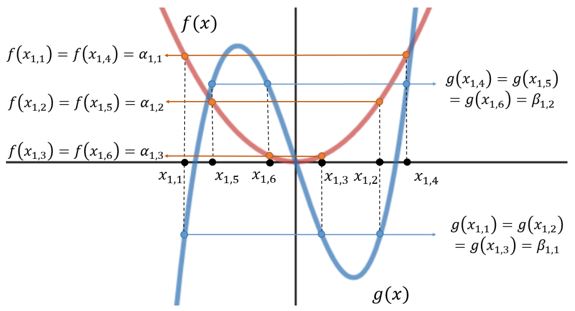


Figure 3. Choosing evaluation points in $f(x)$ and $g(x)$

Fig. 3 shows an example of a pair of polynomial bases $f(x)$ and $g(x)$ and a set of six evaluation points $\{x_{1,1}, x_{1,2}, x_{1,3}, x_{2,1}, x_{2,2}, x_{2,3}\}$. According to Fig. 3, the polynomial function $f(x)$ has three different values, and $g(x)$ has two different values at six evaluation points, satisfying Condition I in Lemma 2. Therefore, encoding functions $\mathbf{p_A}(x)$ and $\mathbf{p_B}(x)$ can generate encoded matrices required for task entanglement using $f(x)$, $g(x)$, and six evaluation points in the example.

Remark 1. *By task entanglement, the number of encoded matrices sent to each worker is reduced asymptotically from $\mathcal{O}(2L)$ to $\mathcal{O}(L_1 + L_2) \simeq \mathcal{O}(2\sqrt{L})$ when the values of L_1 and L_2 are close to $\sqrt{L}$, while satisfying $L = L_1 L_2$. This results in order-wise improvements, from $\mathcal{O}(L)$ to $\mathcal{O}(\sqrt{L})$, on encoding complexity at the master, task-allocation communication load, and memory usage at each worker.*

In the following, we show that one can always find the adequate pair of polynomial bases $f(x)$ and $g(x)$ satisfying the conditions in Lemma 2 for arbitrary matrix partitioning parameter m, n, and order of basis L_1, L_2. This can be done by utilizing the property of Chebyshev polynomials.

4. Chebyshev Polynomial Codes

We dedicate this section to propose Chebyshev polynomial codes, which enable task entanglement in a straggler-exploiting distributed matrix multiplication. To apply task entanglement in various distributed matrix multiplication scenarios, it should be applicable for different number of workers (W) and number of tasks assigned to each workers (L). Thus we need to find adequate polynomial bases and their evaluation points satisfying Lemma 2 for various W and L. Moreover, there should exist enough sets of evaluation points and polynomial bases for arbitrary combinations of L_1 and L_2 satisfying $L = L_1 L_2$.[3]

[3]The number of the sets of evaluation points should be more than the number of workers.

4.1. The selection of evaluation points

We show that Chebyshev polynomials are well-suited for task entanglement by suggesting an algorithm to find enough sets of evaluation points for arbitrary system parameters W and L.[4] To begin with, we need to re-index the notations for evaluation points and encoded matrices to avoid ambiguity. Using task entanglement, the master assigns $L_1 + L_2$ encoded matrices (not $2L$) to each worker. Therefore we change the notation for evaluation points as

$$x_{i,j,k} = x_{i,j+L_2(k-1)}, \quad \forall j \in [1 : L_2], \forall k \in [1 : L_1],$$

for $i \in [1 : W]$. Following this re-indexed notations, we also denote L tasks assigned to each worker and their encoded matrices as

$$\bar{C}_{i,j,k} = \bar{A}_{i,j} \times \bar{B}_{i,k}, \quad \forall j \in [1 : L_2], \forall k \in [1 : L_1],$$

for $i \in [1 : W]$.

We now demonstrate that required evaluation points $x_{i,j,k}$ can be found by using commutative polynomial as polynomial basis.

Lemma 3. *We can always find $x_{i,j,k}$ for $i \in [1 : W], j \in [1 : L_2],\ k \in [1 : L_1]$ satisfying Lemma 2 by using commutative polynomial for $f(x)$ and $g(x)$, each of which has L_1 and L_2 local extremum points, respectively.*

Proof. Let us choose an arbitrary value t_i between the greatest local minimum value and smallest local maximum value of $f(x)$ and $g(x)$. From $f(x) = t_i$, we can find the L_1 roots $\tilde{x}_{i,k}$ for $k \in [1 : L_1]$ as

$$f(\tilde{x}_{i,k}) = t_i. \quad \forall k \in [1 : L_1].$$

If the values of L_1 roots $\tilde{x}_{i,k}$ for $k \in [1 : L_1]$ also fall between the greatest local minimum value and smallest local maximum value of $g(x)$ as t_i, we can find L_2 roots $x_{i,j,k}$ for $j \in [1 : L_2]$ from $g(x) = \tilde{x}_{i,k}$ as

$$g(x_{i,j,k}) = \tilde{x}_{i,k}. \quad \forall j \in [1 : L_2], \forall k \in [1 : L_1]. \tag{4}$$

For $f(x)$ and $g(x)$ are commutative polynomial, the following equation holds.

$$\begin{aligned} f(g(x_{i,j,k})) &= g(f(x_{i,j,k})) \\ &= t_i, \quad \forall j \in [1 : L_2], \forall k \in [1 : L_1]. \end{aligned} \tag{5}$$

Since $g(x)$ has L_2 local extremum points and t_i are choosen between the greatest local minimum value and smallest local maximum value, the equation $g(x) = t_i$ will have L_2 different roots. Note that j is not included in the right-hand-side term in (4), thereby we can rearrange the order of $x_{i,j,k}$ on j. Thus, by denoting roots of $g(x) = t_i$ as $\bar{x}_{i,j}$ and rearranging the order of $x_{i,j,k}$ about k appropriately, we can express it as

$$f(x_{i,j,k}) = \bar{x}_{i,j}, \quad \forall j \in [1 : L_2], \forall k \in [1 : L_1].$$

Therefore, the commutative polynomial $f(x)$ and $g(x)$, and a set of evaluation points $x_{i,j,k}$ for $j \in [1 : L_2]$, $k \in [1 : L_1]$ satisfy Lemma 2. In addition, by choosing arbitrary values t_i for $i \in [1 : W]$, we can always find sets of evaluation points $x_{i,j,k}$ for $i \in [1 : W]$, $j \in [1 : L_2]$, and $k \in [1 : L_1]$. This completes the proof. □

Corollary 1. *A set of Chebyshev polynomials is an unique solution for Lemma 3 among commutative polynomial functions.*

Proof. It is proved in (Block & Thielman, 1951) that the entire sets of commutative polynomial functions are included in one of the sets

$$(I)\ \mathbf{P}_n(x) = \frac{(Ax + B)^n - B}{A}, \tag{6}$$

$$(II)\ \mathbf{T}_n(x) = A^{-1} cos(n \times cos^{-1}(Ax + B) - B), \tag{7}$$

$$(A \neq 0;\ n = 1, 2, 3, \ldots)$$

In set (I), $\frac{d\mathbf{P}_n(x)}{dx} = 0$ only if $x = -\frac{B}{A}$. Accordingly, the commutative polynomial functions in (I) have at most one local extremum points and they cannot satisfy Condition II in Lemma 2. Thus, we can conclude that the set (II), which corresponds to the set of Chebyshev polynomials, is an unique solution satisfying Lemma 3. This completes the proof. □

Given Lemma 3 and Corollary 1, we now provide an algorithm to find the sets of evaluation points for task entanglement when $f(x)$ and $g(x)$ are Chebyshev polynomials.

Algorithm 1 Selecting evaluation points

```
Input: Chebyshev polynomial $f(x)$ and $g(x)$ (deg $f = L_1$, deg $g = L_2$)
Output: A set of evaluation points $x_{i,j,k}$ for $i \in [1 : W]$, $j \in [1 : L_2]$, and $k \in [1 : L_1]$
$t_i = 0$ for $i \in [1 : W]$, $i = 1$            // Initialize
while $i \in W$ do
    Pick an arbitrary constant $t_i$ between $(-1, 1)$ that satisfies $t_a \neq t_b$ if $a \neq b$
    Calculate the roots of $f(x) = t_i$ and save them as $\tilde{x}_{i,k}$ for $k \in [1 : L_1]$
    Calculate the roots of $g(x) = \tilde{x}_{i,k}$ for $k \in [1 : L_1]$ and save them as $x_{i,j,k}$ for $j \in [1 : L_2]$ and $k \in [1 : L_1]$
    Rearrange $x_{i,j,k}$ about $j$ to satisfy $f(x_{i,j,k}) = \bar{x}_{i,j}$
    $i = i + 1$
end
```

[4]Chebyshev polynomials were first used for coded distributed computing in (Fahim & Cadambe, 2021) to guarantee numerical stability in decoding. In Chebyshev polynomial codes, Chebyshev polynomials are used to provide communication-efficient distributed computing.

In the algorithm 1, we use constant t_i in range $(-1 : 1)$ because local extremum values of Chebyshev polynomial are either -1 or 1.

4.2. Encoding by Chebyshev polynomial codes

In Section 3, we proposed the conditions for task entanglement in a distributed matrix multiplication. Furthermore, we prove that Chebyshev polynomials can always satisfy those conditions in 4.1. In this subsection, we will show how to design the encoding functions $\mathbf{p_A}(x)$ and $\mathbf{p_B}(x)$ using Chebyshev polynomials as polynomial bases $f(x)$ and $g(x)$.

Specifically, the encoding procedure by Chebyshev polynomial codes is as follows.

Step 1. Determine matrix partitioning parameters m and n for given input matrices A and B. Since the sizes of encoded matrices are $\frac{1}{m}$ and $\frac{1}{n}$ of the original input matrices A and B, respectively, the size of assigned tasks is a $\frac{1}{mn}$ of the original task $C = AB$. Thus, we determine the size of each task using m and n in this step.

Step 2. Determine L_1 and L_2, which are the degrees of $f(x)$ and $g(x)$, for given m, n, and L. One important consideration is that the degrees of $f(x)$ and $g(x)$ should be determined carefully to guarantee decodability at the master as

$$\deg f^i(x)g^j(x) = \deg f^k(x)g^l(x), \text{if } i = k \text{ or } j = l, \quad (8)$$
$$\forall i, k \in [1 : m-1], \forall j, l \in [1 : n-1].$$

The constraint on the degrees of $f(x)$ and $g(x)$ implies that the degrees of the multiplications of $A_w f(x)^{w-1}, w \in [1 : m]$ and $B_z g(x)^{z-1}, z \in [1 : n]$ in (2) should be unique. This is because the master needs to achieve the sub-blocks $A_w B_z, w \in [1 : m], z \in [1 : n]$ separately from the coefficients of $\mathbf{p_C}(x)$ in order to decode the final product C. Therefore, we should choose adequate values of L_1 and L_2 to satisfy this constraint. For example, if we set $m = n$, this constraint is satisfied by choosing a prime number for $L_2 = m$ and L_1 smaller than L_2.[5].

Step 3. Generate encoding functions $\mathbf{p_A}(x)$, $\mathbf{p_B}(x)$ as in (2), based on the divided sub-matrices of A, B in (1) and L_1, L_2 degree Chebyshev polynomials $f(x)$, $g(x)$.

Step 4. Find adequate evaluation points for generating encoded matrices on L tasks for each worker by Algorithm 1 in 4.1.

Step 5. Generate encoded matrices $\bar{A}_{i,j}$ and $\bar{B}_{i,k}$ for $j \in [1 : L_2]$ and $k \in [1 : L_1]$ for $i \in [1 : W]$. They can be generated by values of encoding functions $\mathbf{p_A}(x)$ and $\mathbf{p_B}(x)$ in Step 3 using evaluation points $x_{i,j,k}$ for $i \in [1 : W]$, $j \in [1 : L_2]$, and $k \in [1 : L_1]$ in Step 4.

[5]However, we can always find the adequate orders of $f(x)$ and $g(x)$ even if $m \neq n$ and L_2 is not a prime number. Detailed explanation is provided in Appendix A

4.3. Computing at workers

After receiving $\bar{A}_{i,j}$ and $\bar{B}_{i,k}$ for $j \in [1 : L_2]$ and $k \in [1 : L_1]$, each worker W_i starts to compute L tasks by calculating $\bar{C}_{i,j,k} = \bar{A}_{i,j} \times \bar{B}_{i,k}$. Whenever each task is finished, individual worker sends the completed computation results immediately to the master.

4.4. Decoding of Chebyshev polynomial codes

According to the divided sub-matrices of A and B in (1), the final product $C = AB$ can be represented as

$$C = \begin{bmatrix} A_1B_1 & \cdots & A_1B_n \\ \vdots & \ddots & \vdots \\ A_mB_1 & \cdots & A_mB_n \end{bmatrix}. \quad (9)$$

Thus, the master can get the final product C by obtaining sub-blocks of C, i.e., $A_1B_1 \ldots A_mB_n$. Let us denote the fastest mn computation results as $\bar{C}_t$ for $t \in [1 : mn]$ and denote the evaluation points of them as β_t for $t \in [1 : mn]$. The fastest mn computation results can be represented as (10).

Therefore, decoding at the master can be done by inversion of the coefficient matrix in (10). However, the existence of the inverse of the coefficient matrix is not guaranteed in general. If the coefficient matrix is singular, the master can decode by interpolation (Kedlaya & Umans, 2011) of $\mathbf{p_C}(x)$ and extracting A_iB_j using repeated division of $f(x)$ and $g(x)$ instead.

$$\begin{aligned} \mathbf{p_C}(x) &= \mathbf{p_A}(x) \times \mathbf{p_B}(x) \\ &= A_1B_1 + \ldots + A_mB_n f^{m-1}(x) g^{n-1}(x). \quad (11) \end{aligned}$$

$\mathbf{p_C}(x)$ is a $(L_1(m-1) + L_2(n-1))$-th order polynomial function, hence it can be interpolated from the values at the $L_1(m-1) + L_2(n-1) + 1$ evaluation points, which can be obtained from the computation results of the tasks from workers. Detailed decoding procedure is provided in the Appendix B.

5. Related Work

In (Yu et al., 2017), the authors have suggested polynomial codes, which use polynomial bases $f(x) = x$ and $g(x) = x^m$. Polynomial codes achieve optimal recovery threshold in aspect of computation load at workers.

On the other hand, Matdot has been introduced in (Dutta et al., 2019) to reduce memory usage at workers and communication load from the master to workers. It can be accomplished by column-wise and row-wise division of input

$$\begin{bmatrix} \bar{C}_1 \\ \vdots \\ \bar{C}_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{p_C}(\beta_1) \\ \vdots \\ \mathbf{p_C}(\beta_{mn}) \end{bmatrix} = \left(\begin{bmatrix} {\beta_1}^0 & f(\beta_1) & \dots & g(\beta_1) & \dots & f(\beta_1)^{m-1} g(\beta_1)^{n-1} \\ \vdots & & & \ddots & & \vdots \\ {\beta_{mn}}^0 & f(\beta_{mn}) & \dots & g(\beta_{mn}) & \dots & f(\beta_{mn})^{m-1} g(\beta_{mn})^{n-1} \end{bmatrix} \otimes I_{\frac{a}{m} \times \frac{a}{m}} \right) \times \begin{bmatrix} A_1 B_1 \\ \vdots \\ A_m B_n \end{bmatrix} \tag{10}$$

matrices A and B, respectively, which corresponds to the opposite division direction of polynomial codes. However, it require higher computation load than polynomial codes under the same recovery threshold.

In (Dutta et al., 2019), (Yu et al., 2020), and (Soto et al., 2019), general coding schemes dividing the input into both row-wise and column-wise have been proposed, which are termed as Polydot, entangled polynomial codes, and dual entangled polynomial codes, respectively. Dual entangled polynomial codes reduce the number of required task results at the master by executing two matrix multiplications for a single task. As a result, computation-return communication overhead from workers to the master is mitigated.

These codes have been originally considered in the scenario where only a single task is allocated to each worker. However, they can be easily extended to straggler-exploiting scenario by assigning multiple tasks to a single worker.

Straggler-exploiting scenario was first suggested in (Kiani et al., 2018). In order to allocate multiple tasks to each worker, the authors have proposed product codes, which apply different MDS codes to each input matrix. However, product codes require stricter decoding conditions than other codes (which use only one MDS code), since they use two MDS codes for encoding, thus their decodability can not be guaranteed by the fixed number of results, i.e., recovery threshold. To reduce the communication load for task allocation in straggler-exploiting scenario, (Hong et al., 2020) has suggested squeezed polynomial codes. Each worker computes L tasks by multiplying L encoded matrices of A and 1 encoded matrix of B. This can be regarded as a special case ($L_1 = 1$) of Chebyshev polynomial codes. However, because they perform task entanglement for only one input B, they cannot provide comparable gain on encoding complexity, communication load, and memory usage compared to Chebyshev polynomial codes. Specifically, their asymptotic gains on encoding complexity, communication load, and memory usage are $\mathcal{O}(L+1)$, whereas those of Chebyshev polynomial codes are $\mathcal{O}(2\sqrt{L})$.

6. Evaluation

In this section, we provide the performance of Chebyshev polynomial codes (denoted as CC in this section), and compare with following: i) polynomial codes (PC) (Yu et al., 2017), ii) entangled polynomial codes (EP) (Yu et al., 2020), iii) dual entangled polynomial codes (DEP) (Soto et al., 2019), and iv) Matdot (MD) (Dutta et al., 2019).

The distributed matrix multiplication proceeds in a cluster of Amazon EC2 cloud, while one t2.Xlarge node is used as the master and twelve t2.micro nodes are used as workers. Implementation of distributed computing system is done by using MPI4py (Dalcín et al., 2005). To simulate stragglers in the large-scale distributed computing scenario, we randomly pick stragglers among workers with straggler-probability 0.2, and run background thread at the stragglers which slows them down than other workers.

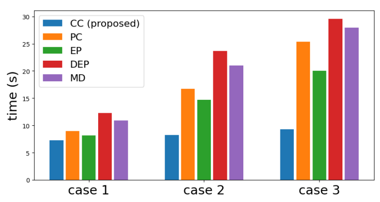
Three metrics are evaluated to estimate the performance of coding schemes: overall processing time, recovery threshold at the master, and memory usage at each worker. Furthermore, we measure processing times for every stage of the distributed computing procedure. To do so, we divide overall processing time into i) encoding time, ii) task-allocation time from the master to workers, iii) computation and task-return time from workers to the master, and iv) decoding time. Computation and task-return time denotes the elapsed time from the moment every worker receive encoded matrices to the moment the master receives required number of computation results to obtain the final product.

We run the experiments to multiply two input matrices A and B, each of which has the size of 2100 by 1800 and 1800 by 2100, respectively. Matrices are randomly generated as two Numpy matrices, and each experiment is repeated 20 times. We indicate the average results on the Figure 4 and 5.

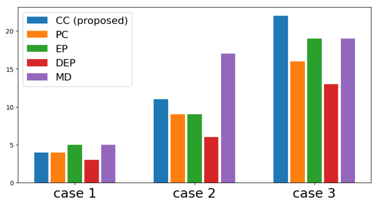
Table 1. Parameter setting for Case 1, 2, and 3

		CC	PC	EP	DEP	MD
Case 1. $L = 2$	m	2	2	2	2	1
$L_1 = 1$	n	2	2	1	1	1
$L_2 = 2$	p	1	1	2	2	3
Case 2. $L = 6$	m	3	3	2	2	1
$L_1 = 2$	n	3	3	2	2	1
$L_1 = 3$	p	1	1	2	2	9
Case 3. $L = 12$	m	4	4	2	2	1
$L_1 = 3$	n	4	4	2	2	1
$L_2 = 4$	p	1	1	4	4	10

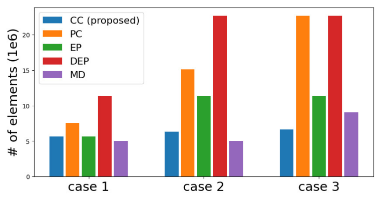
Experiments are performed for three cases, which have different parameter value L (number of the tasks allocated at each worker). Parameter setting of the coding schemes for three cases are is in Table 1. In the setting, p denotes the division parameter that decides column-wise division of A



(a) overall processing time



(b) recovery threshold



(c) memory usage at each worker

Figure 4. Comparison of overall processing time, recovery threshold at the master, and memory usage at each worker

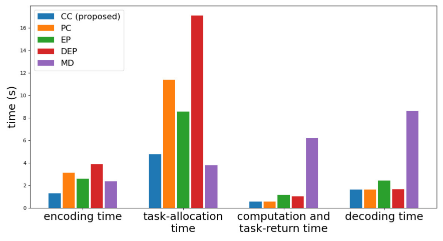


Figure 5. Processing times for every stage of distributed computing procedure in Case 2.

and row-wise division of B. Since CC and PC do not divide the matrix in this way, we use fixed value $p = 1$ for CC and PC. More experimental results including other types of matrices are provided in Appendix C.

In Fig. 4, we compare the performance in terms of (a) overall processing time, (b) recovery threshold, and (c) memory usage at each worker. We can see that CC achieves the smallest overall processing time in all three cases. It is remarkable that CC requires relatively higher recovery threshold compared to other coding schemes in several cases, but it does not lead to an increase in overall processing time. This is due to the fact that CC efficiently reduces encoding complexity and task-allocation communication load. Additionally, although the number of tasks assigned to each worker increases in Case 1, 2 and 3, CC achieves similar level of memory usage and overall processing time for all cases, while others show increasing memory usage and overall processing time. It can be achieved by task entanglement of CC, reducing the number of encoded matrices for each worker. As a result, CC achieves similar level of memory usage with MD, i.e., the memory efficient coding scheme, for Cases 1 and 2. Furthermore, CC even accomplishes lower memory usage for Case 3, while achieving significantly smaller overall processing time than MD in all cases.

We show processing times for every stage of distributed computing procedure of Case 2 in Fig. 5. The master generates $L_1 + L_2 = 5$ encoded matrices to assign 6 tasks to each worker in CC, while other coding schemes generates $2L = 12$ encoded matrices for 6 tasks. Accordingly, it achieves the smallest encoding time, and task-allocation time comparable to that of the smallest. In addition, CC and PC require the smallest computation and task-return time, and decoding time. This is because CC and PC use $p = 1$, while EP, DEP, and MD use $p > 1$, which increases computation load at workers and computation-return communication load from workers to the master under the same recovery threshold. Although MD achieves smaller task-allocation time than CC, CC attains smaller encoding, computation and task-return, and decoding time by the proposed task entanglement scheme. Therefore, it requires smaller overall processing time in a distributed computing. Moreover, DEP is the only scheme that achieves smaller decoding time than CC in this case. Decoding complexity depends on the size of computation results from workers to the master and recovery threshold. Since DEP achieves small size of computation results and requires the smallest recovery threshold, it achieves the smallest decoding time. However, DEP requires a larger number of encoded matrices for each task than CC, which results in larger encoding time and task-allocation time.

7. Conclusion

The existence of stragglers are considered as a major bottleneck delaying the overall processing time in distributed computing. As a solution, coded distributed computing has been proposed to handle the straggler issue. However, existing coding schemes have only considered a single task allocation to each worker, or have not efficiently leveraged the property of multi-task allocation. In this paper, we propose Chebyshev polynomial codes, which can achieve dramatic order-wise improvement (from L to $\sqrt{L}$) in encoding complexity and communication load by task entanglement. We propose the concept of task entanglement in a straggler-exploiting scenario, and show that it can be accomplished by using Chebyshev polynomial as a polynomial basis in encoding function. Consequently, Chebyshev polynomial codes are shown to provide significant reduction in overall processing time in a distributed computing for matrix multiplication, which is key computational operation in modern deep learning.

Acknowledgements

This work is in part supported by National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No.2021R1A2C2014504, 2021R1A4A1030898, 2020R1G1A1003759), Bio-Mimetic Robot Research Center Funded by Defense Acquisition Program Administration, Agency for Defense Development (UD190018ID), INMAC, and BK21-plus.

References

Block, H. D. and Thielman, H. P. Commutative polynomials. *The quarterly journal of mathematics*, 2(1):241–243, 1951.

Dalcín, L., Paz, R., and Storti, M. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

Das, A. B. and Ramamoorthy, A. Coded sparse matrix computation schemes that leverage partial stragglers. *CoRR*, abs/2012.06065, 2020. URL https://arxiv.org/abs/2012.06065.

Dean, J. and Barroso, L. A. The tail at scale. *Commun. ACM*, 56(2):74–80, 2013.

Dutta, S., Fahim, M., Haddadpour, F., Jeong, H., Cadambe, V., and Grover, P. On the optimal recovery threshold of coded matrix multiplication. *IEEE Transactions on Information Theory*, 66(1):278–301, 2019.

Fahim, M. and Cadambe, V. R. Numerically stable polynomially coded computing. *IEEE Transactions on Information Theory*, 67(5):2758–2785, 2021.

Hong, S., Yang, H., and Lee, J. Squeezed polynomial codes: Communication-efficient coded computation in straggler-exploiting distributed matrix multiplication. *IEEE Access*, 8:190516–190528, 2020.

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 *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017, Whistler, BC, Canada, May 8-10, 2017*, pp. 150–155. ACM, 2017.

Kedlaya, K. S. and Umans, C. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

Kiani, S., Ferdinand, N., and Draper, S. C. Exploitation of stragglers in coded computation. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1988–1992. IEEE, 2018.

Kim, M., Yang, H., and Lee, J. Private coded matrix multiplication. *IEEE Transactions on Information Forensics and Security*, 15:1434–1443, 2019.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D. S., and Ramchandran, K. Speeding up distributed machine learning using codes. *IEEE Trans. Inf. Theory*, 64(3): 1514–1529, 2018.

Soto, P., Li, J., and Fan, X. Dual entangled polynomial code: Three-dimensional coding for distributed matrix multiplication. In *International Conference on Machine Learning*, pp. 5937–5945. PMLR, 2019.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pp. 3368–3376. PMLR, 2017.

Wang, D., Joshi, G., and Wornell, G. Efficient task replication for fast response times in parallel computation. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, pp. 599–600, 2014.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4406–4416, 2017.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *IEEE Transactions on Information Theory*, 66(3):1920–1933, 2020.