Appendix

This supplementary material contains

A. Encoding constraint of Chebyshev Polynomial codes.

B. Decoding procedure for Chebyshev Polynomial codes.

C. Experimental results in various settings including different matrix sizes and comparison with squeezed polynomial codes.

A. Encoding constraint

As mentioned earlier in Section 6, Chebyshev polynomial codes have an encoding constraint. The degrees of mn terms of objective function $\mathbf{p_C}(x)$, each of which consists of one of A_iB_j for $i \in [1:m]$, $j \in [1:n]$ as coefficient, should not overlap with each other. If the degree of more than two independent terms are equal, their coefficients A_iB_j will be added together at the same degree in $\mathbf{p_C}(x)$. Thus, a master cannot extract the required information (A_iB_j for $i \in [1:m]$, $j \in [1:n]$) from the computation results on $\mathbf{p_C}(x)$. To satisfy this constraint, if the term $f^i(x)$ or $g^j(x)$ can cause overlapping problem, we need to skip them in the encoding function and use the next degree term, i.e., $f^{i+1}(x)$ or $g^{j+1}(x)$, for encoding A_{i+1} or B_{j+1}. The details are explained in the following example.

A.1. Motivating example

We provide a motivating example of encoding Chebyshev polynomial codes when the parameters do not satisfy the assumption in Section 4.2, $m = n = L_2$ and L_2 is a prime number, which were adopted for the sake of simplicity.

In this example, we determine the parameters as $m = 3$, $n = 2$, $L_1 = 2$, and $L_2 = 4$. For encoding the tasks with $\frac{1}{6}$ size of the original task, each of the two input matrices A and B are divided into three and two pieces as

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}, \qquad B = \begin{bmatrix} B_1 & B_2 \end{bmatrix}.$$

In this case, the final product C can be written as

$$C = \begin{bmatrix} A_1B_1 & A_1B_2 \\ A_2B_1 & A_2B_2 \\ A_3B_1 & A_3B_2 \end{bmatrix}.$$

Using the divided sub-matrices as coefficients, the encoding functions and objective function can be made as

$$\begin{aligned} \mathbf{p_A}(x) &= A_1 + A_2f(x) + A_3f^2(x), \\ \mathbf{p_B}(x) &= B_1 + B_2g(x), \\ \mathbf{p_C}(x) &= \mathbf{p_A}(x) \times \mathbf{p_B}(x), \\ &= A_1B_1 + A_2B_1f(x) + A_3B_1f^2(x) + A_1B_2g(x) + A_2B_2f(x)g(x) + A_3B_2f^2(x)g(x). \end{aligned}$$

where x denotes the variable of the polynomials and $f(x) = 2x^2 - 1$, which is a second order Chebyshev polynomial, and $g(x)$ is $8x^4 - 8x^2 + 1$, a fourth order Chebyshev polynomial. However, since $\deg f^2g^0 = \deg f^0g^1 = 4$, the degree of the terms for A_3B_1 and A_1B_2 are overlapped. Thus, the encoding functions above do not satisfy the encoding constraint. In this case, we can avoid overlapping of coefficients in $\mathbf{p_C}(x)$ by using $g^2(x)$ instead of $g(x)$. The revised encoding functions and objective function are determined as

$$\begin{aligned} \mathbf{p_A}(x) &= A_1 + A_2f(x) + A_3f^2(x), \\ \mathbf{p_B}(x) &= B_1 + B_2g^2(x), \\ \mathbf{p_C}(x) &= A_1B_1 + A_2B_1f(x) + A_3B_1f^2(x) + A_1B_2g^2(x) + A_2B_2f(x)g^2(x) + A_3B_2f^2(x)g^2(x). \end{aligned}$$

In $\mathbf{p_C}(x)$, the degrees of all required sub-blocks of C, i.e., A_iB_j for $i \in [1:3]$ and $j \in [1:2]$, are unique, thus a master can decode the final product C by using the matrix inversion of coefficient matrix in (10) or by interpolation.

As we have seen in the example, we can make adequate encoding functions for arbitrary combination of m, n, L_1, and L_2 by skipping the term in the encoding functions that can cause overlapping problem in the objective function. However, this method results in a larger degrees of encoding functions and objective function. Therefore, it may increase recovery threshold at the master, i.e., deg $\mathbf{p_C}(x)$ +1. To minimize this additional overhead, we can reduce the number of skipping terms by determining L_1 and L_2 as coprime integers.

B. Decoding procedure

Decoding of Chebyshev polynomial codes can be done by i) the inversion of coefficient matrix in (10) or ii) interpolation and repeated divisions. In this section, we explain decoding procedure by interpolation and repeated divisions.

For the sake of simplicity, we assume that the skipping method explained in Appendix A is not happened in the encoding functions. Nonetheless, decoding procedure introduced in this section can be easily extended to the case where skipping was not used.

The objective function $\mathbf{p_C}(x)$ is expressed as

$$\mathbf{p_C}(x) = A_1B_1 + \cdots + A_mB_1f^{m-1}(x) + A_1B_2g(x) + \cdots + A_{m-1}B_nf^{m-2}(x)g^{n-1}(x) + A_mB_nf^{m-1}(x)g^{n-1}(x).$$

Since $\mathbf{p_C}(x)$ is $L_1(m-1) + L_2(n-1)$th order polynomial, it can be interpolated from the $L_1(m-1) + L_2(n-1) + 1$ computation results. After interpolation, the master can get A_mB_n as the quotient by dividing $\mathbf{p_C}(x)$ with the highest degree term $f^{m-1}(x)g^{n-1}(x)$. Similarly, $A_{m-1}B_n$ can be achieved as quotient by dividing the remainder of former division with the highest degree term $f^{m-2}(x)g^{n-1}(x)$. All the required results A_iB_j for $i \in [1:m]$ $j \in [1:n]$ can be achieved by using repeated divisions by the highest degree term in the same way.

While decoding by inversion of coefficients matrix in (10) requires only mn computation results at a master, decoding by interpolation and repeated divisions requires more than $L_1(m-1) + L_2(n-1) + 1$ computation results. (More results are needed if skipping happens in encoding functions.) A master should use decoding by interpolation and repeated divisions when the coefficient matrix is singular, requiring more than mn computation results. However, the notable point is that the probability that coefficient matrix in (10) is singular is quite small.

Remark 1. *The probability that a master needs more than mn computation results to obtain the final product C is quite small, therefore the additional overhead in recovery threshold caused by Chebyshev polynomial codes is negligible.*

We experimentally demonstrate that by choosing the evaluation points in Algorithm 1 with uniform intervals, the probability of singular coefficient matrix in (10) is small enough.

For experiment, we generate 144 evaluation points using $W = 12, L_1 = 3, L_2 = 4$, and randomly choose 16 of them for coefficient matrix in (10). Since we do not impose any constraints on the order of calculation at each worker, we can say that random selection of evaluation points is reasonable.

After generating coefficient matrix, we use `np.linalg.cond` to get the condition number of coefficient matrix and use `1/sys.float_info.epsilon` as a criterion to check the singularity of a matrix and possible erroneous cases that can be caused by an ill-conditioned coefficient matrix. If `np.linalg.cond < 1/sys.float_info.epsilon`, we consider a coefficient matrix as singular. We repeat singularity check for $100,000$ times, and only 21 of them are came out to be singular. This result implies that the probability of singular coefficient matrix is quite small. Furthermore, the master can make $\binom{mn+1}{mn} = mn + 1$ possible coefficient matrices (size of mn by mn) from $mn + 1$ results and $\binom{mn+k}{mn}$ possible coefficient matrices from $mn + k$ results. Thus the master is highly likely to find invertible coefficient matrix and decode final output using mn and only few additional results, instead of all the $L_1(m-1) + L_2(n-1) + 1$ results.

Therefore, by generating a coefficients matrix with various combinations of received computation results and its matrix inversion, the master can decode the final product with significantly lower number than the degree of $\mathbf{p_C}(x)$. Thus, the additional overhead in recovery threshold is negligible.

C. Experimental results

We first provide experimental results of matrix multiplication using three types of matrices: square, tall, and fat matrices. The sizes of the input matrices are specified in Table 1. The other experimental settings (type of instances, number of

workers, matrix division parameters, and straggler probability) for three cases are the same as in Section 6. However, for the sake of clarity, we provide the detailed conditions in Table 2 again.

Table 1. Matrix size setting

matrix $A \in \mathbb{F}^{a\times b}$ $B \in \mathbb{F}^{b\times c}$	a	b	c
square matrix	1800	1800	1800
tall matrix	4500	540	4500
fat matrix	300	5400	300

Table 2. 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	4	4	1
$L_2 = 4$	p	1	1	2	2	10

To begin with, let us explain the relation between encoding, decoding complexities, and the size of input & output matrices. Encoding complexity depends on the size of input matrices and decoding complexity depends on the size of the output matrices (computation results). Therefore, the encoding complexity can be expressed as $\mathcal{O}(ab + bc)$, and the decoding complexity can be expressed as $\mathcal{O}(ac)$. This implies that decoding complexity is much larger than encoding complexity for tall matrix multiplication, and the encoding complexity is much larger than the decoding complexity for fat matrix multiplication.

Fig. 1(a), 1(b), and 1(c) show the processing times for each stage of square matrix multiplication in three cases, and Fig. 1(d) shows the overall processing time of three cases. We can observe that the experimental results for square matrix multiplication are similar to those of experiments in Section 6.

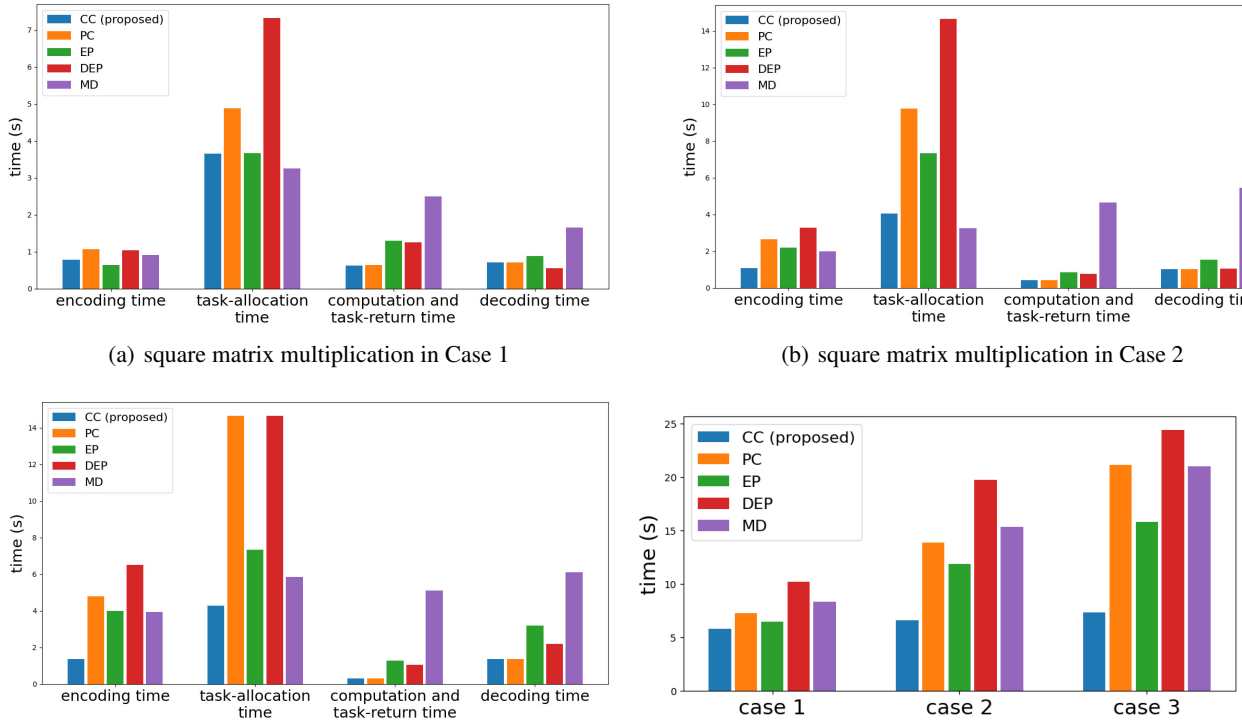
However, the processing time of tall matrix multiplication shows different tendency. The processing times for each stage and the overall processing time are shown in Fig. 2. As can be seen in Fig. 2(a), 2(b), 2(c), the decoding time of MD is much larger than other codes. Since MD has the highest decoding complexity than other codes and decoding is major bottleneck in tall matrix multiplication, MD requires the largest overall processing times in all the three cases, as denoted in Fig. 2(d). We note that CC achieves the lowest overall processing time because CC requires low decoding and encoding complexity.

The processing times for fat matrix multiplication is described in Fig. 3, and it shows different tendency compared to previous results. In fat matrix multiplication, the encoding complexity is much higher than the decoding complexity. Since encoding complexity is a major bottleneck for fat matrix multiplication, CC achieves the lowest overall processing time except Case 1, where MD requires the smallest processing time by different matrix partitioning.

Table 3. Parameter setting for Squeezed polynomial codes and Chebyshev polynomial codes

$A \in \mathbb{F}^{2100\times 1800}$, $B \in \mathbb{F}^{1800\times 2100}$		CC	SQ
Case 2.$L = 6$	m	3	3
$L_1 = 2, L_2 = 3$	n	3	3
Case 3. $L = 12$	m	4	4
$L_1 = 3, L_2 = 4$	n	4	4

We now provide comparison results with squeezed polynomial codes (SQ). As we have mentioned in Section 5, SQ can be regarded as a special case of CC, where $L_1 = 1$ and $L_2 = L$. In the following experiment, we show that CC can improve



(a) square matrix multiplication in Case 1

(b) square matrix multiplication in Case 2

(c) square matrix multiplication in Case 3

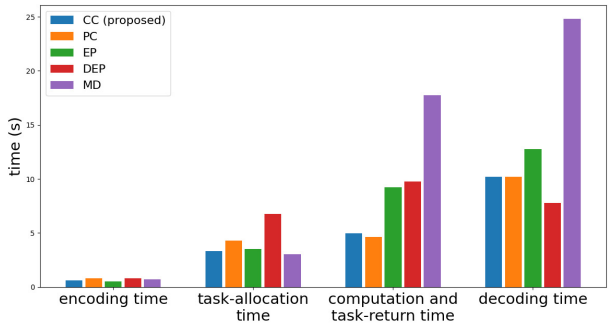
(d) overall processing time for square matrix multiplication

Figure 1. Comparison of processing time for square matrix multiplication

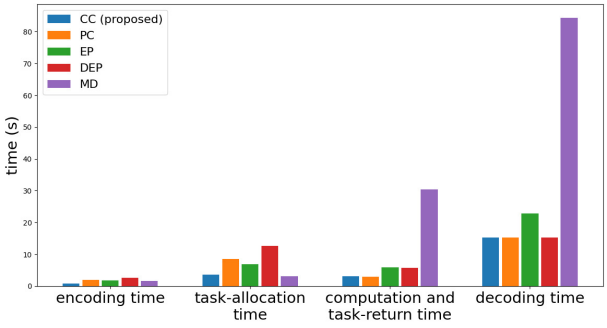
the performance for overall processing time than SQ by comparing them in two different scenarios. One t2.Xlarge node is used as a master and 8 t2.micro nodes are used as workers. We perform the experiments for Case 2 and 3 in Section 6. Since CC and SQ can be regarded as same codes in Case 1, where $L_1 = 1, L_2 = L$, we do not perform the experiment for Case 1 here. The detailed settings including parameter values and matrix size are given in Table 3, and we use straggler probability 0.3. Each experiment is repeated 20 times and the average results are indicated on the figure.

As we can see in Fig. 4(a) and 4(b), the decoding time and the computation and task-return time of CC and SQ are almost the same since matrix division parameters are the same. However, because CC improves order-wise improvement in encoding complexity and task-allocation communication load, it shows a substantial gain in encoding time and task-allocation time.

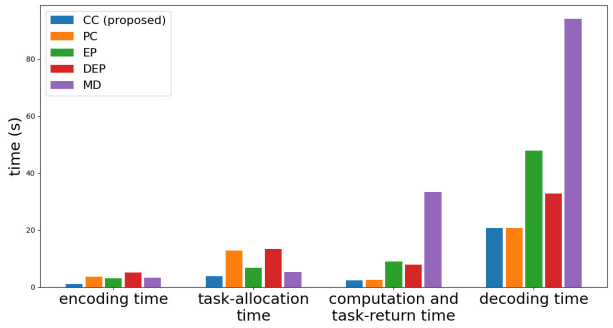
It should be noted that CC shows superior performance in most of the cases in overall processing time for all the matrix shapes (square, fat, and tall) in all three cases with different parameter settings. This results clearly demonstrate the superiority of the proposed codes.



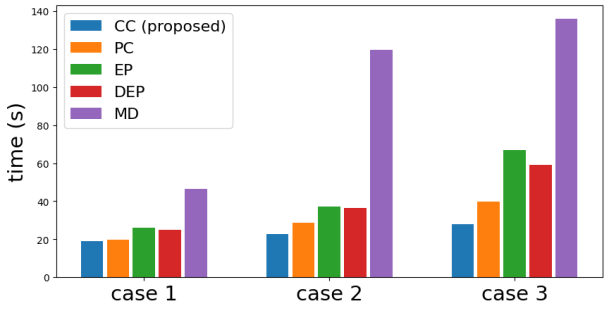
(a) tall matrix multiplication in Case 1



(b) tall matrix multiplication in Case 2

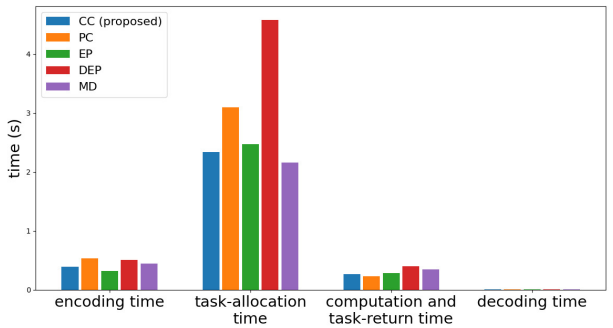


(c) tall matrix multiplication in Case 3

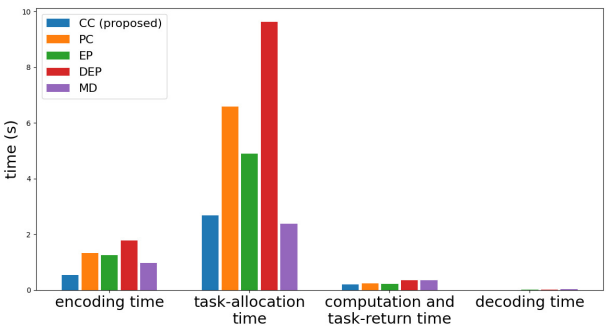


(d) overall processing time for tall matrix multiplication

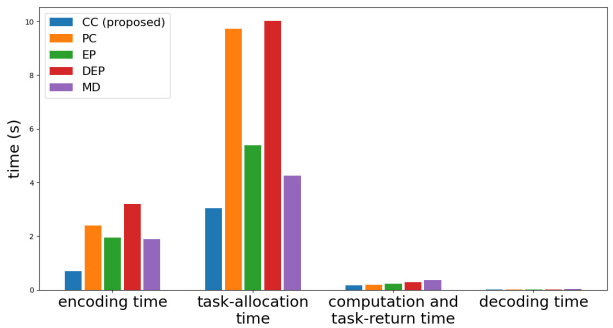
Figure 2. Comparison of processing time for tall matrix multiplication



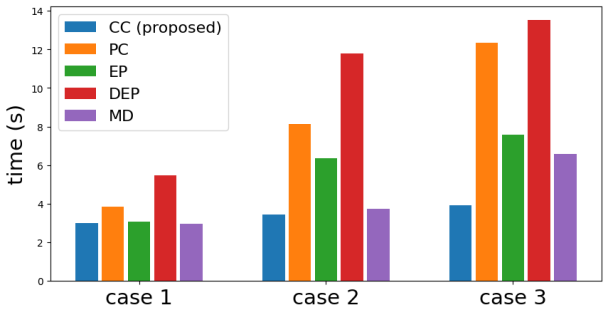
(a) fat matrix multiplication in Case 1



(b) fat matrix multiplication in Case 2

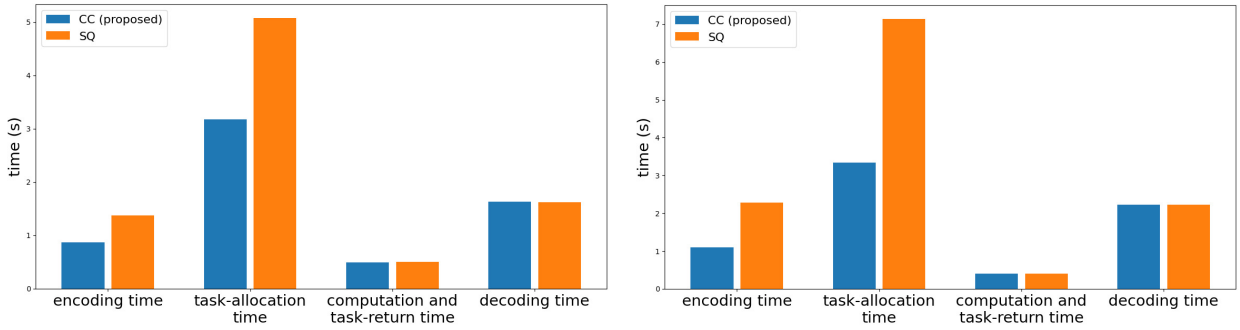


(c) fat matrix multiplication in Case 3



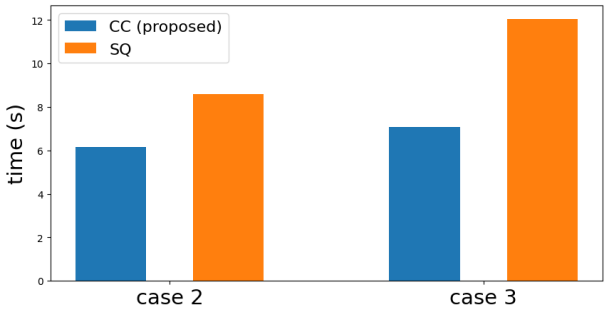
(d) overall processing time for fat matrix multiplication

Figure 3. Comparison of processing time for fat matrix multiplication



(a) comparison with squeezed polynomial codes in Case 2

(b) comparison with squeezed polynomial codes in Case 3



(c) comparison of overall processing time

Figure 4. Comparison of processing time between Chebyshev polynomial codes and squeezed polynomial codes