2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

Masterless Coded Computing: A Fully-Distributed Coded FFT Algorithm

Haewon Jeong, Tze Meng Low, and Pulkit Grover
haewon@cmu.edu, lowt@andrew.cmu.edu, pulkit@cmu.edu

Abstract— **We propose a coded computing strategy for the Fast Fourier Transform (FFT) algorithm in a fully distributed setting, which does not have a powerful master node orchestrating worker nodes. The fully distributed setting requires a large amount of data movements between nodes, and this communication is often the bottleneck in parallel computing. We identify communication cost of each step of the coded FFT algorithm using the α-β model, which is commonly used in high-performance computing literature to estimate communication latency. We show that by using a (P, K) systematic MDS code, the communication overhead of coding is negligible in comparison to the communication costs inherent in the uncoded FFT implementation if $P - K = o(\log K)$.**

I. Introduction

The Fast Fourier Transform (FFT) algorithm is the backbone of scientific computations which is used in a number of applications, such as solving Poisson's equations for particle simulations [1], [2]. For solving differential equations with high accuracy, FFTs of a very large size are often computed. To speed up scientific simulations, a large-scale FFT is often implemented over massively parallel processors [3]–[5]. However, as we head into the era of *exascale* computing, with more than 100,000 processing nodes, we expect to see more frequent faults in the course of computation [6]. Traditionally, a "checkpointing" technique is used to mitigate faults, where the state of the computation is stored offline at regular intervals and the computation restarts at the most recent checkpoint when an error is detected. Now, the error rate of large parallel systems is projected to reach a point where present checkpoint/restart methods will no longer be viable [7], [8]. We thus need a more efficient way to mitigate faults in large-scale numerical algorithms by exploiting algorithm-specific features.

In this work, we propose a *"coded computing"* approach to the large-scale parallel FFT algorithm. Coded computing, which combines computing and error correcting codes (ECCs) to mitigate unreliable nodes in distributed computing, has gained traction in recent years [9]–[15]. However, our work differs from the existing literature in coded computing as we consider *a fully distributed setup* which does not have a single master node. A stream of works in coded computing have assumed a "master-worker setup" where the system has a powerful master node that distributes data to and aggregates the result from worker nodes. This is not a reasonable model for practitioners for several reasons. First, a master node itself can fail, at which point, computation results cannot be guaranteed. Secondly, this assumes that a master node has a very large memory which can store the entire data. Let us denote the number of workers as P, then the master node must have P times more memory than the workers, which is not realistic as P grows large. Lastly, like in the FFT algorithm where a master node must intervene in the middle of the computation, a master node can be the bottleneck of the computation.

With the absence of a master node, all the nodes have to communicate with each other to gather the required data. It is well known in distributed computing that communication is often the bottleneck, and not computation. This is because communication bandwidth is not growing as fast as flop rates of processors [16]. A recent work [17] has explored coded computing approach to FFT under the master-worker setup. If we plainly apply their technique to the fully distributed setting, the communication overhead of encoding/decoding might dominate and coded computing approach could end up much slower than the uncoded FFT algorithm. We thus ask a question on whether the coding approach using maximum distance separable (MDS) codes [17] can be a feasible option in the fully distributed setting. In this work, we provide an efficient communication algorithm for encoding and decoding, called *"multi-reduce"*, and show that as long as the number of redundant nodes is $o(\log K)$ where K is the number of systematic nodes, the communication overhead of "coding" is amortized. We do not include computation cost analysis in this work as it was thoroughly analyzed in [17].

To analyze the communication cost, we use α-β model which is commonly used in high-performance computing literature. In this model, the time spent to communicate a message between two processors is approximated by a linear function as follows:

$$\alpha + \beta \times (\text{message length}).$$

This captures the latency of setting up a link (α) and the communication bandwidth (β). In distributed storage literature, communication cost was considered either in the network bandwidth (regenerating codes) [18]–[20] or the number of node accesses (locally repairable codes) [21]–[24]. The α-β model captures more fine details of communication latency because it not only accounts for the number of nodes or the number of bits, but also accounts for the number of communication rounds. For instance, let us think about two different communication scenarios where P nodes are participating in the communication. If the communication consists of a set of disjoint pairwise communications that can be done in parallel, it only requires one round of communication, and hence communication latency will be low. On the other hand, if all the P nodes have to send a message to all the other nodes, there must be more rounds of communication since not every node can talk to each other simultaneously. This difference due to different communication pattern can be captured in the α-β model. While this is still a simple approximation of communication cost, this captures more realistic constraints especially when a system has comparable α and β.

Using ECCs for fault-tolerant FFT computation has been extensively studied in algorithm-based fault-tolerance (ABFT) literature. The ABFT philosophy, first suggested by Huang and Abraham in 1984 [25], is adding checksums in the beginning of the computation and comparing it against the checksum of outputs at the end of the computation in order to detect errors that happened during the computation. In essence, it uses the simplest error-detection code for computations. The ABFT technique was first applied to FFT by Jou and Abraham in 1986 [26] and since then substantial research has been done to improve on hardware/time overhead by designing new weighted checksums that can be efficiently implemented on an FFT circuit [27]–[30]. None of these works, however, have considered communication overhead of their algorithms, because most of them were focused on circuit-level fault tolerance where communication is very cheap.

II. System Model and Preliminaries

A. Distributed Computing Model

We will use *"processor"* and *"node"* interchangeably in this paper. We assume that we have a total of P processors that have the same computational capabilities and memory. Among P processors, K of them are *"systematic processors"* which store the original data and the remaining $P - K$ processors are *"parity processors"* which store encoded parity symbols. We assume a massively parallel setup where K is very big, but K does not grow faster than $\Theta(\log N/\log\log N)$[1].

We assume a *fully-distributed* setting where no central processor is present during the computation and the processors do not have any shared memory. Data located at different processors can be shared only through explicit communication between two processors.

Using P processors, we want to compute N-point FFT:

$$\boldsymbol{Z} = F_N \boldsymbol{x} \tag{1}$$

where $\boldsymbol{x}$ is a length-N input data vector, F_N is an N-by-N DFT matrix (ω_N: the N-th root of unity) represented as

$$F_N = \begin{bmatrix} \omega_N^0 & \omega_N^0 & \cdots & \omega_N^0 \\ \omega_N^0 & \omega_N^1 & \cdots & \omega_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^0 & \omega_N^{N-1} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix}, \tag{2}$$

and $\boldsymbol{Z}$ is a length-N vector of the Fourier transform of $\boldsymbol{x}$. We assume that N is very large, so that the data cannot be stored in one processor. In the beginning, each processor has a segment of consecutive values of the input vector $\boldsymbol{x}$, e.g., Processor 1 has $\begin{bmatrix} x_1 & x_2 & \cdots & x_{N/K} \end{bmatrix}^T$.

[1]

B. Fault Model

In this work, we consider an erasure model for faults where we lose the entire output if a node fails. A node failure can happen when a node dies after a random fault, or it is a straggler that is unable to finish its computation.

C. Communication Model

We assume a fully-connected network of processors where any processor can send and receive data from every other processor directly. We also assume that a processor has one duplex port, which means that at a given time, a processor can send data to one processor and receive data from another processor simultaneously.[2] For example, Processor 1 can send data to Processor 2 and receive data from Processor 3 at the same time, but it cannot send data to Processor 2 and 3 at the same time.

We use the α-β model to estimate the point-to-point communication cost. In the α-β model, the time to send or receive a message of s bytes is :

$$T = \alpha + s \cdot \beta \tag{3}$$

Here, α is startup time to establish a connection between two nodes, and β is the bandwidth cost required to transfer one symbol. For an algorithm that requires multiple rounds of message exchanges, total communication time can be written as follows:

$$T = C_1\alpha + C_2\beta, \tag{4}$$

where C_1 is the number of communication rounds, C_2 is the number of symbols communicated in a sequence. To be more precise, if we denote b_i as the maximum number of symbols communicated between two nodes at the i-th round, C_2 can be written as:

$$C_2 = \sum_{i=1}^{C_1} b_i. \tag{5}$$

This is because the next round does not start until the previous round is completed, and the bandwidth latency for each round is dominated by the largest message. Symbols can have different units, such as bits or bytes, but in this work we do not specify any units.

D. Distributed FFT algorithm

We want to explain the *"transpose"* algorithm that is commonly used in high-performance FFT libraries [3]. It uses the Cooley-Tukey technique to break down N-point FFT into smaller FFTs of size N_1 and N_2 where $N = N_1N_2$.

[2]We believe that this can be easily extended to k-port model where each node has k duplex ports.

Now, (1) can be rewritten as

$$\begin{aligned} Z_k &= \sum_{n=0}^{N-1} \omega_N^{nk} x_n \\ &= \sum_{n_1=0}^{N_1-1} \omega_{N_1}^{n_1 k_1} t_{n_1,k_2} \sum_{n_2=0}^{N_2-1} \omega_{N_2}^{n_2 k_2} x_{n_2 N_1 + n_1} \end{aligned}$$

where $k = k_1 N_2 + k_2$, $k_1 = 0, \cdots N_1 - 1$, and $k_2 = 0, \cdots, N_2 - 1$. The terms t_{n_1,k_2}'s are called twiddle factor which are equal to $\omega_N^{k_2 n_1}$.

We can now compute N-point FFTs in two steps. In the first step, each processor is assigned to compute N_1/K FFTs of length N_2. Then the processors transpose the data (requiring communication) and compute N_2/K FFTs of size N_1 in the second step. Between the first and the second step, we have to multiply twiddle factors. This complicates our coding approach since multiplying twiddle factors is an element-wise multiplication of two matrices (Hadamard product), which does not commute with matrix-matrix multiplication (See Remark 1). We now explain the algorithm in detail:

Algorithm 1. ***Uncoded Distributed FFT Algorithm (Transpose Algorithm)***

1) Rearrange the input data $\boldsymbol{x}$ into X:

$$\begin{aligned} X &= \begin{bmatrix} x_1 & x_{N_1+1} & \cdots & x_{(N_2-1)N_1+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_1} & x_{2N_1} & \cdots & x_{N_1 N_2} \end{bmatrix} \\ &= \begin{bmatrix} X_1^{(\text{row})} \\ \vdots \\ X_K^{(\text{row})} \end{bmatrix} = \begin{bmatrix} X_1^{(\text{col})} & \cdots & X_K^{(\text{col})} \end{bmatrix}. \end{aligned}$$

We use $X_i^{(\text{row})}$'s ($X_i^{(\text{col})}$'s) to denote equal-sized submatrices of X divided horizontally (vertically). From our system assumption, in the beginning, the i-th processor has $X_i^{(\text{col})}$ [3]. To begin the distributed FFT computation, we transpose the data distributed over K processors so that the i-th processor can now have $X_i^{(\text{row})}$.

2) Compute N_1/K row-wise FFTs of size N_2 at each processor.

$$Y_i^{(\text{row})} = X_i^{(\text{row})} F_{N_2}$$

3) Transpose the data so that the i-th processor has $Y_i^{(\text{col})}$.

$$Y = \begin{bmatrix} Y_1^{(\text{row})} \\ \vdots \\ Y_K^{(\text{row})} \end{bmatrix} = \begin{bmatrix} Y_1^{(\text{col})} & \cdots & Y_K^{(\text{col})} \end{bmatrix}$$

4) Multiply twiddle factors at each processor.

$$Y_i^{(\text{col})} = T_{N,i}^{(\text{col})} \circ Y_i^{(\text{col})}$$

where $\circ$ represents Hadamard product and T_N is a matrix of twiddle factors

$$\begin{aligned} T_N &= \begin{bmatrix} \omega_N^0 & \omega_N^0 & \cdots & \omega_N^0 \\ \omega_N^0 & \omega_N^1 & \cdots & \omega_N^{N_2-1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^0 & \omega_N^{N_1-1} & \cdots & \omega_N^{(N_1-1)(N_2-1)} \end{bmatrix} \\ &= \begin{bmatrix} T_{N,1}^{(\text{col})} & \cdots & T_{N,K}^{(\text{col})} \end{bmatrix}. \end{aligned}$$

5) Compute N_2/K column-wise FFTs of size N_1 at each processor.

$$Z_i^{(\text{col})} = F_{N_1} Y_i^{(\text{col})}. \quad (6)$$

[3]This assumption is coming from that it is more natural for a processor to store contiguous data without the knowledge that the next computation is going to be FFT. If we assume that processors have row-wise data in the beginning, we can avoid the first transpose step. This does not change the result in Theorem 2 in scaling sense.

III. Coded Distributed FFT

We will now explain our coding strategy for the distributed FFT algorithm. The uncoded distributed algorithm described in Algorithm 1 has transpose step in the middle which requires all the nodes in the system to exchange data with all the other nodes. If there is any failed node before the transpose step, the computation will fail at the transpose step. Hence, simply adding fault tolerance which recovers faults at the end of the algorithm is not adequate for the distributed FFT algorithm. We need to apply fault resilience technique twice: once right before the transpose step, and once when the entire computation is complete. This requires *distributed encoding and decoding in the middle of the computation* which poses unique challenges for coded FFT algorithm.

In our coding strategy, we utilize $(P - K)$ redundant processors to encode the first and the second FFT steps separately. In the first step, processors perform FFT on the row-wise data $X_i^{(row)}$'s. In order to protect from the lost output at a failed node, we have to encode parity symbols across columns (column-wise encoding). By doing this, at the end of the first step, any successful K processors can recover the output and proceed to the next step. In the second step, each processor computes FFT on the column-wise data, $Y_i^{(col)}$'s, so we encode row-wise parity symbols (row-wise encoding). Our coded computing algorithm is described below (*: additional steps that are not present in the uncoded algorithm).

Algorithm 2. ***Coded Distributed FFT Algorithm***

1) * Encode column-wise parity symbols at each processor.

$$\tilde{X} = G_1^T X = \begin{bmatrix} \tilde{X}_1^{(row)} \\ \vdots \\ \tilde{X}_P^{(row)} \end{bmatrix} \quad (7)$$

G_1 is an N_1-by-N_1' encoding matrix for where $N_1' = \frac{P}{K} N_1$:

$$G_1 = \begin{bmatrix} I_{N_1} & \mathcal{P}_1 \end{bmatrix} \quad (8)$$

2) Rearrange the encoded data. Now the i-th processor has $\tilde{X}_i^{(row)}$.
3) Compute N_1/K row-wise FFTs of size N_2 at each processor.

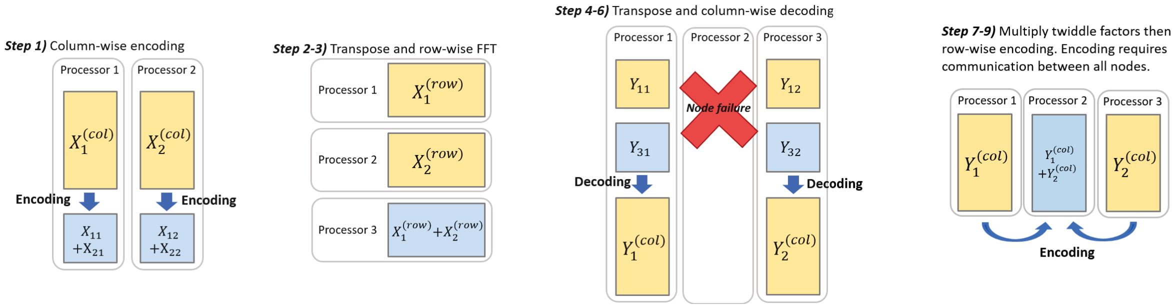

Fig. 1: This diagram summarizes encoding and decoding steps in Algorithm 2 with an example of $P = 3, K = 2$.

4) Wait for the first successful K processors and transpose the output within the successful K processors.
5) * If needed, decode to retrieve the uncoded output at each processor.
6) Multiply twiddle factors.
7) * Encode row-wise parity symbols and send them to the remaining $P - K$ processors.

$$\tilde{Y} = YG_2 = \begin{bmatrix} \tilde{Y}_1^{(col)} & \cdots & \tilde{Y}_P^{(col)} \end{bmatrix} \tag{9}$$

G_2 is an N_2-by-N_2' encoding matrix where $N_2' = \frac{P}{K}N_2$:

$$G_2 = \begin{bmatrix} I_{N_2} & \mathcal{P}_2 \end{bmatrix} \tag{10}$$

8) Compute N_2/K row-wise FFTs of size N_1 at each processor.
9) * Wait for the first successful K processors and halt the remaining $P - K$ processors. Decode if needed.

For both encoding steps in Step 1 and Step 7, we use *a (P, K) systematic MDS code*. In the following theorem, we show that using the proposed coded distributed FFT algorithm, any K successful processors are enough to recover the computed outputs at Step 5 and Step 9 [4].

Theorem 1. *In Algorithm 2 where we compute distributed FFT of size N using P processors each of which can store and process $\frac{1}{K}$ fraction of the input ($P > K$), any successful K processors can recover Y and Z at Step 5 and 9, respectively.*

Proof. Let us first prove that we can recover Y with any K successful processors at Step 5 and the similar argument holds for recovering Z at step 9.

At Step 4, we will have the result from K successful workers. Let us denote the indices of the successful K workers as $\{i_1, \cdots, i_K\}$. Then the output from the successful workers is:

$$Y_{\text{suc}} = \begin{bmatrix} \tilde{X}_{i_1}^{(\text{row})} F_{N_2} \\ \tilde{X}_{i_2}^{(\text{row})} F_{N_2} \\ \vdots \\ \tilde{X}_{i_K}^{(\text{row})} F_{N_2} \end{bmatrix} = \begin{bmatrix} Y_{\text{suc},i_1}^{(\text{col})} & \cdots Y_{\text{suc},i_K}^{(\text{col})} \end{bmatrix}. \tag{11}$$

[4]Note that we do not have any fault recovery for twiddle multiplication step. However, computational complexity of twiddle factor multiplication is $O(N)$ compared to that of $O(N \log N)$. Hence, it is less probable to have faults during twiddle factor multiplication step

After transposing at Step 5, processors $i_1, \cdots, i_K$ will have column-wise output $Y_{\text{suc},i_1}^{(\text{col})}, \cdots Y_{\text{suc},i_K}^{(\text{col})}$. $Y_{\text{suc},i}^{(\text{col})}$ can be written as:

$$Y_{\text{suc},i}^{(\text{col})} = G_{1,suc}^T X F_{N_2,i}^{(\text{col})} = G_{1,suc}^T Y_i^{(\text{col})} \tag{12}$$

where $G_{1,suc}^T$ is a submatrix of G_1^T which only has rows from successful nodes and hence has the size N_1-by-N_1.

As we assume the erasure model where we lose the entire data from a failed node, we only code across nodes, not within a node. Hence, our encoding matrix G_1 has the following structure:

$$G_1 = \mathcal{G}_1 \otimes I_{N_1/K} \tag{13}$$

where $\mathcal{G}_1$ is the encoding matrix for a systematic (P, K)-MDS code which has size K-by-P.

Now, $G_{1,\text{suc}}$ can be rewritten as:

$$G_{1,\text{suc}}^T = \mathcal{G}_{1,\text{suc}}^T \otimes I_{N_1/K} \tag{14}$$

where $\mathcal{G}_{1,\text{suc}}$ is a submatrix of $\mathcal{G}$ that only has K columns from the K successful nodes, i.e., i_1-th to i_K-th columns of $\mathcal{G}$. Because $\mathcal{G}_1$ is a (P, K) MDS code, $\mathcal{G}_{1,\text{suc}}$ always has a full rank. As $\text{rank}(A \otimes B) = \text{rank}(A) \cdot \text{rank}(B)$ for any matrices A and B, $\text{rank}(G_{1,\text{suc}}) = N_1$. Hence, we can recover $Y_i^{(\text{col})}$ at every successful node at Step 5. Similar argument applies to recovering Z at Step 9. □

IV. Communication Cost of Coded FFT Algorithm

In this section, we prove our main theorem which states that as long as the number of parity processors is $o(\log K)$, communication overhead of encoding and decoding can be amortized:

Theorem 2. *In our proposed coded FFT algorithm, if $P - K = o(\log_2 K)$, communication overhead of coding is negligible compared to the communication cost of uncoded FFT.*

To prove the theorem, we first identify the communication cost of uncoded FFT algorithm. Then, we analyze communication cost of encoding and decoding and we compare them to obtain the theorem.

A. Communication cost of uncoded FFT algorithm

Let us begin with understanding the communication cost of uncoded FFT algorithm. In Algorithm 1, steps that require communication are Step 1 and 3. Both steps need communication to transpose the data stored in distributed processors. For transposing the data, all processors have to exchange data with all the other processors. This communication is known as *"all-to-all"* communication. Bruck et al. showed lower bounds and explicit algorithms that achieve lower bounds for two special cases of all-to-all communication [31] – a minimum-communication-rounds regime and a minimum-bandwidth regime. Let us first formally define all-to-all communication.

Definition 1 (All-to-all)**.** *In all-to-all(p, n) communication, there are p nodes each of which stores n symbols. The data stored in the i-th node can be broken down into p data blocks, $M_{i,1}, \cdots M_{i,p}$, where the size of each block is n/p symbols. The goal of the communication is to transpose the data stored in p processors so that at the end of the communication, the i-th node has $M_{1,i}, \cdots, M_{p,i}$ data blocks.*

We will first give a simple lower bound of all-to-all(p, n) communication.

Theorem 3 (Proposition 2.3 and 2.4 in [31])**.** *For all-to-all(p, n) communication, C_1 and C_2 are lower bounded by:*

$$C_1 \geqslant \lceil \log_2 p \rceil, \quad C_2 \geqslant \frac{p-1}{p} n \tag{15}$$

However, Bruck et al. showed that the lower bounds on C_1 and C_2 cannot be achieved simultaneously which is stated in the theorem below.

Theorem 4 (Theorem 2.5 and 2.6 in [31])**.** *If all-to-all(p, n) communication uses the minimum number of rounds, i.e., $C_1 = \lceil \log_2 p \rceil$, C_2 is lower bounded by:*

$$C_2 \geqslant \frac{n}{2} \log_2 p. \tag{16}$$

If all-to-all(p, n) communication uses the minimum number of symbols transferred in sequence, i.e., $C_2 = \frac{p-1}{p} n$ symbols in a sequence, then C_1 is lower bounded by:

$$C_1 \geqslant p - 1. \tag{17}$$

Furthermore, both lower bounds are achievable.

Now, by using Theorem 4, we can give communication cost lower bounds on the transpose step in the distributed FFT algorithm.

Corollary 5. *The transpose step of N-point FFT requires the communication cost at least*

$$\lceil \log_2 K \rceil \alpha + \frac{1}{2} \frac{N}{K} \log_2 K \beta \tag{18}$$

when using the minimum communication rounds regime, and

$$(K-1)\alpha + \frac{(K-1)}{K} \frac{N}{K} \beta \tag{19}$$

when using the minimum communication bits regime.

Under our massively parallel system model where K is very large, we have $\log K << \sqrt{K}$. Hence, we should always choose the minimum-communication-round regime over the minimum-bandwidth regime. From now on, *we will only consider minimum communication round regime* and use its communication cost given in (18).

B. Communication overhead of coding

Now, let us identify additional communication cost due to coding in Algorithm 2. In the first encoding step where we compute column-wise parity symbols, we do not need any communication since processors already have column-wise data in the beginning. Also, for the first decoding in Step 5, column-wise decoding can be done in local processors as each processor has column-wise data after the transpose step. In Step 7, it requires inter-processor communication to encode row-wise parity symbols as one row of the data is spread over all the processors. Also in step 9, we have to perform row-wise decoding while every node has column-wise data, and thus we need inter-processor communication for decoding. Hence, in this section, *we will analyze the communication cost of the second encoding step and decoding step*. We will first show the communication cost of the second encoding step where we compute:

$$\tilde{Y} = Y G_2. \tag{20}$$

Before we begin our communication cost analysis, we want to make a few remarks.

Remark 1. *[Why do we need distributed encoding?] If we can do the second encoding, which is computing row-wise parity symbols, at local processors before the transpose step, we can avoid communication for distributed encoding at Step 7. However, there is no trivial way of doing this using a linear code due to the twiddle factors. After Step 3, the i-th processor has*

$$Y_i^{(\text{row})} = \tilde{X}_i^{(\text{row})} F_{N_1} = G_{1,i}^{(\text{row})} X F_{N_1}. \tag{21}$$

If we do row-wise encoding at the i-th processor locally before the transpose step, the i-th processor will have

$$\tilde{Y}_i^{(\text{row})} = G_{1,i}^{(\text{row})} X F_{N_2} G_2. \tag{22}$$

We then perform the transpose of the output from the first K successful nodes. The i-th node now has

$$\tilde{Y}_i^{(\text{col})} = G_{1,suc} X F_{N_2} G_{2,i}^{\text{col}}. \tag{23}$$

Column-wise decoding can be done locally by inverting $G_{1,suc}$:

$$\hat{Y}_i^{(\text{col})} = G_{1,suc}^{-1} G_{1,suc} X F_{N_2} G_{2,i}^{\text{col}} = X F_{N_2} G_{2,i}^{\text{col}}. \tag{24}$$

We now have to multiply twiddle factors to $\hat{Y}_i^{(\text{col})}$:

$$\hat{Y}_i^{(\text{col})} = T_N \circ \hat{Y}_i^{(\text{col})} = T_N \circ (X F_{N_2} G_{2,i}^{\text{col}}) \tag{25}$$

However, this will produce a different final output from what we expect because of the nonlinearity of Hadamard product:

$$A \circ (BC) \neq (A \circ B)C. \tag{26}$$

Hence,

$$T_{N,i}^{(col)} \circ (XF_{N_2} G_{2,i}^{\text{col}}) \neq (T_{N,i}^{(col)} \circ XF_{N_2}) G_{2,i}^{\text{col}}. \quad (27)$$

From our modified coding strategy, our final output from successful nodes will be $F_{N_1} T_N \circ (XF_{N_2} G_{2,\text{suc}})$ and even after decoding, we will have

$$F_{N_1} T_N \circ (XF_{N_2} G_{2,\text{suc}}) G_{2,\text{suc}}^{-1} \neq F_{N_1} T_N \circ (XF_{N_2}). \quad (28)$$

This means that we have to perform twiddle factor multiplication before proceeding to the row-wise encoding step. With the same argument, we can show that column-wise decoding must be done before multiplying twiddle factors. It concludes that because of the twiddle factors, the second-step encoding must be done across the processors incurring some communication cost.

We now want to analyze the communication cost of the second encoding step. Let us first investigate the communication cost of a simple encoding scheme where we add one parity node that stores the checksums of data, $X_1 + \cdots + X_K$. The encoding matrix G_2 for this can be written as follows:

$$\mathcal{G}_{\text{cks}} = \begin{bmatrix} & 1 \\ I_K & \vdots \\ & 1 \end{bmatrix} \quad (29)$$

$$G_2 = \mathcal{G}_{\text{cks}} \otimes I_{N_2/K} \quad (30)$$

For this computation, all K nodes have to send its data to one checksum node to compute the sum of all the data in the network. This is a well-known communication operation called *"reduce(-to-one)"*.

Definition 2 (Reduce)**.** *In reduce(p, n) communication, there are p data nodes which have data $M_1, \cdots, M_p$ of size n and one reduction node. The goal of the communication is to send $M_1 + \cdots + M_p$ to the reduction node.*

A lower bound on the communication cost of reduce(p, n) operation is given in the following theorem.

Theorem 6. *The communication cost of reduce(p, n) is lower bounded by*

$$\lceil \log_2 p \rceil \alpha + n\beta. \quad (31)$$

It was found that reduce operation can be done by reversing any broadcasting algorithm, where one broadcasting node sends its message to all the other processors in the network. Traff and Ripke [32] proposed a near-optimal broadcasting algorithm that achieves the lower bound (31) within a factor of 2 . By reversing their broadcasting algorithm, we can achieve the same communication cost for reduce(p, n) communication.

Theorem 7. *Reduce(p, n) can be done with the communication cost of at most*

$$(\sqrt{\lceil \log_2 p \rceil \alpha} + \sqrt{n\beta})^2 \leqslant 2(\lceil \log_2 p \rceil \alpha + n\beta). \quad (32)$$

Whether (32) is optimal or not is an open problem. We will use this as a state-of-the-art communication algorithm for reduce operation. By applying (32), we can obtain the communication cost for encoding one checksum node.

Corollary 8. *A $(K+1, K, 2)$ systematic MDS code over K systematic processors each of which hs N/K data symbols can be encoded with the communication cost of*

$$(\sqrt{\lceil \log_2 K \rceil \alpha} + \sqrt{N/K\beta})^2 \leqslant 2(\lceil \log_2 K \rceil \alpha + N/K\beta). \quad (33)$$

We can now extend computing checksums to computing parity symbols for a generic $(P, K, d = P - K + 1)$ systematic MDS code. Unlike checksum computation which only requires a single reduce(-to-one) operation, here we need multiple reductions to $P - K$ nodes.

From the intuition we got from reduce(-to-one) problem, we will first establish bounds for multi-broadcasting problem (will be defined below) and show that multi-reduce problem for encoding a $(P, K, d = P - K + 1)$ systematic MDS code can be solved by reversing the multi-broadcasting algorithm.

Definition 3 (Multi-broadcast)**.** *In multi-broadcast(p, r, n) communication, there are r broadcasting nodes and p destination nodes. Broadcasting nodes have distinct messages $M_1, \cdots, M_r$ of size n symbols. At the end of the communication, all p destination nodes should have all r messages, $M_1, \cdots, M_r$.*

We want to note that multi-message broadcasting has been studied in the literature [33], [34]. However, their models have one broadcasting node which sends multiple messages in a sequence. This is fundamentally different from our *multi-broadcast* which has multiple broadcasting nodes that can send out their messages simultaneously. To the best of our knowledge, communication cost analysis of this specific problem has not been studied before.

We will first show a communication algorithm for multi-broadcast(p, r, n) and then show that it achieves the lower bound within a factor of 2.

Theorem 9. *Multi-broadcast(p, r, n) can be done with the communication cost at most*

$$2(\lceil \log_2 p \rceil \alpha + rn\beta) \quad (34)$$

Proof. First, divide p processors into r disjoint sets of size p/r. Let us denote the sets as $S_1, S_2, \cdots, S_r$. The i-th broadcasting node broadcasts its message to all the nodes in S_i. With the optimal broadcasting algorithm [32], it takes communication cost of $(\sqrt{\log_2 \frac{p}{r} \alpha} + \sqrt{n\beta})^2$.

After the broadcasting step, the j-th nodes in S_i's $(i = 1, \cdots, r)$ communicate with each other so that all of them can share $M_1, \cdots, M_r$. This is all-gather(r, n) communication which is defined as follows.

Definition 4 (All-gather)**.** *In all-gather(p, n) communication, there are p nodes which have distinct messages $M_1, \cdots, M_p$ of size n symbols. At the end of the communication, all p nodes should have all p messages.*

All-gather(r, n) can be done with communication cost of $(\log_2 r)\alpha + (r-1)n\beta$ using the *bidirectional algorithm* [35].

The total communication cost of this two-step algorithm is

$$\begin{aligned}&(\sqrt{\log_2 \frac{p}{r}\alpha} + \sqrt{n\beta})^2 + \log_2 r\alpha + (r-1)n\beta \\ &\leqslant \lceil \log_2 p \rceil \alpha + rn\beta + (\log_2 \frac{p}{r}\alpha + n\beta) \\ &\leqslant 2(\lceil \log_2 p \rceil \alpha + rn\beta).\end{aligned}$$

□

We now show a lower bound for multi-broadcast(p,r,n) communication.

Theorem 10. *The communication cost of multi-broadcast(p,r,n) is lower bounded by*

$$\lceil \log_2 p \rceil \alpha + rn\beta \tag{35}$$

Proof. Each broadcasting node must communicate to p destination nodes which takes at least $\lceil \log_2 p \rceil$ communication rounds. Each destination node has to receive messages $M_1, \cdots M_r$ which have n. Hence, mutlti-broadcast(p,r,n) requires at least the bandwidth of rn. □

By comparing (34) and (35), we can see that the algorithm given in Theorem 9 achieves the lower bound within a factor of 2.

Finally, we define *multi-reduce* operation which is the communication required for encoding parity symbols, and show that it can be done with the same communication cost as multi-broadcast operation.

Definition 5 (Multi-reduce)**.** *In multi-broadcast(p,r,n) communication, there are p data nodes and r reduction nodes ($r < p$). p data nodes have data $M_1, \cdots, M_p$ each of which consist of n symbols. At the end of communication, the i-th reduction node will have $a_{i,1}M_1 + \cdots + a_{i,p}M_p$ where $a_{i,j}$'s ($i = 1, \cdots, r, j = 1, \cdots, p$) are chosen so that the data from any p nodes are linearly independent combinations of $M_1, \cdots, M_p$.*

Theorem 11. *Multi-reduce(p,r,n) communication can be done by reversing the multi-broadcast algorithm given in Theorem 9. Hence, the communication cost of multi-reduce(p,r,n) is at most*

$$2(\lceil \log_2 p \rceil \alpha + rn\beta) \tag{36}$$

Proof. Let $D_1, D_2, \cdots, D_p$ denote the data at p data processors. Let us divide data processors into r disjoint sets of size p/r and let S_i denote the set of indices of the i-th set: $S_i = \{(i-1)\cdot p/r + 1, \cdots, (i-1)\cdot p/r + p/r\}$. This is all-gather$(r,n)$ communication.

First, the j-th nodes in S_i's ($i = 1, \cdots, n$) perform all-gather communication. All the j-th processors in S_i's will have $D_j, D_{j+p/r}, \cdots, D_{j+(r-1)p/r}$ after the communication.

In the second step, all the nodes in S_i will carry out reduce communication with the i-th reduction node. Each node in S_i will compute a corresponding linear combination of the data it has and send only n symbols of data to the i-th reduction node. For instance, the j-th node in S_i will compute

$$a_{i,j}D_j + a_{i,j+p/r}D_{j+p/r} + \cdots + a_{i,j+(r-1)p/r}D_{j+(r-1)p/r}.$$

This is reduce$(p/r,n)$ which can be done with the communication cost of $(\sqrt{\log_2 \frac{p}{r}\alpha} + \sqrt{n\beta})^2$. This completes multi-reduce(p,r,n) communication. □

This gives an achievable communication scheme for encoding parity symbols and decoding systematic symbols of a (P,K,d) systematic MDS code.

Corollary 12. *A $(P,K,d = P-K+1)$ systematic MDS code over K systematic processors each of which has N/K data symbols can be encoded with the communication cost of*

$$2\left(\lceil \log_2 K \rceil \alpha + (P-K)\frac{N}{K}\beta\right). \tag{37}$$

Proof. The encoding matrix of $(P,K,P-K+1)$ MDS code has the form

$$\mathcal{G} = \begin{bmatrix} I_K & | & \mathcal{P} \end{bmatrix}$$

where I_K is a K-by-K identity matrix and $\mathcal{P}$ is a parity matrix of dimension K-by-$P-K$ whose entries are all non-zero [36]. This means that every parity symbol is a linear combination of all K symbols distributed in K nodes. Hence, encoding parity symbols for a systematic $(P,K,d = P - K+1)$ MDS code is exactly multi-reduce$(K,P-K,N/K)$ operation. Simply substituting this to (36) completes the proof. □

A similar argument can be applied to show that decoding at Step 11 of Algorithm 2 can also be done with the same communication cost.

Corollary 13. *Reconstructing N/K data symbols in failed systematic nodes of at Step 11 of Algorithm 2 can be done with the communication cost at most:*

$$2\left(\lceil \log_2 K \rceil \alpha + (P-K)\frac{N}{K}\beta\right). \tag{38}$$

Proof. First, note that we only have to recover the data in systematic nodes. The worst case is when there are $P-K$ failed nodes among the systematic nodes. In this case, the remaining K successful nodes have to send their data to $P-K$ systematic nodes. A failed node's data symbol can be represented as a linear combination of K output symbols from successful nodes. Hence, this is multi-reduce$(K,P-K,N/K)$ operation. □

C. Proof of Theorem 2

Proof. By comparing the encoding communication overhead given in (37) with the communication cost of uncoded FFT algorithm given in (18), we can prove our main theorem. Uncoded FFT algorithm requires two transpose operation, one in the beginning and one before the second FFT step. This requires communication cost of

$$2\left(\lceil \log_2 K \rceil \alpha + \frac{N}{2K}\lceil \log_2 K \rceil \beta\right) \tag{39}$$

If we compare this against the communication cost of encoding given in (37), the condition for the encoding cost to be smaller than the all-to-all communication is given as follows:

$$4\big(\lceil \log_2 K\rceil\alpha + (P-K)\frac{N}{K}\beta\big) < 2\big(\lceil \log_2 K\rceil\alpha + \frac{N}{2K}\lceil \log_2 K\rceil\beta\big)$$
$$P-K < \frac{\log_2 K}{4}.$$

Hence, as long as $P-K$ is smaller than $\frac{\log_2 K}{4}$ in scaling sense, communication overhead of coding is negligible compared to the intrinsic communication cost of uncoded distributed FFT algorithm. □

V. Conclusion and Future Work

Identifying the communication cost of the coded FFT algorithm for specific network topologies commonly used in HPC is an interesting future direction. Also, in the process of examining the communication cost of encoding/decoding in a given network topology, we believe that new innovative coding schemes can be discovered which are more communication efficient for a given topology. Also, codes that have sparse generator matrices, such as LT codes [37], [38], might be able to reduce communication overhead for encoding. Locally repairable codes with high availability [39] could reduce communication overhead of decoding. More generally, expanding our understanding beyond MDS codes, and establishing bounds on the trade-off between communication cost and error correction capability would be interesting.

References

[1] R. D. Ryne, "On FFT-based convolutions and correlations, with application to solving poisson's equation in an open rectangular pipe," *arXiv preprint arXiv:1111.4971*, 2011.

[2] R. W. Hockney, "A fast direct solution of poisson's equation using Fourier analysis," *Journal of the ACM (JACM)*, vol. 12, no. 1, 1965.

[3] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[4] M. Pippig, "PFFT: An extension of FFTW to massively parallel architectures," *SIAM Journal on Scientific Computing*, 2013.

[5] T. V. T. Duy and T. Ozaki, "A decomposition method with minimum communication amount for parallelization of multi-dimensional FFTs," *Computer Physics Communications*, vol. 185, no. 1, 2014.

[6] Bergman *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *DARPA IPTO*, 2008.

[7] K. Ferreira *et al.*, "Evaluating the viability of process replication reliability for exascale systems," in *SC 2011*. IEEE, 2011.

[8] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, 2010.

[9] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*.

[10] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[11] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," *arXiv preprint arXiv:1612.03301*, 2016.

[12] H. Park, K. Lee, J. yong Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," 2018.

[13] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," *arXiv preprint arXiv:1705.03875*, 2017.

[14] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *arXiv preprint arXiv:1705.10464*, 2017.

[15] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 709–719.

[16] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *High Performance Computing for Computational Science – VECPAR 2010*, J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–25.

[17] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded Fourier transform," *arXiv preprint arXiv:1710.06471*, 2017.

[18] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, Sept 2010.

[19] K. Rashmi, N. B. Shah, and K. Ramchandran, "A piggybacking design framework for read-and download-efficient distributed storage codes," *IEEE Transactions on Information Theory*, 2017.

[20] J. Sohn, B. Choi, and J. Moon, "A class of msr codes for clustered distributed storage," 2018.

[21] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Transactions on Information Theory*, Oct 2014.

[22] S. Kadhe and A. Sprintson, "Codes with unequal locality," in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016.

[23] V. Cadambe and A. Mazumdar, "An upper bound on the size of locally recoverable codes," in *2013 International Symposium on Network Coding (NetCod)*, June 2013, pp. 1–5.

[24] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, 2014.

[25] K.-H. Huang *et al.*, "Algorithm-based fault tolerance for matrix operations," *IEEE transactions on computers*, vol. 100, no. 6, 1984.

[26] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proceedings of the IEEE*, vol. 74, no. 5, 1986.

[27] J. Y. Jou and J. A. Abraham, "Fault-tolerant FFT networks," *IEEE Transactions on Computers*, vol. 37, no. 5, May 1988.

[28] D. L. Tao and C. R. P. Hartmann, "A novel concurrent error detection scheme for FFT networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, Feb 1993.

[29] S.-J. Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," *IEEE Transactions on Computers*, vol. 43, no. 7, Jul 1994.

[30] C. G. Oh, H. Y. Youn, and V. K. Raj, "An efficient algorithm-based concurrent error detection for FFT networks," *IEEE Transactions on Computers*, vol. 44, no. 9, Sep 1995.

[31] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on parallel and distributed systems*, 1997.

[32] J. L. Träff and A. Ripke, "Optimal broadcast for fully connected processor-node networks," *Journal of Parallel and Distributed Computing*, vol. 68, no. 7, 2008.

[33] P. Sanders and J. F. Sibeyn, "A bandwidth latency tradeoff for broadcast and reduction," *Information Processing Letters*, vol. 86, no. 1, 2003.

[34] A. Bar-Noy and S. Kipnis, "Broadcasting multiple messages in simultaneous send/receive systems," *Discrete Applied Mathematics*, 1994.

[35] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, 2007.

[36] R. E. Blahut, *Algebraic codes for data transmission*. Cambridge university press, 2003.

[37] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*. IEEE, 2002, pp. 271–280.

[38] A. Severinson, E. Rosnes *et al.*, "Block-diagonal and lt codes for distributed computing with straggling servers," *arXiv preprint arXiv:1712.08230*, 2017.

[39] S. Kadhe and R. Calderbank, "Rate optimal binary linear locally repairable codes with small availability," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 166–170.