Generalized Cross Subspace Alignment Codes for Coded Distributed Batch Matrix Multiplication

Zhuqing Jia and Syed A. Jafar
Center for Pervasive Communications and Computing (CPCC)
University of California, Irvine
Email: {zhuqingj, syed}@uci.edu

***Abstract*—The goal of coded distributed batch matrix multiplication is to efficiently multiply L instances of $\lambda \times \kappa$ matrices, $\mathbf{A} = (\mathbf{A}_1, \cdots, \mathbf{A}_L)$, with L instances of $\kappa \times \mu$ matrices $\mathbf{B} = (\mathbf{B}_1, \cdots, \mathbf{B}_L)$, by distributing the computation across S servers, such that the response from any R servers (R is called the recovery threshold) is sufficient to compute the L matrix products, $\mathbf{AB} = (\mathbf{A}_1\mathbf{B}_1, \mathbf{A}_2\mathbf{B}_2, \cdots, \mathbf{A}_L\mathbf{B}_L)$. Existing solutions either compute each $\mathbf{A}_l\mathbf{B}_l$ one at a time by partitioning individual matrices and coding across these partitions, or rely only on batch processing, i.e., coding across the batch of matrices without any matrix partitioning. The state-of-art for matrix-partitioning and batch processing approaches is represented by Entangled Polynomial Codes (EP codes), and Lagrange Coded Computing (LCC), respectively. In order to combine the benefits of the two approaches, we propose Generalized Cross-Subspace Alignment Codes (GCSA codes) that unify, generalize and improve upon the state of art. GCSA codes bridge the two extremes by efficiently combining both matrix-partitioning and batch processing, and offer flexibility in how much of each approach is used. Both EP codes and LCC codes can be recovered as special cases of GCSA codes. Remarkably, even without matrix partitioning, GCSA codes demonstrate an advantage over LCC codes in download-constrained settings. This is due to cross-subspace alignment, characterized by a Cauchy-Vandermonde code structure that aligns interference along Vandermonde terms, while the desired matrix products remain resolvable along Cauchy terms.**

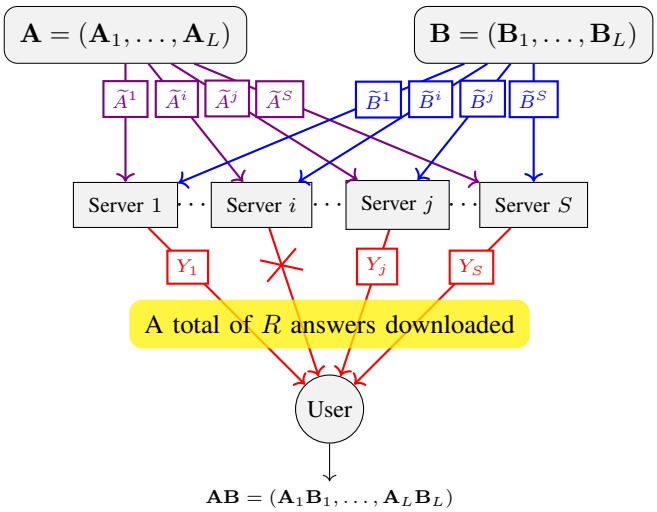

Fig. 1: *The CDBMM problem. Source (master) nodes generate matrices* $\mathbf{A} = (\mathbf{A}_1, \cdots, \mathbf{A}_L)$ *and* $\mathbf{B} = (\mathbf{B}_1, \cdots, \mathbf{B}_L)$*, and upload them to* S *distributed servers in coded form* $\widetilde{A}^{[s]}$*,* $\widetilde{B}^{[s]}$*, respectively. For all* $l \in [L]$*,* $\mathbf{A}_l$ *and* $\mathbf{B}_l$ *are* $\lambda \times \mu$ *and* $\mu \times \kappa$ *matrices, respectively, over a field* $\mathbb{F}$*. The* s^{th} *server computes the answer* Y_s*. Upon downloading answers from any* R *servers, the user must be able to recover the product* $\mathbf{AB} = (\mathbf{A}_1\mathbf{B}_1, \mathbf{A}_2\mathbf{B}_2, \ldots, \mathbf{A}_L\mathbf{B}_L)$*.*

I. Introduction

Matrix multiplication is an essential building block for computing applications. In the era of big data and cloud computing along with massive parallelization, there is particular interest in algorithms for distributed matrix multiplication that are resilient to stragglers [1]. Illustrated in Figure 1, the goal of coded distributed batch matrix multiplication (CDBMM) is to efficiently multiply L instances of $\lambda \times \kappa$ matrices, $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_L)$, with L instances of $\kappa \times \mu$ matrices $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_L)$, by distributing the computation task across S servers (through a coding scheme), such that the response from any R servers (R is called the recovery threshold) is sufficient for the user to compute the batch of L matrix products, $\mathbf{AB} = (\mathbf{A}_1\mathbf{B}_1, \mathbf{A}_2\mathbf{B}_2, \cdots, \mathbf{A}_L\mathbf{B}_L)$. The main metrics of interest for coded distributed computation include: the encoding and decoding complexity, server computation complexity, the recovery threshold, and the upload and download costs (communication costs).

Existing solutions to CDBMM fall into two distinct categories — those based on matrix partitioning [2]–[5], and those based on batch processing [6]. Matrix-partitioning approaches compute each of the L products $\mathbf{A}_l\mathbf{B}_l$ one at a time by partitioning individual matrices $\mathbf{A}_l$ and $\mathbf{B}_l$ and coding across these partitions. Batch processing approaches do not partition individual matrices, instead they code across the batch of $\mathbf{A}$ matrices and across the batch of $\mathbf{B}$ matrices. The state-of-art for matrix-partitioning approaches is represented by Entangled Polynomial Codes (EP codes) [5], while Lagrange Coded Computing (LCC) [6] represents the state of art for batch processing.

This work is motivated by two contrasting observations. On the one hand, we find that when normalized by the batch size (i.e., per matrix multiplication), batch processing approaches are much more efficient than matrix partitioning approaches in terms of their communication cost, as well as their encoding/decoding complexity. On the other hand, we note that since there is no partitioning of individual matrices in batch processing codes, this means that each server must carry a computational load equivalent to at least one full matrix multiplication. This presents a latency barrier for batch

processing schemes. For applications with stricter latency requirements such a solution may be infeasible, making it necessary to reduce the computational load per server by further parallelization, i.e., partitioning of individual matrices.

Recognizing this challenge, we propose Generalized Cross-Subspace Alignment Codes (GCSA codes) that unify, generalize and improve upon the state of art for CDBMM. GCSA codes bridge the two extremes by efficiently combining both matrix-partitioning and batch processing, and offer flexibility in how much of each approach is used. Both EP codes and LCC codes can be recovered as special cases of GCSA codes. Remarkably, even when no matrix partitioning is used, GCSA codes demonstrate an advantage over LCC codes in download-constrained settings. This is due to cross-subspace alignment (CSA), an idea that was originally introduced in the context of private information retrieval [7]–[9]. CSA has also been used recently to minimize download costs for secure and/or private matrix multiplication [9]–[11]. CSA is characterized by a Cauchy-Vandermonde code structure that facilitates interference alignment along Vandermonde terms, while the desired matrix products remain resolvable along the Cauchy terms. With GCSA codes, the degree of matrix partitioning controls the computational load per server, while the batch partitioning on top maintains the advantage of batch processing. How to efficiently combine the benefits of CSA with matrix partitioning is the key technical challenge behind this work. The combination is far from trivial. For example, consider a matrix partitioning approach that splits the task among 10 servers such that any $R_1 = 7$ need to respond, and a similar batch processing approach that also splits the task among 10 servers such that any $R_2 = 7$ need to respond. Then if we simply take the 10 matrix-partitioned sub-tasks and use batch processing on top to further distribute each sub-task among 10 servers, for a total of 100 servers, then the recovery threshold of the naive combination is $6 \times 10 + 4 \times 6 + 1 = 85$. However, GCSA code achieves the significantly lower recovery threshold of $R = R_1 R_2 = 49$ instead.

II. State of Art: EP Codes, LCC Codes

In this section, we provide an overview of the state of art approaches for CDBMM (EP codes and LCC codes), and compare them with GCSA codes.

A. Matrix Partitioning: EP Codes [5]

Entangled Polynomial (EP) codes [5] for coded distributed matrix multiplication problem are based on matrix partitioning. The constituent matrices $\mathbf{A}$ and $\mathbf{B}$ are partitioned into $m \times p$ blocks and $p \times n$ blocks, respectively, so that the desired matrix product involves a total of mn linear combinations of products of block matrices. Coded matrices are constructed as follows,

$$\widetilde{A}(\alpha) = \sum_{m' \in [m]} \sum_{p' \in [p]} \mathbf{A}^{m',p'} \alpha^{p'-1+p(m'-1)}, \tag{1}$$

$$\widetilde{B}(\alpha) = \sum_{p' \in [p]} \sum_{n' \in [n]} \mathbf{B}^{p',n'} \alpha^{p-p'+pm(n'-1)}, \tag{2}$$

and the s^{th} server is sent the values $\widetilde{A}(\alpha_s)$ and $\widetilde{B}(\alpha_s)$. Here $\alpha_1, \alpha_2, \cdots, \alpha_S$ are distinct elements from $\mathbb{F}$. If responsive, Server s returns $\widetilde{A}(\alpha_s)\widetilde{B}(\alpha_s)$, which can be expressed as

$$\widetilde{A}(\alpha)\widetilde{B}(\alpha) = \sum_{i=1}^{R} \mathbf{C}^{(i)} \alpha^{i-1}, \tag{3}$$

where $R = pmn + p - 1$ is the recovery threshold, and $\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \cdots, \mathbf{C}^{(R)}$ are various linear combinations of products of matrix blocks. Note that for all $i \in [R]$, $\mathbf{C}^{(i)}$ are distributed over $1, \alpha, \cdots, \alpha^{R-1}$, thus from the answers of any R servers, $\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \cdots, \mathbf{C}^{(R)}$ are recoverable by inverting a Vandermonde matrix. Furthermore, it is proved in [5] that by the construction of $\widetilde{A}(\alpha)$ and $\widetilde{B}(\alpha)$, the $\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \cdots, \mathbf{C}^{(R)}$ terms include the mn desired terms, while the remaining undesired terms (interference) align into the remaining $R - mn$ dimensions. Remarkably, EP codes may be seen as bridging the extremes of Polynomial codes and MatDot codes. Polynomial codes [2] can be recovered from EP codes by setting $p = 1$, and MatDot codes [3] can be obtained from EP codes by setting $m = n = 1$. EP codes also represent an improvement of PolyDot codes [3] within a factor of 2 in terms of recovery threshold, due to better interference alignment. Finally, EP codes have similar performance as Generalized PolyDot codes [4]. Thus, EP codes represent the state of art of prior work in terms of matrix partitioning approaches to coded distributed matrix multiplication.

B. Batch Processing: LCC Codes [6]

Lagrange Coded Computing (LCC) codes [6] represent the state of art of prior work in terms of batch processing approaches for coded distributed batch multivariate polynomial evaluation, which includes as a special case CDBMM. LCC codes are so named because they exploit the Lagrange interpolation polynomial to encode input data. For example, suppose we are interested in L evaluations of the multivariate polynomial $\Phi(\cdot)$ of total degree N, namely $\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2), \cdots, \Phi(\mathbf{x}_L)$ over the given batch of L data points $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L$. Note that for matrix multiplication, $\mathbf{x}_l = (\mathbf{A}_l, \mathbf{B}_l)$ and $\Phi(\mathbf{x}_l) = \mathbf{A}_l \mathbf{B}_l$, which is a bilinear operation ($N = 2$). LCC codes encode the dataset according to the Lagrange interpolation polynomial,

$$\widetilde{X}(\alpha) = \sum_{l \in [L]} \mathbf{x}_l \prod_{l' \in [L] \setminus \{l\}} \frac{\alpha - \beta_{l'}}{\beta_l - \beta_{l'}}, \tag{4}$$

and the s^{th} server is sent the evaluation $\widetilde{X}(\alpha_s)$. Here $\alpha_1, \alpha_2, \cdots, \alpha_S, \beta_1, \beta_2, \cdots, \beta_L$ are $(S + L)$ distinct elements from the operation field $\mathbb{F}$. The s^{th} server returns the user with the answer $\Phi(\widetilde{X}(\alpha_s))$. Note that the degree of the polynomial $\Phi(\widetilde{X}(\alpha))$ is less than or equal to $N(L-1) = NL - N$. Therefore, from the answers of any $R = NL - N + 1$ servers, the user is able to reconstruct the polynomial $\Phi(\widetilde{X}(\alpha))$ by polynomial interpolation. Upon obtaining the polynomial $\Phi(\widetilde{X}(\alpha))$, the user evaluates it at β_l for every $l \in [L]$ to obtain $\Phi(\widetilde{X}(\beta_l)) = \Phi(\mathbf{x}_l)$.

For ease of reference, Table I compares EP codes, LCC codes and GCSA codes with respect to their recovery thresholds, communication costs (uploads and downloads), encoding

	Recovery Threshold (R)	Upload Cost (U_A, U_B)	Download Cost (D)
EP	$pmn+p-1$	$S/(pm), S/(pn)$	$(pmn+p-1)/(mn)$
codes	R	$\mathcal{O}(m), \mathcal{O}(m)$	$\mathcal{O}(R/m^2)$
LCC	$2K_c'-1$	$S/K_c', S/K_c'$	$(2K_c'-1)/K_c'$
codes	R	$\mathcal{O}(1), \mathcal{O}(1)$	$\mathcal{O}(1)$
GCSA	$((\ell+1)K_c-1)(pmn+p-1)$	$S/(K_cpm), S/(K_cpn)$	$\frac{((\ell+1)K_c-1)(pmn+p-1)}{\ell K_c mn}$
codes	R	$\mathcal{O}(m), \mathcal{O}(m)$	$\mathcal{O}(p)$
	Server Computation Complexity ($\mathcal{C}_s$)	Encoding Complexity ($\mathcal{C}_{eA}, \mathcal{C}_{eB}$)	Decoding Complexity ($\mathcal{C}_d$)
EP	$\mathcal{O}\left(\lambda\mu\kappa/pmn\right)$	$\widetilde{\mathcal{O}}\left((\lambda\kappa/(pm))S\log^2 S\right), \widetilde{\mathcal{O}}\left((\kappa\mu/(pn))S\log^2 S\right)$	$\widetilde{\mathcal{O}}(\lambda\mu p\log^2 R)$
codes	$\mathcal{O}\left(\lambda^3/R\right)$	$\widetilde{\mathcal{O}}\left(\lambda^2 m\log^2 S\right), \widetilde{\mathcal{O}}\left(\lambda^2 m\log^2 S\right)$	$\widetilde{\mathcal{O}}\left((\lambda^2/m^2)R\log^2 R\right)$
LCC	$\mathcal{O}\left(\lambda\mu\kappa/K_c'\right)$	$\widetilde{\mathcal{O}}\left((\lambda\kappa/K_c')S\log^2 S\right), \widetilde{\mathcal{O}}\left((\kappa\mu/K_c')S\log^2 S\right)$	$\widetilde{\mathcal{O}}(\lambda\mu\log^2 R)$
codes	$\mathcal{O}\left(\lambda^3/R\right)$	$\widetilde{\mathcal{O}}\left(\lambda^2\log^2 S\right), \widetilde{\mathcal{O}}\left(\lambda^2\log^2 S\right)$	$\widetilde{\mathcal{O}}(\lambda^2\log^2 R)$
GCSA	$\mathcal{O}\left(\lambda\mu\kappa/K_c''\right)$	$\widetilde{\mathcal{O}}\left((\lambda\kappa/(K_cpm)S\log^2 S\right), \widetilde{\mathcal{O}}\left((\kappa\mu/(K_cpn)S\log^2 S\right)$	$\widetilde{\mathcal{O}}(\lambda\mu p\log^2 R)$
codes	$\mathcal{O}\left(\lambda^3/R\right)$	$\widetilde{\mathcal{O}}\left(\lambda^2 m\log^2 S\right), \widetilde{\mathcal{O}}\left(\lambda^2 m\log^2 S\right)$	$\widetilde{\mathcal{O}}(\lambda^2 p\log^2 R)$

TABLE I: *Performance summary of EP [5], LCC [6] and GCSA codes for CDBMM. Shaded rows represent balanced settings with $m=n, \lambda=\mu=\kappa$, and fixed ratio R/S. The batch size is $L=\ell K_c$ for GCSA codes, and $L'=K_c'$ for LCC codes.*

and decoding complexity, and server computation complexity. Note that choosing $\ell = K_c = 1$ reduces GCSA codes to EP codes, while setting $m = n = p = 1$ recovers batch-processing codes (further restricting $\ell = 1$ recovers LCC codes). The comparison is further illustrated through an example setting in Fig. 2, which shows lower convex hulls of achievable (balanced upload cost, download cost) pairs of GCSA codes for various bounds on $R_E = pmn + p - 1$, given that the number of servers $S = 300$ and the overall recovery threshold $R \leq 250$. Each value of (S, R, R_E) produces an achievable region in the (U, D) plane (including all possible choices of parameters m, n, p, ℓ, K_c). What is shown in the figure is the union of these regions. The larger the value of R_E, the more the GCSA code construction shifts toward EP codes, generally with the benefit of reduced computation complexity per server that comes with matrix partitioning. On the other hand, the smaller the value of R_E, the more the GCSA code construction shifts toward batch processing, with the benefit of improved communication costs. The figure also shows that GCSA codes in general improve upon LCC codes in terms of download cost by choosing $\ell > 1$.

Notation: $[N]$ stands for the set $\{1, 2, \ldots, N\}$. $X_{[N]}$ denotes the set $\{X_1, X_2, \ldots, X_N\}$. For $\mathcal{I} = \{i_1, i_2, \ldots, i_N\}$, $X_{\mathcal{I}}$ denotes the set $\{X_{i_1}, X_{i_2}, \ldots, X_{i_N}\}$. The notation $\otimes$ denotes the Kronecker product of two matrices. $\mathbf{I}_N$ denotes the $N \times N$ identity matrix. $\mathbf{T}(X_1, X_2, \cdots, X_N)$ denotes the $N \times N$ lower triangular Toeplitz matrix, with the terms in each column below the diagonal sequentially labeled $X_1, X_2, \cdots$. For square matrices $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_N$, $\mathbf{D}(\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_N)$ denotes the block diagonal matrix composed with blocks $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_N$. The notation $\widetilde{\mathcal{O}}(a \log^2 b)$ suppresses polylog terms. It may be replaced with $\mathcal{O}(a \log^2 b)$ if the field supports the Fast Fourier Transform (FFT), and with $\mathcal{O}(a \log^2 b \log\log(b))$ if it does not.

III. Problem Statement

As shown in Figure 1, consider two source (master) nodes, each of which generates a sequence of L matrices, denoted as $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_L)$ and $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_L)$, such that for all $l \in [L]$, we have $\mathbf{A}_l \in \mathbb{F}^{\lambda \times \kappa}$ and $\mathbf{B}_l \in \mathbb{F}^{\kappa \times \mu}$, i.e., $\mathbf{A}_l$ and $\mathbf{B}_l$ are $\lambda \times \kappa$ and $\kappa \times \mu$ matrices, respectively, over a finite[1] field $\mathbb{F}$. The sink node (user) is interested in the sequence of product matrices, $\mathbf{AB} = (\mathbf{A}_1\mathbf{B}_1, \mathbf{A}_2\mathbf{B}_2, \ldots, \mathbf{A}_L\mathbf{B}_L)$. To help with this computation, there are S servers (worker nodes). Each of the sources encodes its matrices according to the functions $\boldsymbol{f} = (f_1, f_2, \ldots, f_S)$ and $\boldsymbol{g} = (g_1, g_2, \ldots, g_S)$, where f_s and g_s correspond to the s^{th} server. Specifically, let us denote the encoded matrices for the s^{th} server as $\widetilde{A}^s$ and $\widetilde{B}^s$, so we have

$$\widetilde{A}^s = f_s(\mathbf{A}), \tag{5}$$

$$\widetilde{B}^s = g_s(\mathbf{B}). \tag{6}$$

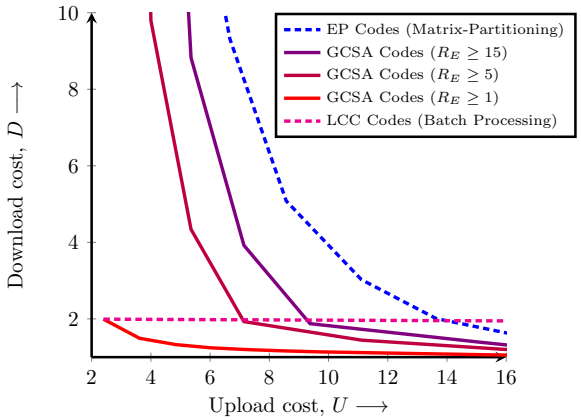

Fig. 2: *Lower convex hulls of achievable (balanced upload cost, download cost) pairs (U, D) of GCSA codes for various bounds on $R_E = pmn + p - 1$, given that $S = 300$ and the overall recovery threshold $R \leq 250$. Note that EP codes [5] and LCC codes [6] are also special cases of GCSA codes, obtained by setting $\ell = K_c = 1$, and $\ell = m = n = p = 1$, respectively.*

[1]Our coding schemes are applicable over infinite fields $(\mathbb{R}, \mathbb{C})$ as well. However, the problem statement assumes $\mathbb{F}$ is a finite field, because of the difficulty of defining communication or computation costs over infinite fields.

The encoded matrices, $\widetilde{A}^s, \widetilde{B}^s$, are uploaded to the s^{th} server. Let us denote the number of elements from $\mathbb{F}$ in $\widetilde{A}^s$ and $\widetilde{B}^s$ as $|\widetilde{A}^s|$ and $|\widetilde{B}^s|$, respectively.

Upon receiving the encoded matrices, each Server s, $s \in [S]$, if responsive, returns Y_s, that is a function of $\widetilde{A}^s$ and $\widetilde{B}^s$,

$$Y_s = h_s(\widetilde{A}^s, \widetilde{B}^s), \tag{7}$$

where $h_s, s \in [S]$ are the functions used to produce the answer, and we denote them collectively as $\boldsymbol{h} = (h_1, h_2, \ldots, h_S)$. Some servers may fail to respond, such servers are called stragglers. The user downloads the responses from the remaining servers, from which, using a class of decoding functions (denoted $\boldsymbol{d}$), he attempts to recover the desired product $\mathbf{AB}$. Define

$$\boldsymbol{d} = \{d_{\mathcal{R}} : \mathcal{R} \subset [S]\}, \tag{8}$$

where $d_{\mathcal{R}}$ is the decoding function used when the set of responsive servers is $\mathcal{R}$. We say that $(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h}, \boldsymbol{d})$ form a CDBMM code. A CDBMM code is said to be r-recoverable if the user is able to recover the desired products from the answers obtained from any r servers. In particular, a CDBMM code $(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h}, \boldsymbol{d})$ is r-recoverable if for any $\mathcal{R} \subset [S]$, $|\mathcal{R}| = r$, and for any realization of $\mathbf{A}$, $\mathbf{B}$, we have

$$\mathbf{AB} = d_{\mathcal{R}}(Y_{\mathcal{R}}). \tag{9}$$

Define the recovery threshold R of a CDBMM code $(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h}, \boldsymbol{d})$ to be the minimum integer r such that the CDBMM code is r-recoverable.

The communication cost of CDBMM is comprised of upload and download costs. The (normalized)[2] upload costs U_A and U_B are defined as follows.

$$U_A = \frac{\sum_{s\in[S]} |\widetilde{A}^s|}{L\lambda\kappa}, \qquad U_B = \frac{\sum_{s\in[S]} |\widetilde{B}^s|}{L\kappa\mu}. \tag{10}$$

Similarly, the (normalized) download cost is defined as follows.

$$D = \max_{\mathcal{R}, \mathcal{R}\subset[S], |\mathcal{R}|=R} \frac{\sum_{s\in\mathcal{R}} |Y_s|}{L\lambda\mu}, \tag{11}$$

where $|Y_s|$ is the number of elements from $\mathbb{F}$ in Y_s.

Next let us consider the complexity of encoding, decoding and server computation. Define the (normalized) computational complexity at each server, $\mathcal{C}_s$, to be the order of the number of arithmetic operations required to compute the function h_s at each server, normalized by L. Similarly, define the (normalized) encoding computational complexity $\mathcal{C}_{eA}$ for $\widetilde{A}^{[S]}$ and $\mathcal{C}_{eB}$ for $\widetilde{B}^{[S]}$ as the order of the number of arithmetic operations required to compute the functions $\boldsymbol{f}$ and $\boldsymbol{g}$, respectively, each normalized by L. Finally, define the (normalized) decoding computational complexity $\mathcal{C}_d$ to be the order of the number of arithmetic operations required to compute $d_{\mathcal{R}}(Y_{\mathcal{R}})$, maximized over $\mathcal{R}, \mathcal{R} \subset [S], |\mathcal{R}| = R$, and normalized by L. Note that normalizations by L are needed to have fair comparisons between batch processing approaches and individual matrix-partitioning solutions *per matrix multiplication*.

[2]Upload cost and download cost are normalized with the number of elements of $\mathbb{F}$ contained in $\mathbf{A}, \mathbf{B}$, and the desired product $\mathbf{AB}$, respectively.

IV. Main Result

Our main result appears in the following theorem.

Theorem 1. *For CDBMM over a field $\mathbb{F}$ with S servers, and positive integers (ℓ, K_c, p, m, n) such that $m \mid \lambda$, $p \mid \mu$, $n \mid \kappa$ and $L = \ell K_c \leq |\mathbb{F}| - S$, the GCSA codes presented in this work achieve*

Recovery Threshold: $R = ((\ell+1)K_c - 1)(pmn + p - 1)$,

Upload Cost for $\widetilde{A}^{[S]}, \widetilde{B}^{[S]}$: $(U_A, U_B) = \left(\frac{S}{K_c pm}, \frac{S}{K_c pn}\right)$,

Download Cost: $D = \left(\frac{(\ell+1)K_c - 1}{\ell K_c}\right)\left(\frac{pmn + p - 1}{mn}\right)$,

Server Computation Complexity: $\mathcal{C}_s = \mathcal{O}\left(\frac{\lambda\kappa\mu}{K_c pmn}\right)$,

Encoding Complexity for $\widetilde{A}^{[S]}, \widetilde{B}^{[S]}$:

$$(\mathcal{C}_{eA}, \mathcal{C}_{eB}) = \left(\widetilde{\mathcal{O}}\left(\frac{\lambda\kappa S \log^2 S}{K_c pm}\right), \widetilde{\mathcal{O}}\left(\frac{\kappa\mu S \log^2 S}{K_c pn}\right)\right),$$

Decoding Complexity: $\mathcal{C}_d = \widetilde{\mathcal{O}}\left(\lambda\mu p \log^2 R\right)$.

V. Proof of Theorem 1

Let us recall a standard result for Confluent Cauchy-Vandermonde matrices [12].

Lemma 1. *If $f_{1,1}, f_{1,2}, \cdots, f_{\ell,K_c}, \alpha_1, \alpha_2, \cdots, \alpha_R$ are $R + L$ distinct elements of $\mathbb{F}$, with $|\mathbb{F}| \geq R + L$ and $L = \ell K_c$, then the $R \times R$ Confluent Cauchy-Vandermonde matrix in* (31) *is invertible over $\mathbb{F}$.*

Let us now present the general construction of GCSA codes. Simple examples to illustrate the construction are provided in the full paper [13]. Define $R_E = pmn + p - 1$, to be the recovery threshold of Entangled Polynomial component of GCSA codes. Let $f_{1,1}$, $f_{1,2}$, $\cdots$, f_{ℓ,K_c}, α_1, α_2, $\cdots$, α_S be $(S+L)$ distinct elements from the field $\mathbb{F}$. For all $l \in [\ell], k \in [K_c]$, we define $c_{l,k,i}, i \in \{0, 1, \cdots, R_E(K_c - 1)\}$ to be the coefficients satisfying

$$\Psi_{l,k}(\alpha) = \prod_{k'\in[K_c]\setminus\{k\}} (\alpha + (f_{l,k'} - f_{l,k}))^{R_E} = \sum_{i=0}^{R_E(K_c-1)} c_{l,k,i}\alpha^i,$$

i.e., they are the coefficients of the polynomial $\Psi_{l,k}(\alpha) = \prod_{k'\in[K_c]\setminus\{k\}} (\alpha + (f_{l,k'} - f_{l,k}))^{R_E}$, which is defined here by its roots. Now for all $l \in [\ell], s \in [S]$, let us define

$$\Delta_s^{l,K_c} = \prod_{k\in[K_c]} (f_{l,k} - \alpha_s)^{R_E}. \tag{12}$$

Split the $L = \ell K_c$ instances of $\mathbf{A}$ and $\mathbf{B}$ matrices into

$$\mathbf{A}_{l,k} = \mathbf{A}_{K_c(l-1)+k}, \tag{13}$$

$$\mathbf{B}_{l,k} = \mathbf{B}_{K_c(l-1)+k}, \tag{14}$$

for all $l \in [\ell], k \in [K_c]$. Further, for each matrix $\mathbf{A}_{l,k}$, we partition it into $m \times p$ blocks, denoted as $\mathbf{A}_{l,k}^{1,1}, \mathbf{A}_{l,k}^{1,2}, \cdots, \mathbf{A}_{l,k}^{m,p}$.

Similarly, for each matrix $\mathbf{B}_{l,k}$, we partition it into $p \times n$ blocks, denoted as $\mathbf{B}_{l,k}^{1,1}, \mathbf{B}_{l,k}^{1,2}, \cdots, \mathbf{B}_{l,k}^{p,n}$. Now, for all $l \in [\ell], k \in [K_c]$, let us define

$$P_s^{l,k} = \sum_{m' \in [m]} \sum_{p' \in [p]} \mathbf{A}_{l,k}^{m',p'} (f_{l,k} - \alpha_s)^{p'-1+p(m'-1)}, \quad (15)$$

$$Q_s^{l,k} = \sum_{p' \in [p]} \sum_{n' \in [n]} \mathbf{B}_{l,k}^{p',n'} (f_{l,k} - \alpha_s)^{p-p'+pm(n'-1)}, \quad (16)$$

i.e., we apply EP codes for each $\mathbf{A}_{l,k}$ and $\mathbf{B}_{l,k}$. Note that the original EP codes can be regarded as polynomials of α_s, whereas here for each (l,k), we construct the EP codes as polynomials of $(f_{l,k} - \alpha_s)$. Now recall that by the construction of EP codes, the product $P_s^{l,k} Q_s^{l,k}$ can be written as weighted sums of the terms $1, (f_{l,k} - \alpha_s), \cdots, (f_{l,k} - \alpha_s)^{R_E - 1}$,

$$P_s^{l,k} Q_s^{l,k} = \sum_{i=0}^{R_E - 1} \mathbf{C}_{l,k}^{(i+1)} (f_{l,k} - \alpha_s)^i, \quad (17)$$

where $\mathbf{C}_{l,k}^{(1)}, \mathbf{C}_{l,k}^{(2)}, \cdots, \mathbf{C}_{l,k}^{(R_E)}$ are various linear combinations of products of blocks of $\mathbf{A}_{l,k}$ and blocks of $\mathbf{B}_{l,k}$, and the desired product $\mathbf{A}_{l,k}\mathbf{B}_{l,k}$ can be obtained from them. Now for all $s \in [S]$, let us construct

$$\widetilde{A}^s = (\widetilde{A}_1^s, \widetilde{A}_2^s, \ldots, \widetilde{A}_\ell^s), \quad \widetilde{B}^s = (\widetilde{B}_1^s, \widetilde{B}_2^s, \ldots, \widetilde{B}_\ell^s), \quad (18)$$

where for $l \in [\ell]$, we have

$$\widetilde{A}_l^s = \Delta_s^{l,K_c} \sum_{k \in [K_c]} \frac{1}{(f_{l,k} - \alpha_s)^{R_E}} P_s^{l,k}, \quad (19)$$

$$\widetilde{B}_l^s = \sum_{k \in [K_c]} \frac{1}{(f_{l,k} - \alpha_s)^{R_E}} Q_s^{l,k}. \quad (20)$$

The answer returned by the s^{th} server to the user is constructed as $Y_s = \sum_{l \in [\ell]} \widetilde{A}_l^s \widetilde{B}_l^s$. Let us prove that GCSA codes are $R = ((\ell+1)K_c - 1)(pmn + p - 1)$ recoverable.

$$Y_s = \widetilde{A}_1^s \widetilde{B}_1^s + \widetilde{A}_2^s \widetilde{B}_2^s + \cdots + \widetilde{A}_\ell^s \widetilde{B}_\ell^s \quad (21)$$

$$= \sum_{l \in [\ell]} \Delta_s^{l,K_c} \left(\sum_{k \in [K_c]} \frac{1}{(f_{l,k} - \alpha_s)^{R_E}} P_s^{l,k} \right) \left(\sum_{k \in [K_c]} \frac{1}{(f_{l,k} - \alpha_s)^{R_E}} Q_s^{l,k} \right) \quad (22)$$

$$= \sum_{l \in [\ell]} \sum_{k \in [K_c]} \frac{\prod_{k' \in [K_c] \setminus \{k\}} (f_{l,k'} - \alpha_s)^{R_E}}{(f_{l,k} - \alpha_s)^{R_E}} P_s^{l,k} Q_s^{l,k} + \sum_{l \in [\ell]} \sum_{\substack{k,k' \in [K_c] \\ k \neq k'}} \left(\prod_{\substack{k'' \in [K_c], \\ k'' \neq k,k'}} (f_{l,k''} - \alpha_s)^{R_E} \right) P_s^{l,k} Q_s^{l,k'}. \quad (23)$$

Let us consider the first term in (23). For each $l \in [\ell], k \in [K_c]$, we have

$$\frac{\prod_{k' \in [K_c] \setminus \{k\}} (f_{l,k'} - \alpha_s)^{R_E}}{(f_{l,k} - \alpha_s)^{R_E}} P_s^{l,k} Q_s^{l,k} \quad (24)$$

$$= \frac{\prod_{k' \in [K_c] \setminus \{k\}} \left((f_{l,k} - \alpha_s) + (f_{l,k'} - f_{l,k}) \right)^{R_E}}{(f_{l,k} - \alpha_s)^{R_E}} P_s^{l,k} Q_s^{l,k} \quad (25)$$

$$= \frac{\Psi_{l,k}(f_{l,k} - \alpha_s)}{(f_{l,k} - \alpha_s)^{R_E}} P_s^{l,k} Q_s^{l,k} \quad (26)$$

$$= \left(\frac{c_{l,k,0}}{(f_{l,k} - \alpha_s)^{R_E}} + \cdots + \frac{c_{l,k,R_E-1}}{f_{l,k} - \alpha_s} \right) P_s^{l,k} Q_s^{l,k} + \left(\sum_{i=R_E}^{R_E(K_c-1)} c_{l,k,i} (f_{l,k} - \alpha_s)^{i-R_E} \right) P_s^{l,k} Q_s^{l,k}, \quad (27)$$

where in (26), we used the definition of $\Psi_{l,k}(\cdot)$, and in the next step, we rewrite the polynomial $\Psi_{l,k}(f_{l,k} - \alpha_s)$ in terms of its coefficients. Let us consider the first term in (27).

$$\left(\frac{c_{l,k,0}}{(f_{l,k} - \alpha_s)^{R_E}} + \cdots + \frac{c_{l,k,R_E-1}}{f_{l,k} - \alpha_s} \right) P_s^{l,k} Q_s^{l,k} \quad (28)$$

$$= \left(\frac{c_{l,k,0}}{(f_{l,k} - \alpha_s)^{R_E}} + \cdots + \frac{c_{l,k,R_E-1}}{f_{l,k} - \alpha_s} \right) \sum_{i=0}^{R_E-1} \mathbf{C}_{l,k}^{(i+1)} (f_{l,k} - \alpha_s)^i$$

$$= \sum_{i=0}^{R_E-1} \frac{\sum_{i'=0}^{i} c_{l,k,i-i'} \mathbf{C}_{l,k}^{(i'+1)}}{(f_{l,k} - \alpha_s)^{R_E - i}} + \sum_{i=0}^{R_E-2} (f_{l,k} - \alpha_s)^i \left(\sum_{i'=i+1}^{R_E-1} c_{l,k,R_E-i'+i} \mathbf{C}_{l,k}^{(i'+1)} \right). \quad (29)$$

We further note that both the second term in (23) and the second term in (27) can be expanded into weighted sums of the terms $1, \alpha_s, \cdots, \alpha_s^{R_E(K_c-1)-1}$. Because $R_E(K_c - 1) - 1 = R - R_E L - 1$, in the matrix form, answers from any $R = ((\ell+1)K_c - 1)(pmn + p - 1)$ servers, whose indices are denoted as $s_1, s_2, \cdots, s_R$, can be written as follows.

$$\begin{bmatrix} Y_{s_1} \\ \vdots \\ Y_{s_R} \end{bmatrix} = \hat{\mathbf{V}}_{\ell,K_c,R_E,R} \hat{\mathbf{V}}'_{\ell,K_c,R_E,R} \otimes \mathbf{I}_{\lambda/m} \mathbf{C}, \quad (30)$$

where $\hat{\mathbf{V}}_{\ell,K_c,R_E,R}$ is defined in (31),

$$\hat{\mathbf{V}}'_{\ell,K_c,R_E,R} = \mathbf{D}(\mathbf{T}(c_{1,1,0}, \cdots, c_{1,1,R_E-1}), \cdots, \mathbf{T}(c_{\ell,K_c,0}, \cdots, c_{\ell,K_c,R_E-1}), \mathbf{I}_{R-R_E L}),$$

$$\mathbf{C} = \left[\mathbf{C}_{1,1}^{(1)}, \cdots, \mathbf{C}_{1,1}^{(R_E)} \middle| \cdots \middle| \mathbf{C}_{\ell,K_c}^{(1)}, \cdots, \mathbf{C}_{\ell,K_c}^{(R_E)} \middle| * \cdots * \right]^T,$$

and we have used $*$ to represent various combinations of interference symbols that can be found explicitly by expanding (23), whose exact forms are irrelevant. Now since $f_{1,1}, f_{1,2}, \cdots, f_{\ell,K_c}$ are distinct, for all $l \in [\ell], k \in [K_c]$, we must have

$$c_{l,k,0} = \prod_{k' \in [K_c] \setminus \{k\}} (f_{l,k'} - f_{l,k})^{R_E} \quad (32)$$

are non-zero. Hence, the lower triangular toeplitz matrices $\mathbf{T}(c_{1,1,0}, \cdots, c_{1,1,R_E-1}), \cdots, \mathbf{T}(c_{\ell,K_c,0}, \cdots, c_{\ell,K_c,R_E-1})$ are non-singular, and the block diagonal matrix $\hat{\mathbf{V}}'_{\ell,K_c,R_E,R}$ is

invertible. Guaranteed by Lemma 1 and the fact that the Kronecker product of non-singular matrices is non-singular, the matrix $(\hat{\mathbf{V}}_{\ell,K_c,R_E,R}\hat{\mathbf{V}}'_{\ell,K_c,R_E,R})\otimes\mathbf{I}_{\lambda/m}$ is invertible. Therefore, the user is able to recover $(\mathbf{C}_{l,k}^{(i)})_{l\in[\ell],k\in[K_c],i\in[R_E]}$ by inverting the matrix. And the desired products $(\mathbf{A}_l\mathbf{B}_l)_{l\in[L]}$ are recoverable from $(\mathbf{C}_{l,k}^{(i)})_{l\in[\ell],k\in[K_c],i\in[R_E]}$, guaranteed by the correctness of Entangled Polynomial codes. This completes the proof of recovery threshold $R=((\ell+1)K_c-1)(pmn+p-1)$. It is also easy to see that the upload cost $U_A=S/(K_cpm)$ and $U_B=S/(K_cpn)$. Note that we are able to recover Lmn desired symbols from R downloaded symbols, the download cost is $D=\frac{R}{Lmn}=\left(\frac{(\ell+1)K_c-1}{\ell K_c}\right)\left(\frac{pmn+p-1}{mn}\right)$. Thus the desired costs are achievable. Note that the encoding procedure can be considered as products of Confluent Cauchy matrices by vectors. By fast algorithms [14], the encoding complexity of $(\mathcal{C}_{eA},\mathcal{C}_{eB})=\left(\widetilde{\mathcal{O}}\left(\frac{\lambda\kappa S\log^2 S}{K_cpm}\right),\widetilde{\mathcal{O}}\left(\frac{\kappa\mu S\log^2 S}{K_cpn}\right)\right)$ is achievable. Now let us consider the decoding complexity. Note that the decoding procedure involves matrix-vector multiplications of inverse of Toeplitz matrix and inverse of confluent Cauchy-Vandermonde matrix. From the inverse formula of confluent Cauchy-Vandermonde matrix presented in [15], the matrix-vector multiplication of the inverse of confluent Cauchy-Vandermonde matrix $\hat{\mathbf{V}}_{\ell,K_c,R_E,R}$ can be decomposed into a series of structured matrix-vector multiplications including confluent Cauchy matrix, transpose of Vandermonde matrix, Hankel matrix and Toeplitz matrix. By fast algorithms [14], [16], the complexity of decoding is at most $\widetilde{\mathcal{O}}(\lambda\mu p\log^2 R)$. This completes the proof of Theorem 1.

VI. Conclusion

Inspired by the idea of Cross Subspace Alignment (CSA) which originated in secure private information retrieval (PIR) literature [7]–[9] and has recently found applications in secure and private distributed computing [9]–[11], we introduce a new class of codes, called GCSA codes, for coded distributed batch matrix multiplication (CDBMM). GCSA codes unify, generalize and improve upon state of art algorithms (EP codes [5], LCC codes [6]) for CDMM. GCSA codes, which include both LCC codes and EP codes as special cases, bridge the extremes of matrix partitioning based approaches (EP codes) and batch processing approaches (LCC codes), and allow a tradeoff between server computation complexity, which may be improved by emphasizing the matrix partitioning aspect, and communication costs, which may be improved by emphasizing the batch processing aspect. Even when no matrix partitioning is used, GCSA codes improve upon LCC codes in download-constrained settings due to cross-subspace alignment. Further code constructions based on CSA that generalize LCC codes for N-linear batch computations and multivariate polynomial evaluations, as well as extensions to include X-secure and B-byzantine settings are included in the full paper [13].

VII. Acknowledgement

This work is supported by funding from NSF grants CNS-1731384, CCF-1907053 and by ONR grant N00014-18-1-2057.

References

[1] V. Cadambe and P. Grover, "Codes for distributed computing: A tutorial," *IEEE ITSOC Newsletter*, vol. 67, no. 4, pp. 3–15, December 2017.

[2] 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.

[3] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the Optimal Recovery Threshold of Coded Matrix Multiplication," *arXiv preprint arXiv:1801.10292*, 2018.

[4] S. Dutta, Z. Bai, H. Jeong, T. Low, and P. Grover, "A Unified Coded Deep Neural Network Training Strategy Based on Generalized PolyDot Codes for Matrix Multiplication," *ArXiv:1811.1075*, Nov. 2018.

[5] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler Mitigation in Distributed Matrix Multiplication: Fundamental Limits and Optimal Coding," *ArXiv:1801.07487*, 2018.

[6] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange Coded Computing: Optimal Design for Resiliency, Security and Privacy," *ArXiv:1806.00939*, 2018.

[7] Z. Jia, H. Sun, and S. A. Jafar, "Cross Subspace Alignment and the Asymptotic Capacity of X -Secure T -Private Information Retrieval," *IEEE Trans. on Info. Theory*, vol. 65, no. 9, pp. 5783–5798, Sep. 2019.

[8] Z. Jia and S. A. Jafar, "On the Asymptotic Capacity of X-Secure T-Private Information Retrieval with Graph Based Replicated Storage," *ArXiv:1904.05906*, 2019.

[9] ——, "X-secure T-private Information Retrieval from MDS Coded Storage with Byzantine and Unresponsive Servers," *ArXiv:1908.10854*.

[10] J. Kakar, S. Ebadifar, and A. Sezgin, "On the Capacity and Straggler-Robustness of Distributed Secure Matrix Multiplication," *IEEE Access*, vol. 7, pp. 45 783–45 799, 2019.

[11] Z. Jia and S. Jafar, "On the Capacity of Secure Distributed Matrix Multiplication," *ArXiv:1908.06957*, 2019.

[12] M. Gasca, J. Martinez, and G. Mühlbach, "Computation of Rational Interpolants with Prescribed Poles," *Journal of Computational and Applied Mathematics*, vol. 26, no. 3, pp. 297–309, 1989.

[13] Z. Jia and S. Jafar, "Cross-Subspace Alignment Codes for Coded Distributed Batch Computation," *ArXiv:1909.13873*, 2019.

[14] V. Olshevsky and A. Shokrollahi, "A Superfast Algorithm for Confluent Rational Tangential Interpolation Problem via Matrix-Vector Multiplication for Confluent Cauchy-like Matrices," *Contemporary Mathematics*, vol. 280, pp. 31–46, 2001.

[15] Z.-H. Yang and Y.-J. Hu, "Displacement Structure Approach to Cauchy and Cauchy–Vandermonde Matrices: Inversion Formulas and Fast Algorithms," *Journal of Computational and Applied Mathematics*, vol. 138, no. 2, pp. 259–272, 2002.

[16] I. Gohberg and V. Olshevsky, "Fast Algorithms with Preprocessing for matrix-Vector multiplication Problems," *Journal of Complexity*, vol. 10, no. 4, pp. 411–427, 1994.

$$\hat{\mathbf{V}}_{\ell,K_c,R_E,R}\triangleq\left[\begin{array}{ccc:c:ccc:ccc}\frac{1}{(f_{1,1}-\alpha_1)^{R_E}}&\cdots&\frac{1}{f_{1,1}-\alpha_1}&\cdots&\frac{1}{(f_{\ell,K_c}-\alpha_1)^{R_E}}&\cdots&\frac{1}{f_{\ell,K_c}-\alpha_1}&1&\cdots&\alpha_1^{R-R_EL-1}\\ \frac{1}{(f_{1,1}-\alpha_2)^{R_E}}&\cdots&\frac{1}{f_{1,1}-\alpha_2}&\cdots&\frac{1}{(f_{\ell,K_c}-\alpha_2)^{R_E}}&\cdots&\frac{1}{f_{\ell,K_c}-\alpha_2}&1&\cdots&\alpha_2^{R-R_EL-1}\\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\ \frac{1}{(f_{1,1}-\alpha_R)^{R_E}}&\cdots&\frac{1}{f_{1,1}-\alpha_R}&\cdots&\frac{1}{(f_{\ell,K_c}-\alpha_R)^{R_E}}&\cdots&\frac{1}{f_{\ell,K_c}-\alpha_R}&1&\cdots&\alpha_R^{R-R_EL-1}\end{array}\right]\tag{31}$$