Compression-Informed Coded Computing

Michael Rudow[♮], **Neophytos Charalambides**[♯], **Alfred O. Hero III**[♯], **and K.V. Rashmi**[♮]
[♮]CS Department Carnegie Mellon University [♯]EECS Department University of Michigan
Email: mrudow@andrew.cmu.edu, neochara@umich.edu, hero@umich.edu, rvinayak@cs.cmu.edu

2023 IEEE International Symposium on Information Theory (ISIT) | 978-1-6654-7554-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISIT54713.2023.10206890

***Abstract*—Large-scale computations are ubiquitous and demand exorbitant resources, with matrix multiplication being a prominent example. Multiplying high-dimensional matrices is cumbersome for an individual server but is frequently needed in many applications. To alleviate the computational cost, one can take a low-rank approximation of the matrix product and distribute it over multiple workers. However, the tail latency of such distributed computations is degraded by straggling workers. One solution is to query extra workers with coded inputs to replace the outputs of straggling workers; this technique is called "coded computing." Nearly all existing coded computing schemes apply to multiplying *any* matrices. Instead, we propose a new framework to design coded computing schemes to take advantage of the structure induced by compression, which we call *compression-informed coded computing*. We then showcase the benefits of the framework in two steps. First, we illustrate how sketching can lead to linear dependencies in the matrices multiplied by the workers. Second, we apply locality-based coded computing to leverage these linear dependencies to make do with fewer workers compared to coded computing schemes that ignore the structure of the matrices being multiplied.**

I. Introduction

Source coding and channel coding are crucial components of information theory that date back to the seminal work of Claude Shannon [1]. Source coding is used to compress data or computation. Channel coding adds robustness to noise to ensure the fidelity of the original data. Shannon's source-channel separation theorem shows that there is no cost to first compressing the input and then adding redundancy under certain conditions. Several works (e.g., [2]–[4]) have further studied this topic. The separation theorem does not hold in many scenarios (e.g., for multiple users [4]), but separating source and channel coding retains great practical importance by simplifying the process of designing codes.

Inspired by the potential benefits of combining source and channel coding, we turn to a different domain where an analog of separate source and channel coding is currently the norm: large-scale distributed matrix multiplication. Naturally arising in signal processing, numerical analysis, machine learning, and network analysis, these matrix products are too burdensome to be computed directly by an individual server. To tame the computational complexity, *sketching* is used to reduce the size of the computation (source coding). The computation is then partitioned into smaller matrix products that are distributed over multiple workers. However, a few slow or otherwise unavailable workers, termed "stragglers," serve as a bottleneck leading to high tail latencies. The adverse effect of stragglers can be mitigated by adding redundancy to the computation (channel coding); this approach is called *coded computing*. Sketching and coded computing have been well-studied in disparate communities.

Sketching was introduced in [5] for dimension reduction and is a broadly applicable tool (see [6] for an overview of its applications). The most relevant use of sketching for our paper is approximate matrix multiplication [7], [8] through what is known as CR-Matrix Multiplication (CR-MM). Under CR-MM, to estimate a matrix product $\mathbf{A} * \mathbf{B}$, one can multiply randomly sampled pairs of columns of $\mathbf{A}$ and rows of $\mathbf{B}$. The technique has recently been shown to be amenable to coded computing in [9] by generalizing CR-MM to involve sampling pairs of block-submatrices of $\mathbf{A}$ and $\mathbf{B}$ to multiply rather than column-row pairs. Finally, coded computing is applied to provide straggler tolerance when using workers to multiply the sampled pairs of matrices.

Suppose that the number of pairs of matrices to be multiplied is k. These (smaller) matrix products are to be distributed over workers, where each worker can compute a single matrix product, and some workers are stragglers. Coded computation can be used to mitigate the negative impact of stragglers as follows. Queries are sent to w workers to each compute a single (coded) matrix product so that the outputs of the $(w-s)$ non-straggling workers suffice to complete the computation. For example, the Matdot Code [10] does so using $w_{k,s} = (2k-1+s)$ workers, which Yu et al. [11] showed is optimal for an arbitrary k matrix products.

In contrast, a recently introduced methodology [12] called *locality-based coded computing* requires fewer workers when the k pairs of matrices to be multiplied are linearly dependent. For example, an established complex matrix multiplication algorithm called the "3M method" [13] involves computing three linearly dependent real-valued matrix products. By exploiting these linear dependencies, the coded computing scheme from [12] makes do with only $2/3w_{k,s}$ workers. Similar savings may be possible when sketching is used by exploiting its structure for coded computing for matrix multiplication (and, more generally, for multivariate polynomials).

Our work introduces a framework for what we term *compression-informed coded computing* wherein the coded computing scheme can be tuned based on the sketching function (Section III). We show how to leverage two recent works [12], [14] to build coded computing schemes for (compressed) matrix multiplication that leverage the structure of the compression (Section IV). The proposed framework maintains a low complexity of code design while retaining the benefits of exploiting properties introduced by compression.

II. Background and Related Work

Recall that our work aims to improve the synergy between compression and coded computing. To do so, Section IV will connect the structure introduced by an example of sketching (inspired by [9]) to the toolkit of locality-based coded computation [12], [14]). The relevant background on sketching and locality-based coded computation is introduced in detail in Sections II-A and II-B, respectively. Then we discuss other related works in Section II-C.

A. CR-MM

Recall that [7], [8] presented a sketching technique called CR-MM for approximate matrix multiplication of matrices $\mathbf{A} \in \mathbb{F}^{L\times N}$ and $\mathbf{B} \in \mathbb{F}^{N\times M}$ for a field $\mathbb{F}$. CR-MM approximates $\mathcal{C} \approx \mathbf{A} * \mathbf{B}$ by randomly sampling and multiplying a subset of the columns of $\mathbf{A}$ and rows of $\mathbf{B}$. This approach was extended to the domain of distributed matrix multiplication by instead sampling q pairs comprising a block of columns of $\mathbf{A}$ and a block of rows of $\mathbf{B}$ in [9] for some $q|N$. The probability of sampling each pair of blocks is proportional to their Frobenius norms. Specifically, for $\tau := N/q$, $\mathbf{A}$ and $\mathbf{B}$ are partitioned into q equal-size submatrices as

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \cdots & \mathbf{A}_q\end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix}\mathbf{B}_1^T & \cdots & \mathbf{B}_q^T\end{bmatrix}^T. \quad (1)$$

Then $k < q$ pairs are sampled with replacement. Let $[n]$ denote $\{1,\ldots,n\}$, and $\{Y_i\}_{i\in[n]}$ denote the set of elements $\{Y_1,\ldots,Y_n\}$. Then for each $\iota \in [q]$, $(\mathbf{A}_{\{\iota\}}, \mathbf{B}_{\{\iota\}})$ is sampled with probability $p_\iota^{\mathbf{A},\mathbf{B}} \propto \|\mathbf{A}_{\{\iota\}}\|_F * \|\mathbf{B}_{\{\iota\}}\|_F$. Finally, $\mathcal{C}$ is taken to be a weighted sum of the sampled matrix products.

Later, in Section IV we generalize the variant of $CR-\text{MM}$ from [9] and show how the generalization leads to sampling matrices well-suited to the locality-based approach to coded computing, which is discussed next in Section II-B.

B. Coded computing for linear dependencies

Recall that most coded computing schemes for matrix multiplication use the same encoding and decoding functions to multiply *any* k pairs of matrices. Such coded computing schemes thus ignore any structure in the inputs and require $w_{k,s} = (2k-1+s)$ workers [10], [11]. However, compression may lead to multiplying k linearly dependent pairs of matrices. Such linear dependencies enable coded computing using workers by applying techniques from locality-based coded computation [12], [14].[1]

Next, we summarize the existing toolkit of locality-based coded computing that can be leveraged to design compression-informed coded computing schemes. In [12], Rudow et al. showed how to efficiently multiply any k linearly dependent pairs matrices, $\{(X_i^{(1)}, X_i^{(2)})\}_{i\in[k]}$. Specifically, suppose there are $\alpha_1,\ldots,\alpha_{k-1} \subseteq \mathbb{F}\setminus\{0\}$ such that $\sum_{i\in[k-1]} \alpha_i(X_i^{(1)}, X_i^{(2)}) = (X_k^{(1)}, X_k^{(2)})$. Then $(w_{k,s}-2)$ workers are needed. The methodology of [12] was extended in [14] to leverage linear dependencies in each coordinate of the input points (i.e., for $\{X_i^{(j)}\}_{i\in[k]}$ for $j \in \{1,2\}$).

[1]Computing a degree d polynomial, f, at any k input points requires $w_{k,s}$ [15]. Fewer workers are needed for linearly dependent input points [12].

C. Related work

One of our goals is to unlock coded computing schemes that exploit existing results from the well-studied literature on approximate matrix multiplication. Hence, we introduce a simple framework for designing new compression-informed coded computing schemes. In contrast, the few existing works combining compression and coded computing do not provide a simple means to leverage new sketching techniques to design compression-informed coded computing schemes. In [16], the authors propose approximate coded computing by leveraging sketching for each row-column block. The main drawback is the number of extra workers needed for straggler tolerance scales as the product of s times the number of row-column blocks. Recall from Section II-A that [9] involved compressing a matrix product into a weighted sum of column-row block pair products, and then using the MatDot Code [10] for straggler tolerance without exploiting the structure of the sketch. In [17], count-sketches are incorporated into the design of a variant of the improved Entangled Polynomial Code [11] to combine approximate matrix multiplication with straggler tolerance. The code approximates a matrix product by computing approximations of each row-column block whose accuracy in each position depends on the norm of the corresponding position of *all* blocks of the matrix product.

Finally, we discuss the background of coded computing. Coded computing was originally introduced in [18] for matrix multiplication. Coded computation has since been widely studied for matrix-vector products [19]–[22] and matrix multiplication [10], [11], [23]–[26]. More generally, coded computing has been studied for multivariate polynomial functions in [12], [14], [15]. Coded computing also has extensive applications in machine learning, such as for gradient descent [27]–[34] and for inference [35]–[37]. Coded computing is closely related to algorithm-based fault tolerance, which was introduced in [38] and later studied in [39]–[43]. The main difference is that algorithm-based fault tolerance adds resiliency to errors in matrix multiplication whereas coded computation primarily mitigates erasures in the form of stragglers.

III. Compression-informed coded computing

Recall that under most existing works, a large-scale computation is compressed (via sketching) and distributed over multiple workers. Then coded computing is applied to add redundancy. Most existing coded computing schemes apply for arbitrary inputs. This section introduces a framework connecting the two components to facilitate leveraging the structure of compression in designing coded computing schemes. An overview is shown in Figure 1 for multiplying matrices.

Compression. Evaluating a function, $f^{(*)}(\cdot)$, at an input X, is converted into evaluating a smaller function, $f(\cdot)$, at several inputs to create an estimate of $f^{(*)}(X)$ called Y. Let function used to select the inputs to $f(\cdot)$ be called a *compression function*, and the function mapping evaluations

of $f(\cdot)$ to Y be called the *reconstruction function*. The pair of compression and reconstruction functions, $c(\cdot)$ and $r(\cdot)$, must be chosen from admissible sets of compression and reconstruction functions, C and R, respectively. Limiting to $c \in \mathsf{C}$ and $r \in \mathsf{R}$ ensures the computational cost is acceptable. Let the pair $(c(\cdot), r(\cdot))$ be called *valid* if the combination of compression and then reconstruction results in a sufficiently accurate estimate of $f^{(*)}(X)$ in expectation. Formally, let the compression result in $c(f^{(*)}, X) = \{X_i\}_{i\in[k]}$. Then $(c(\cdot), r(\cdot))$ are an ϵ-*valid* for $\epsilon \in (0, 1)$ if

$$\mathbb{E}\Big[\|f^{(*)}(X) - r(\{f(X_i)\}_{i\in[k]})\|_F\Big] < \epsilon.$$

Coded computing. The input to the coded computing component is the function, $f(\cdot)$, the compression function, $c(\cdot)$, and the set of k inputs to $f(\cdot)$ resulting from the compression (i.e., $c(f^{(*)}, X)$). Then a set of inputs to $f(\cdot)$ are chosen for w workers to evaluate so that any $(w - s)$ of their outputs suffice to recover $f(\cdot)$ at the desired k input points. The inputs to $f(\cdot)$ must be chosen via an *encoding* function, and the technique for recovering $f(\cdot)$ at the desired input points is to use a *decoding* function. To ensure the procedure has an acceptable computational cost, the encoding and decoding functions, $e(\cdot)$ and $d(\cdot)$, must be taken from admissible sets of encoding and decoding functions, Enc and Dec, respectively. Let a pair (e, d) be called *valid* if when any s workers straggle, the evaluations of $f(\cdot)$ at the k desired points are obtained. Formally, $\{e(\cdot), d(\cdot)\}$ are valid if the points of f chosen to be evaluated, $e(c(f^{(*)}, X)) = \{\widetilde{X}_i\}_{i\in[w]}$, leads to any size $(w-s)$ subset $\mathcal{N} \subseteq [w]$ of the workers' outputs (i.e., $\{f(\widetilde{X}_i)\}_{i\in\mathcal{N}}$) sufficing to recover f evaluated at all k points dictated by compression. Formally,

$$d_{\mathcal{N}}(\{f(e(c(f^{(*)}, X)))_i\}_{i\in\mathcal{N}}) = \{f(X_i)\}_{i\in[k]}.$$

The goal is to minimize the number of workers used (i.e., w).

Compression-informed coded computing. Consider any function to be computed, $f^{(*)}$, at an input, X, smaller function, $f(\cdot)$, $\epsilon \in (0, 1)$, and number of stragglers, s. Then a quadruple of compression, reconstruction, encoding, and decoding functions is deemed *valid* if in expectation (over the randomness used in compression/reconstruction) having workers evaluate $f(\cdot)$ at the points chosen by compression then encoding, then decoding the evaluations at the compressed points, and, finally, applying reconstruction leads to a sufficiently accurate estimate of $f^{(*)}(X)$. Formally, $\{c(\cdot), r(\cdot), e(\cdot), d(\cdot)\}$ is deemed *valid* if for any $\mathcal{N} \subseteq [w]$ of size $(w - s)$,

$$\mathbb{E}\Big[\|f^{(*)}(X) - r(d_{\mathcal{N}}(\{f(e(c(f^{(*)}, X)))_i\}_{i\in[\mathcal{N}]}))\|_F\Big] < \epsilon.$$

The key idea is to apply coded computing to the specific compression of X rather than arbitrary inputs to f. This enables coded computing using fewer workers (Section IV).

Finally, we make a few observations. First, the choice of $\mathsf{C}, \mathsf{R}, \mathsf{Enc}, \mathsf{Dec}$ (a) dictate the computational load of encoding/decoding, (b) affect how many workers are needed, and (c) set the per-worker communication load. Second, it is straightforward to extend the model to handle b byzantine workers that return adversarial outputs instead of correct matrix products in addition to the s straggling workers. Third, the model allows exploiting the algorithms employed by workers to compute "f". One example is workers using the $3M$ method [13] to multiply two complex-valued matrices using three real-valued matrix products at linear combinations of the real and complex values of the two matrices. This can be modeled even more generally as workers computing a function f by evaluating a different function, f', multiple times. Then coded computing can be used to exploit not only the compressed inputs to f but also the (smaller) compressed inputs to f'.

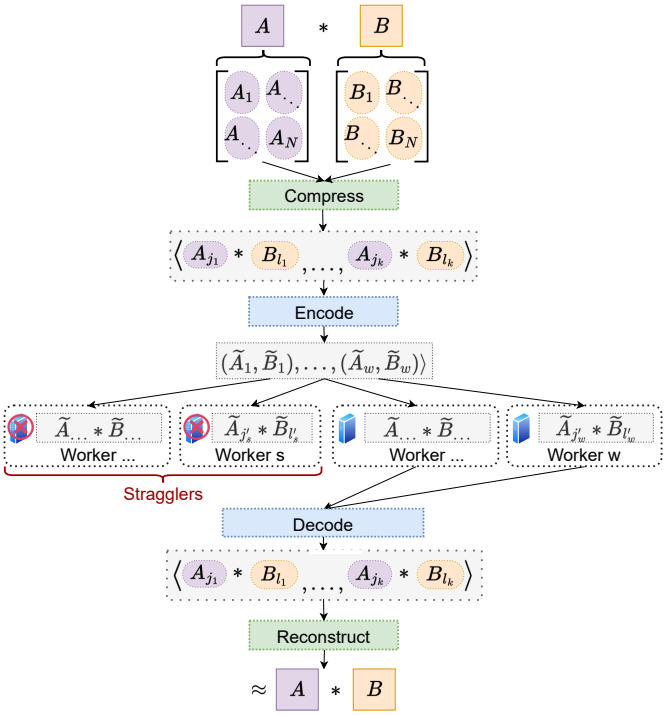

Fig. 1: Example of compression-informed coded computing to approximate a matrix product $f^{(*)}(\mathbf{A}, \mathbf{B}) = \mathbf{A}*\mathbf{B}$ by computing smaller matrix products $f(\cdot)$. Components of compression and coded computing are green and blue, respectively.

IV. Matrix Multiplication via Compression-Informed Coded Computing

Next, we illustrate the advantage of compression-informed coded computation for approximate matrix multiplication, although the results apply more generally to multivariate polynomial computations. First, Section IV-A showcases how to develop compression-informed coded computing schemes by illustrating (a) how a sketch can lead to multiplying linearly dependent pairs of matrices, and (b) how to leverage these linear dependencies to make do with fewer workers. Then Section IV-B lists other settings where compression-informed coded computing is likely to be well-suited.

A. Building a compression-informed coded computing scheme

This section highlights the benefits of compression-informed coded computing when approximately multiplying

large matrices $\mathbf{A}$ and $\mathbf{B}$ for one specific sketch (introduced below). Similar techniques could be applied more generally.

To provide an interface between sketching and coded computing under the proposed framework, it suffices for the coded computing to be given access to the linear dependencies in each coordinate induced by compression.[2]

Next, we formalize compression, reconstruction, encoding, and decoding. We consider the sets of projections and linear maps, C and R, as the sets of valid compression and reconstruction functions, respectively. This ensures a systematic source code with low complexity for reconstruction without considering stragglers. For coded computing, we consider linear encoding and decoding as being linear for any given set of stragglers; this restriction is common in the literature on coded computing to ensure the operations are computationally efficient. Formally, the set of encoding functions, Enc, is the set of linear maps of appropriate dimensions. The set of decoding functions, Dec, is a set of tuples of linear maps, where the ith position in the tuple corresponds to the map used for the ith subset s straggling workers.

We now introduce the sketch considered in this section. Recall from Section II-A that the variant of CR-MM from [9] may be computationally intensive depending on the workers' capabilities and the sizes of $\mathbf{A}$ and $\mathbf{B}$ (e.g., a wide $\mathbf{A}$ and a tall $\mathbf{B}$). We introduce a simple modification that further reduces the computational cost and makes the sketch more suitable for locality-based coded computing. Let us divide $\mathbf{A}$ and $\mathbf{B}$ into equal-size submatrices using the following *grid partitioning*. For $m|L$ and $n|M$, we define

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \dots & \mathbf{A}_{1,q} \\ \vdots & \dots & \vdots \\ \mathbf{A}_{m,1} & \dots & \mathbf{A}_{m,q} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \dots & \mathbf{B}_{1,n} \\ \vdots & \dots & \vdots \\ \mathbf{B}_{q,1} & \dots & \mathbf{B}_{q,n} \end{bmatrix} \tag{2}$$

$$\mathbf{A} * \mathbf{B} = \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \dots & \mathbf{C}_{1,n} \\ \vdots & \dots & \vdots \\ \mathbf{C}_{m,1} & \dots & \mathbf{C}_{m,n} \end{bmatrix}.$$

For any $i \in [m], j \in [n]$, let the block matrices corresponding to the ith block row of $\mathbf{A}$ and jth block column of $\mathbf{B}$ be:

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{A}_{i,1} & \cdots & \mathbf{A}_{i,q} \end{bmatrix} \quad \text{and} \quad \mathbf{B}_j = \begin{bmatrix} \mathbf{B}_{1,j}^T & \cdots & \mathbf{B}_{q,j}^T \end{bmatrix}^T,$$

leading to $\mathbf{C}_{i,j} = \mathbf{A}_i * \mathbf{B}_j$. One can apply the CR-MM scheme from [9] for each $i \in [m], j \in [n]$ to determine an estimate, $\mathcal{C}_{i,j}$ of $\mathbf{C}_{i,j}$; we call this approach "Grid CR-MM." For each $i \in [m], j \in [n]$, the number of pairs $\{\mathbf{A}_{i,\iota} * \mathbf{B}_{\iota,j}\}_{\iota \in [q]}$ that are sampled with replacement is called $k_{i,j}$. For any $\iota \in [q]$, $\mathbf{A}_{i,\iota} * \mathbf{B}_{\iota,j}$ is chosen with probability:

$$p_\iota^{\mathbf{A}_i, \mathbf{B}_j} = \frac{\|\mathbf{A}_{i,\iota}\|_F * \|\mathbf{B}_{\iota,j}\|_F}{\sum_{\iota \in [q]} \|\mathbf{A}_{i,\iota}\|_F * \|\mathbf{B}_{\iota,j}\|_F}.$$

Next, we show the correctness of Grid CR-MM.

[2]For multivariate polynomial computations, the equations of linear dependence of the k input points are likewise sufficient to use the tools from [14].

Lemma 1. *If Grid CR-MM outputs $\langle \mathcal{C}_{i,j} \mid i \in [m], j \in [n] \rangle$ to approximate $\langle \mathbf{C}_{i,j} = \mathbf{A}_i * \mathbf{B}_j \mid i \in [m], j \in [n] \rangle$ then for all $i \in [m], j \in [n]$ $\mathbb{E}\big[\|\mathbf{C}_{i,j} - \mathcal{C}_{i,j}\|_F^2\big] \leq \|\mathbf{A}_i\|_F^2 \|\mathbf{B}_j\|_F^2 / k_{i,j}$.*

Proof: Follows directly from [9] Theorem 2.1, by applying it to each $i \in [m], j \in [n]$. ∎

We note that the accuracy of the estimates of each $\mathcal{C}_{i,j}$ are easy to determine and depend only on $\|\mathbf{A}_i\|_F$ and $\|\mathbf{B}_j\|_F$. In contrast, under [17], the accuracy of estimating $\mathbf{C}_{i,j}$ depends on $\|\mathbf{A}_{i'}\|_F$ and $\|\mathbf{B}_{j'}\|_F$ for $i' \neq i, j' \neq j$; the guarantees on the accuracy also depend on the value of entries of $\mathbf{A} * \mathbf{B}$, which will not be known in advance. Hence, the number of samples needed under Lemma 1 (i.e., $\sum_{i \in [m], j \in [n]} k_{i,j}$) may be significantly smaller than under [17] to accurately approximate each $\mathbf{C}_{i,j}$ in terms of $\|\mathbf{A}_i\|_F$ and $\|\mathbf{B}_j\|_F$. For example, if there is a block row of $\mathbf{A}$ and block column of $\mathbf{B}$ whose norms are much larger than the other block rows and block columns.

Next, we apply tools from locality-based coded computing from [14] to leverage linear dependencies in each coordinate of the input points, $X_1, \dots, X_k$, sampled under this sketch. We consider a basis for each coordinate $j \in \{1, 2\}$ comprising λ_j basis elements, $\mathrm{B}_i^{(j)}$ for $i \in [\lambda_j]$. The basis for the first and second coordinates are subsets of $\{\mathbf{A}_{i,j}\}_{i \in [m], j \in [q]}$ and $\{\mathbf{B}_{j,l}\}_{j \in [q], l \in [n]}$, respectively, that comprise all sampled elements. Let the minimum number of times a basis element from coordinate j appears in the corresponding component of an input point be denoted as u_j. Formally,

$$u_j = \min_{l \in [\lambda_j]} \Big(\sum_{r \in [k]} \mathbb{1}[X_r^{(j)} = \mathrm{B}_l^{(j)}] \Big). \tag{3}$$

The result from [14] fits into our model as follows

Theorem 1 (Direct application of [14])**.** *For any k pairs of matrices $\{X_i\}_{i \in [k]}$, the number of workers needed for coded computing is at most*

$$w_{k,s} + 2 - \sum_{j \in \{1,2\}} u_j.$$

Next, to enable comparison, we discuss how well existing coded computing schemes work for approximately multiplying $\mathbf{A}$ by $\mathbf{B}$. Recall that $\mathbf{A}$ and $\mathbf{B}$ are partitioned into q disjoint submatrices consisting of τ columns and rows, respectively (Equation 1). As a reference for comparison, the exact product (without any compression) can be computed with Entangled Polynomial Codes [11] and $(mnq + q - 1 + s)$ non-stragglers. Let the compression from using Grid CR-MM involve a total of $k = \sum_{i \in [m], j \in [n]} k_{i,j}$ evaluations of f. Then, the MatDot Code [10] allows approximate coded computating with $w_{k,s}$ workers. However, depending on how frequently each block is sampled, it may be possible to make do with fewer workers by applying Theorem 1, as we show next.

For example, suppose for $i \in [m], j \in [n], k_{i,j} = k' \ll q/2$, and s is small. Suppose there is a set $X^\dagger$ of 8 samples whose pairs of matrices are limited to $\{\mathbf{A}_{i,1}\}_{i \in [4]}$ and $\{\mathbf{B}_{1,i}\}_{i \in [4]}$. Suppose the samples of $X^\dagger$ are chosen as shown in Figure 2. By Theorem 1, only $(13 + s)$ workers are needed to compute

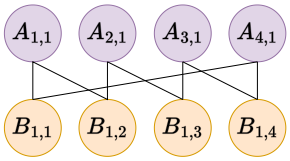

Fig. 2: Example of computing $\mathbf{AB}$ using Grid CR-MM restricted to the first 4 elements each of the first block column of $\mathbf{A}$ and block row of $\mathbf{B}$ with edges between sampled pairs.

$\mathbf{A}_{i,1}\mathbf{B}_{1,j}$ for each $(\mathbf{A}_{i,1}\mathbf{B}_{1,j}) \in X^{\dagger}$. Combining Theorem 1 with the Matdot Code on the remaining $(k' - 8)$ samples reduces the total number of workers compared to $w_{k,s}$ by $(3 - s)$ if $s < 3$. A similar procedure could be applied to the remaining $(k' - 8)$ pairs of matrix products. If $s = 2$ and every set of 8 samples has a structure like that of Figure 2, the overhead of coding reduced by at least 12.5%.

Next, we show that the sampling procedure of Grid CR-MM likely leads to Theorem 1 showing improvement over separate compression and coded computing.

Theorem 2. *Let* $\delta = \min_{i\in[\lambda_1],j\in[\lambda_2]}\left(p_{\iota}^{\mathbf{A}_i,\mathbf{B}_j}\right)$ *and* $r = \max\left(1/\delta, (m+n)p\right)$. *Then, if the number of samples is* $k' = \upsilon\gamma r\log(r)$ *for* $\upsilon \in \mathbb{Z}^+$ *and* $\gamma > 1$*:*

$$\mathbb{P}[u_1 \geq \upsilon, u_2 \geq \upsilon] \geq (1 - \upsilon \cdot 1/r^{\gamma-1}).$$

Proof: By the union bound, it suffices to show after k'/υ samples, for each $j \in \{1, 2\}$ and $i \in [\lambda_j]$, the probability that $\mathrm{B}_i^{(j)}$ is not sampled is at most $1/n^{\gamma-1}$. This can be viewed as sampling r elements where each is picked with probability at least $1/r$. The problem is an instance of the coupon collector problem, and the result follows from the tail bounds of the coupon collector problem. ∎

For concreteness, we note that when $s = 1$, Theorem 2 shows that with high probability sampling leads to Theorem 1 showing at most $(w_{k,s} - 2(\upsilon - 1))$ workers are needed.

Next, we show how many samples are needed to ensure that Theorem 1 applies to some subset of the samples.

Theorem 3. *For any* $k = (m + n)p$ *samples, there exists subsets of the basis for coordinates* 1 *and* 2*, respectively, and* $I \subseteq [k]$ *for which* $u_1 \geq 2$ *and* $u_2 \geq 2$.

Proof: Consider a graph with one row of vertices corresponding to the basis for the first coordinate and another corresponding to the basis of the second coordinate. Any sampled $(\mathrm{B}_i^{(1)}, \mathrm{B}_j^{(2)})$ is added as an edge to the graph.

Suppose there are c connected components comprising $v_1, \ldots, v_c$ vertices, respectively. If the graph is acyclic, the connected components are trees, and there are at most $\sum_{i=1}^{c}(v_i - 1) = ((\sum_{i=1}^{c} v_i) - c) = (v - c)$ edges. Since the graph has $(n+m) = v > (v-c)$ edges, it must be cyclic.

Consider any cycle $(\mathrm{B}_{l_1}^{(1)}, \mathrm{B}_{r_1}^{(2)}), \ldots, (\mathrm{B}_{l_t}^{(1)}, \mathrm{B}_{r_t}^{(2)})$ where for $i \in [t]$ $(\mathrm{B}_{l_i}^{(1)}, \mathrm{B}_{r_i}^{(2)})$ is sampled, $(\mathrm{B}_{l_1}^{(1)}, \mathrm{B}_{r_t}^{(2)})$ is sampled, $l_1, \ldots, l_t$ are distinct, and $r_1, \ldots, r_t$ are distinct. Then taking $\bigcup_{i\in[t]} \mathrm{B}_{l_i}^{(1)}$ and $\bigcup_{i\in[t]} \mathrm{B}_{r_i}^{(2)}$ concludes the proof. ∎

When $s = 1$, combining Theorem 3 with Theorem 1 shows that only $(w_{k,s} - 2)$ workers are needed. More generally, Theorems 2 and 3 show that tuning coded computing based on the properties of grid CR-MM enables using fewer workers than is possible when coded computing is applied without considering the structure used in compression.

B. Other applications of compression-informed coded computing

We have highlighted how to design a compression-informed coded computing scheme for approximate matrix multiplication. Next, we illustrate two other potential applications.

Sparse matrix multiplication. Suppose $X = (X^{(1)}, X^{(2)}) \sim \mathcal{D}$, for a distribution, $\mathcal{D}$, of pairs of sparse matrices. Then compression could be dividing $X^{(1)}$ and $X^{(2)}$ into grids of $m \times p$ and $p \times n$ equal-size submatrices (as in Equation 2) and considering all mnp pairwise products of nonzero pairs. The sparsity leads to many pairs where at least one element (thus, the product) is zero; so Theorem 1 is suitable.

Two-stage matrix multiplication. There are scenarios where a "two-stage" computation may be appropriate. First, the matrix multiplication is compressed and distributed over the workers. No coded computing is used in the first stage (e.g., if stragglers are unlikely to arise). Second, copies of the pairs of matrices that had been sent to stragglers are sent to new workers, this time using coded computing (to rein in the tail latency). The matrix products in the second stage can have linear dependencies that can be exploited using Theorem 1.

V. Conclusion and Future Work

To compute large-scale matrix products using unreliable workers, two separate existing techniques can be applied. Sketching reduces the computational load, and then coded computing adds redundancy to provide tolerance to failures. The two methods are typically used separately, as most coded computing schemes are designed to apply for any matrix products. In contrast, our work introduces an interface to tune the design of coded computing schemes based on the sketch. We showcase how this approach can improve computational efficiency by illustrating how sketching can introduce linear dependencies into the computation and how coded computing can exploit these linear dependencies to make do with fewer workers. Future work could expand our proposed interface for designing compression-informed coded computing schemes in three main ways. First, one could show a broader class of properties imposed by sketching/derandomizing sketching to ensure useful linear dependencies arise. Second, one could extend the toolkit for exploiting properties of the input data for coded computing schemes. Third, one could extend the techniques for approximate coded computing, such as from [44]–[46], to our framework.

Acknowledgment

This work was funded in part by the following grants: ARO W911NF-19-1026 and DE-NA0003921.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[2] S. Vembu, S. Verdu, and Y. Steinberg, "The source-channel separation theorem revisited," *" IEEE Trans. Inf. Theory*, vol. 41, no. 1, pp. 44–54, 1995.

[3] C. Tian, J. Chen, S. N. Diggavi, and S. Shamai, "Optimality and approximate optimality of source-channel separation in networks," *" IEEE Trans. Inf. Theory*, vol. 60, no. 2, pp. 904–918, 2013.

[4] T. Cover, A. E. Gamal, and M. Salehi, "Multiple access channels with arbitrarily correlated sources," *" IEEE Trans. Inf. Theory*, vol. 26, no. 6, pp. 648–657, 1980.

[5] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," 1984.

[6] D. P. Woodruff *et al.*, "Sketching as a tool for numerical linear algebra," *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.

[7] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast Monte Carlo algorithms for matrices I: Approximating Matrix Multiplication," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.

[8] T. Sarlos, "Improved approximation algorithms for large matrices via random projections," in *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE, 2006, pp. 143–152.

[9] N. Charalambides, M. Pilanci, and A. O. Hero, "Approximate Weighted CR Coded Matrix Multiplication," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5095–5099.

[10] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *" IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, 2019.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *" IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.

[12] M. Rudow, K. Rashmi, and V. Guruswami, "A locality-based lens for coded computation," in *2021 IEEE Int. Symp. Inf. Theory (ISIT)*, 2021, pp. 1070–1075.

[13] N. J. Higham, "Stability of a method for multiplying complex matrices with three real matrix multiplications," *SIAM journal on matrix analysis and applications*, vol. 13, no. 3, pp. 681–687, 1992.

[14] M. Rudow, V. Guruswami, and K. Rashmi, "On expanding the toolkit of locality-based coded computation to the coordinates of inputs," in *2023 IEEE Int. Symp. Inf. Theory (ISIT)*, 2023, p. to appear.

[15] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1215–1225.

[16] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "Oversketch: Approximate matrix multiplication for the cloud," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 298–304.

[17] T. Jahani-Nezhad and M. A. Maddah-Ali, "CodedSketch: A coding scheme for distributed computation of approximated matrix multiplication," *" IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 4185–4196, 2021.

[18] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *" IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[19] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–40, 2019.

[20] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2019, pp. 1777–1781.

[21] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2018, pp. 1585–1589.

[22] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2018, pp. 1620–1624.

[23] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4403–4413.

[24] Q. Yu and A. S. Avestimehr, "Entangled polynomial codes for secure, private, and batch distributed matrix multiplication: Breaking the"cubic"barrier," *arXiv preprint arXiv:2001.05101*, 2020.

[25] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch computation," *IEEE Trans. Inf. Theory*, vol. 67, no. 5, pp. 2821–2846, 2021.

[26] H. Park, K. Lee, J.-y. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *2018 IEEE Int. Symp. Inf. Theory (ISIT)*. IEEE, 2018, pp. 1630–1634.

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

[28] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using Reed-Solomon codes," in *2018 IEEE Int. Symp. Inf. Theory (ISIT)*. IEEE, 2018, pp. 2027–2031.

[29] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[30] N. Charalambides, H. Mahdavifar, and A. O. Hero, "Numerically Stable Binary Gradient Coding," in *2020 IEEE Int. Symp. Inf. Theory (ISIT)*, 2020, pp. 2622–2627.

[31] N. Charalambides, H. Mahdavifar, M. Pilanci, and A. O. Hero, "Orthonormal Sketches for Secure Coded Regression," in *2022 IEEE Int. Symp. Inf. Theory (ISIT)*. IEEE, 2022, pp. 826–831.

[32] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," pp. 5610–5619, 2018.

[33] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," *IEEE Trans. Inf. Theory*, vol. 66, no. 12, pp. 7475–7489, 2020.

[34] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5434–5442, 2017.

[35] J. Kosaian, K. V. Rashmi, and S. Venkataraman, "Learning-based coded computation," *IEEE J. Sel. Areas Inf. Theory*, 2020.

[36] J. Kosaian, K. Rashmi, and S. Venkataraman, "Parity models: erasure-coded resilience for prediction serving systems," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 30–46.

[37] K. G. Narra, Z. Lin, G. Ananthanarayanan, S. Avestimehr, and M. Annavaram, "Collage inference: Using coded redundancy for lowering latency variation in distributed image classification systems," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, December 2020.

[38] Kuang-Hua Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518–528, 1984.

[39] T. Herault and Y. Robert, *Fault-tolerance techniques for high-performance computing*. Springer, 2015.

[40] Jing-Yang 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, pp. 732–741, 1986.

[41] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 410–416, 2009.

[42] C. J. Anfinson and F. T. Luk, "A linear algebraic model of algorithm-based fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1599–1604, 1988.

[43] J. Kosaian and K. Rashmi, "Arithmetic-intensity-guided fault tolerance for neural network inference on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[44] H. Jeong, A. Devulapalli, V. R. Cadambe, and F. P. Calmon, "ϵ-Approximate Coded Matrix Multiplication Is Nearly Twice as Efficient as Exact Multiplication," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 845–854, 2021.

[45] S. Kiani and S. C. Draper, "Successive approximation coding for distributed matrix multiplication," *IEEE Journal on Selected Areas in Information Theory*, vol. 3, no. 2, pp. 286–305, 2022.

[46] T. Jahani-Nezhad and M. A. Maddah-Ali, "Berrut approximated coded computing: Straggler resistance beyond polynomial computing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.