On expanding the toolkit of locality-based coded computation to the coordinates of inputs

Michael Rudow[†], **Venkatesan Guruswami**[*], **and K.V. Rashmi**[†]
[†] Carnegie Mellon University [*] University of California, Berkeley

2023 IEEE International Symposium on Information Theory (ISIT) | 978-1-6654-7554-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISIT54713.2023.10206923

***Abstract*—The tail latency of large-scale distributed computations, such as matrix multiplication, is adversely affected by unavailable workers. A technique called "coded computation" alleviates this problem by using extra workers to evaluate the function being computed at coded inputs and substitute the extra workers for the unavailable ones. Most of the literature on coded computation of multivariate polynomials applies to arbitrary inputs and ignores the structure of the inputs. However, a recent work introduced a locality-based coded computation framework and showed how to leverage the structure of inputs to reduce the overhead of coded computation. Our work expands the toolkit of locality-based approaches to coded computation beyond linearly dependent input points for the class of m-homogeneous polynomials. Specifically, we present new methods to exploit the structure of each coordinate of the inputs. Finally, we apply our new tools to multiplying upper (or lower) triangular matrices and show a reduction in the number of workers needed compared to the best known coded computation schemes.**

I. Introduction

Large-scale distributed computation is ubiquitous, with applications ranging from machine learning to scientific computing. However, these computations often suffer from high latency due to server unavailabilities [1] from slow or failing workers, which are termed "stragglers." To mitigate the impact of s stragglers, one may employ $(s+1)$-wise replication. The factor s overhead motivates a less costly approach to provide robustness to stragglers: coded computation.

First introduced in [2], coded computation involves encoding inputs and using extra workers to replace the stragglers (i.e., removing the bottleneck caused by stragglers to rein in the tail latency). The model of coded computation is shown in Figure 1. An encoder receives k input points $\{X_1, \ldots, X_k\}$ where each input point consists of m coordinates. The encoder queries w workers to each evaluate the function being computed at one point. The number of workers that could straggle is denoted by s. A decoder uses $(w-s)$ non-straggling workers' outputs to determine the value of the function at the k input points. Coded computation has been studied for multilinear functions [3]–[20], such as matrix multiplication, and more generally for polynomial functions [21], [22].

Most existing works on coded computation consider arbitrary inputs and do not exploit the structure of the inputs. The structure can be useful in many applications. For example, coded computation is often used after breaking down a large computation into multiple smaller computations to be completed in parallel. Methods of so-partitioning the larger computation can induce useful structures, such as linear dependencies. Linear dependencies in the inputs were leveraged in [22] via a notion of locality of codes to reduce the overhead of coded computing. This is termed "locality-based coded computation." Exploiting the linear dependencies when workers use the "3M method" [23] for complex matrix multiplication enables 33.3% fewer workers than state-of-the-art alternatives [22]. The benefits arise from unlocking better synergy between the computations completed by the workers and coded computation, motivating the need to discover more such techniques.

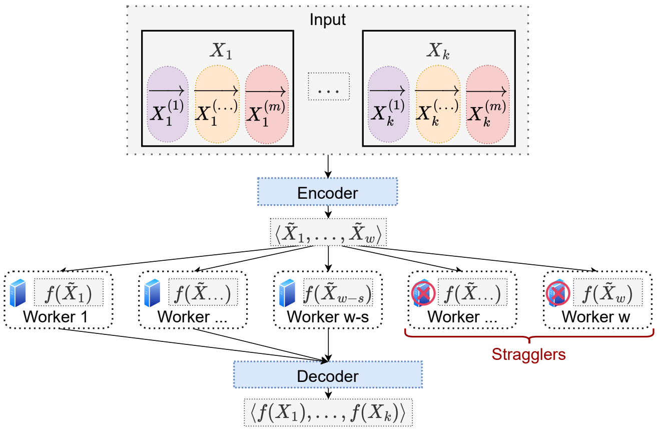
Input
X_1
$\overrightarrow{X_1^{(1)}}$
$\overrightarrow{X_1^{(\ldots)}}$
$\overrightarrow{X_1^{(m)}}$
$\ldots$
X_k
$\overrightarrow{X_k^{(1)}}$
$\overrightarrow{X_k^{(\ldots)}}$
$\overrightarrow{X_k^{(m)}}$
Encoder
$\langle \tilde{X}_1, \ldots, \tilde{X}_w \rangle$
$f(\tilde{X}_1)$
Worker 1
$f(\tilde{X}\ldots)$
Worker ...
$f(\tilde{X}_{w-s})$
Worker w-s
$f(\tilde{X}\ldots)$
Worker ...
$f(\tilde{X}_w)$
Worker w
Stragglers
Decoder
$\langle f(X_1), \ldots, f(X_k) \rangle$

Fig. 1. A high-level overview of coded computation. An encoder/decoder evaluates f at k input points, each comprising m coordinates, by querying w workers and using the outputs of any $(w-s)$ non-straggling workers.

Building on [22], we extend the toolkit for leveraging the structure of inputs to design coded computation schemes for m-homogeneous polynomials. An $m-$homogeneous polynomial is the product of m polynomials, each of which is a linear combination of monomials of the same degree; for example, the matrix-matrix product is a $2-$homogeneous polynomial. We first introduce methods to exploit up to one linear dependency in each *coordinate* of the input points (potentially different dependency for each coordinate). We then introduce a graph to model multiple linear dependencies within each coordinate and then exploit the dependencies for further savings in the number of works. The approach is useful for computations like matrix multiplication. For instance, applying the new methodology to multiply upper (or lower) triangular matrices leads to a lower overhead in the number of workers than the best known alternatives, with negligible computational cost.

II. Setting and Background

Next, we formalize the model and discuss related works.

A. Setting

Our model of coded computation draws from common notation from the literature, such as [21], [22]. An encoder/decoder receives k distinct inputs $\langle X_1, \ldots, X_k \rangle$ with entries from a field, $\mathbb{F}$, where each input point, X_i, comprises m coordinates, $\left\langle X_i^{(1)}, \ldots, X_i^{(m)} \right\rangle$. The goal is to evaluate a multivariate polynomial function, f, at the k input points using w workers. To do so, the ith worker is queried with the coded input $\widetilde{X_i}$ to compute $f(\widetilde{X_i})$. Up to s arbitrary workers are stragglers, and the remaining $(w-s)$ workers return their evaluations of $f(\cdot)$ to a decoder. The decoder then uses the returned evaluations of $f(\cdot)$ to determine $\langle f(X_1), \ldots, f(X_k) \rangle$. The encoder/decoder does not know f but has access to its degree.[1]

The goal is to use as few workers as possible, which is formally defined in [22] as follows

Definition 1 (Worker threshold)**.** *The worker threshold, $w_{d,\langle X_1,\ldots,X_k\rangle,s}$, is the smallest non-negative integer so that for any degree d multivariate polynomial function, f, coded computation of $\langle f(X_1), \ldots, f(X_k) \rangle$ with robustness to s stragglers is possible.*

Our work specifically considers $m-$homogeneous polynomials. A homogeneous polynomial is a linear combination of monomials of the same degree. A polynomial, $f(X)$, is $m-$homogeneous if it is a product of m polynomials of degree $d_1, \ldots, d_m$ over its input's m coordinates, $\left\langle X^{(1)}, \ldots, X^{(m)} \right\rangle$. Thus, the set of functions consider in our work is

$$span\left(\left\{\prod_{j\in[m]} h_j(X^{(j)})\right\}\right) \tag{1}$$

where each h_j is a degree $d_j \geq 1$ homogeneous polynomial. The total degree, d, of the polynomial is defined as $d = \sum_{j=1}^{m} d_j$.[2] One key property is that for all $\alpha \in \mathbb{F}$, and $j \in [m]$ $f(X^{(1)}, \ldots, \alpha X^{(j)}, \ldots, X^{(m)}) = \alpha^{d_j} f(X^{(1)}, \ldots, X^{(m)})$.

Finally, we present some notation. We denote $\{1, \ldots, n\}$ as $[n]$ for any positive integer n, and the size of $\mathbb{F}$ as $|\mathbb{F}|$.

B. Background

Coded computation is closely related to algorithm-based fault tolerance. Introduced in [24] and studied further in [25]–[28], algorithm-based fault tolerance provides robustness to errors in matrix-multiplication. In contrast, coded computation was designed to address stragglers, which can be viewed as erasures rather than errors. Coded computation was introduced in [2]. It has since been studied in several settings, including gradients for gradient descent [29]–[36], neural networks for inference [37]–[40], matrix-vector products [3]–[5], [8], [9], [20], matrix-matrix products [10]–[15], [17]–[19], [41], [42], and multivariate polynomial computations [21], [22].

A coded computing scheme was introduced in [21] for degree d multivariate polynomials using $w_{k,s} = \min(k(s+1), (k-1)deg(f)+s+1)$ workers for a model where the same linear encoding and decoding schemes apply to any k possible input points. That is, any structure in input points is not used. Such an approach is termed *input-oblivious* [22]. It was shown in [21] that the worker threshold of an input-oblivious approach is $w_{k,s}$.

Nearly all existing coded computation schemes use input-oblivious approaches. However, several applications induce useable structure to input points. For example, the matrix multiplication algorithms employed by the workers can introduce useful dependency structures. The lens of locality [22] provides a coded-computation framework to exploit these properties. Specifically, the authors presented a systematic scheme tolerating s stragglers for any multivariate polynomial f at k linearly dependent input points using $\big((k-2)deg(f) + s + 1\big)$ workers. They applied the scheme to complex matrix multiplication when workers use the 3M method [23] (i.e., multiply linearly dependent pairs of real-valued matrices) and reduced the worker threshold by up to 33.3% compared to the best possible under input-oblivious approaches. The approach involves defining an appropriate code for the function being computed and mapping the worker threshold for coded-computation of the function to a local decodability property of the code called "computational locality" [22].

III. Exploiting dependency in each coordinate

Recall from Section II-B that there is a coded computation scheme from [22] for any homogeneous polynomial, f', that leveraged linear dependencies in input points to use fewer workers than is possible under input-oblivious approaches. To do so, the authors introduced a polynomial $p(z)$ passing through nonzero multiples of the k input points and interpolated $f'(p(z))$, which yields $f'(\cdot)$ at the k input points. Our work introduces a simple extension of this work to leverage linear dependencies in each coordinate of the input points when computing an m-homogeneous polynomial, f. This is especially useful in scenarios where there are different equations of linear dependencies across the coordinates of the input points or the input points are linearly independent in certain coordinates, rendering the approaches from [22] inapplicable.

To do so, we first term any polynomial, $p(z)$, from $\mathbb{F}$ to the domain of f as a *curve*, and, for any $j \in [m]$, we denote its jth coordinate $p^{(j)}(z)$. Our methodology is to interpolate $f(p(z))$ for a curve, $p(z)$, that passes through nonzero multiples of the coordinates of the input points. When the k input points are linearly dependant in a coordinate, the degree of $p(z)$ in that coordinate is only $(k-2)$ instead of $(k-1)$, so fewer evaluations are needed to interpolate $f(p(z))$. This leads to a reduction in the worker threshold for coded computation. We define $p(z) = \langle p^{(1)}(z), \ldots, p^{(m)}(z) \rangle$ so that for any $i \in [k], j \in [m]$, p's jth coordinate passes through a nonzero multiple of $X_i^{(j)}$. Next, Theorem 1 establishes the computational locality of interpolating $f(p(z))$.

[1]The stipulation prevents the encoder/decoder from completing the entire computation, which would not be computationally feasible.

[2]The encoder/decoder has access to $d_1, \ldots, d_m$.

Theorem 1. *If there is a curve $p(z)$ so that for $i \in [k], j \in [m], \beta_1, \dots, \beta_k \in \mathbb{F}$, and $\eta_{1,1}, \dots, \eta_{1,m}, \dots, \eta_{k,1}, \dots, \eta_{k,m} \in \mathbb{F} \setminus \{0\}$, $p(\beta_i) = \left\langle \eta_{i,j} X_i^{(j)} \mid j \in [m] \right\rangle$, then $w_{d_1,\dots,d_m,\langle X_1,\dots,X_k\rangle,s} \leq w = \left(s + 1 + \sum_{j\in[m]} d_j deg(p_j(z))\right)$ whenever $|\mathbb{F}| \geq w$.*

Proof: The proof is shown in [43]. ■

Next, Corollary 1 applies Theorem 1 to *linear dependencies in each coordinate* to reduce the worker threshold compared to the lower bound for input-oblivious approaches ($w_{k,s}$).

Corollary 1. *If there is a $J \subseteq [m]$ so that $\forall j \in J$ there exists $\eta_{1,j}, \dots, \eta_{k-1,j} \in \mathbb{F} \setminus \{0\}$ for which $\sum_{i=1}^{k-1} \eta_{i,j} X_i^{(j)} = X_k^{(j)}$, then $w_{d_1,\dots,d_m,\langle X_1,\dots,X_k\rangle,s} \leq \left(w_{k,s} - \sum_{j\in J} d_j\right)$ whenever $|\mathbb{F}| \geq \left(w_{k,s} - \sum_{j\in J} d_j\right)$.*

Proof: The proof is presented [43]. ■

Theorem 1 and Corollary 1 extend to vector functions comprising products of homogeneous polynomials in each coordinate by applying them coordinate-wise. Note that the results take advantage of *a single* linear dependency in each coordinate.

IV. Computational Graph

Section III showed how to exploit a single linear dependency in each coordinate to reduce the worker threshold. This section shows how to exploit *multiple* linear dependencies in any coordinate(s) to reduce the worker threshold further. Specifically, we illustrate how to represent the linear dependencies with a graph and then leverage certain properties of the graph.

The key step is to break down each coordinate of the input points into basis elements. Specifically, for input points $\{X_1, \dots, X_k\}$ and coordinate $j \in [m]$, we define $X^{(j)} = \{X_1^{(j)}, \dots, X_k^{(j)}\}$ and $\mathrm{B}_1^{(j)}, \dots, \mathrm{B}_{\lambda_j}^{(j)}$ as a basis of $X^{(j)}$.

Next, we observe that λ_j may be much smaller than k for many computations. For example, consider computing $U * Z$ for matrices U and Z by using the divide and conquer approach of partitioning U into g block rows, $U_1, \dots, U_g$, and Z into g block columns, $Z_1, \dots, Z_g$. Then $U * Z$ follows from computing $f(U_i, Z_j) = U_i * Z_j$ for all $i, j \in [g]$ (where U_i is the first coordinate and Z_j is the second). Consequently, the $k = g^2$ input points are $\left\langle (U_i, Z_j) \middle| i, j \in [g] \right\rangle$. However, $\{U_1, \dots, U_g\}$ is a basis for the first coordinate and $\{Z_1, \dots, Z_g\}$ is a basis for the second coordinate. As such, the sizes of the basis for each component is $\lambda_1 = \lambda_2 = g = \sqrt{k}$. Because each basis element occurs in g input points, the input points contain many linear dependencies.

Breaking down each coordinate into basis elements leads to a natural graphical representation of the input points, which we call "computational graph," shown in Figure 2. The computational graph is undirected and comprises $2m$ rows. For each $j \in [m]$, there are λ_j nodes to reflect the basis elements of the jth coordinate; the nodes are labeled $\mathrm{B}_i^{(j)}$ for $i \in [\lambda_j]$. For each $l \in [k]$, we decompose X_l coordinate-wise into a linear combination of basis elements

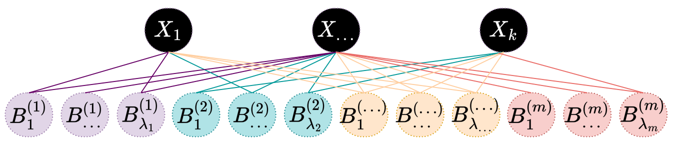

Fig. 2. Overview of the computational graph. For any $j \in [m], i \in [\lambda_j]$, $\mathrm{B}_i^{(j)}$ reflects the ith basis element of coordinate j. For each $i \in [k]$, X_i reflects the ith input point. For each coordinate $j \in [m]$, X_i has edges to the smallest set of basis elements whose span contains $X_i^{(j)}$.

as $X_l = \langle \sum_{i=1}^{\lambda_j} \alpha_{l,i,j} \mathrm{B}_i^{(j)} \mid j \in [m] \rangle$ where each $\alpha_{l,i,j} \in \mathbb{F}$. A node X_l is added and connected with an edge to each $\mathrm{B}_i^{(j)}$ for $j \in [m], i \in [\lambda_j]$ such that $\alpha_{l,i,j} \neq 0$; in other words, X_l is connected to all basis elements it comprises. We note that for each $l \in [k]$ and $j \in [m]$, X_l is connected to at least one of $\mathrm{B}_1^{(j)}, \dots, \mathrm{B}_{\lambda_j}^{(j)}$ if and only if $X_l^{(j)}$ is nonzero.

1) Exploiting dependency via the computational graph: Next, we discuss how to exploit the structure of the input points. To do so, we introduce the following term to group input points such that multiple (exploitable) linear dependencies exist in each coordinate.

Definition 2 (Multi-connected component)**.** *A* multi-connected component *is a connected component of the computational graphrestricted to a nonempty set of coordinates $J \subseteq [m]$ that satisfies all of the following conditions:*

1) *For any $j \in J, i \in [\lambda_j]$ $\mathrm{B}_i^{(j)}$ is only included if and only if it is connected to at least two input points of the connected component (i.e., $deg(\mathrm{B}_i^{(j)}) \geq 2$).*
2) *For any $l \in [k]$, the connected component contains all or none of both X_l and its edges in the coordinates of J.*
3) *For any $l \in [k]$, X_l is included only if for all $j \in J$ there is at most one basis element of coordinate j that X_l is connected to (i.e., $\sum_{i=1}^{\lambda_j} \mathbb{1}[\alpha_{l,i,j} \neq 0] = 1$).*

For conciseness, when the multi-connected component is over all coordinates, the term J may be omitted. We note that one consequence of the final condition is to prune out input points that are 0 in one or more coordinates because the image of such points under any $m-$multihomogeneous polynomial is always 0.

For concreteness, we illustrate a multi-connected component for the earlier example of matrix multiplication that involved computing $f(\cdot)$ at input points $\left\langle U_i, Z_j \middle| i, j \in [g] \right\rangle$. For simplicity, we consider $g = 2$, leading to $\{\mathrm{B}_1^{(1)}, \mathrm{B}_2^{(1)}\} = \{U_1, U_2\}$ and $\{\mathrm{B}_1^{(2)}, \mathrm{B}_2^{(2)}\} = \{Z_1, Z_2\}$ being the basis for first and second components, respectively. We can write the input points in terms of the basis elements as $\left\langle (\mathrm{B}_1^{(1)}, \mathrm{B}_1^{(2)}), (\mathrm{B}_1^{(1)}, \mathrm{B}_2^{(2)}), (\mathrm{B}_2^{(1)}, \mathrm{B}_1^{(2)}), (\mathrm{B}_2^{(1)}, \mathrm{B}_2^{(2)}) \right\rangle$. The overall graph satisfies the three conditions of being a multi-connected component. Specifically, each basis element is connected to two input points, each input point is connected only to basis elements, and each input point is connected to exactly one basis element in each coordinate. Hence, the graph is a single multi-connected component.

For any multi-connected component, we can exploit multiple linear dependencies in each coordinate by passing a curve through *nonzero multiples* of its basis elements; relaxing to nonzero multiples of basis elements reduces the degree of the curve compared to interpolating a polynomial through the input points (i.e., the best known approach in the input-oblivious setting). The technique also applies to multiple multi-connected components. Thus, we term a *disjoint union of multi-connected components* as the union of multi-connected components where each input point belongs to exactly one multi-connected component. The result is formalized in Theorem 2.

Theorem 2. *If the input points form a disjoint union of multi-connected components over coordinates* $J \subseteq [m]$ *then* $w_{d_1,\ldots,d_m,\langle X_1,\ldots,X_k\rangle,s} \leq w = \Big(s+1+\sum_{j\in J} d_j\big(k-\min_{i\in[\lambda_j]} deg(\mathrm{B}_i^{(j)})\big)\Big)$ *whenever* $|\mathbb{F}| \geq w$.

Proof: The proof is shown in [43]. ■

Thus, Theorem 2 leads to savings of $\sum_{j\in J} d_j\Big(\big(\min_{i\in[\lambda_j]} deg(\mathrm{B}_i^{(j)})\big)-1\Big)$ workers compared to the input-oblivious setting.

We provide three observations about Theorem 2. First, in certain scenarios, only some of the input points will be linearly dependent while the rest are not. Then the coded computing scheme from the proof of Theorem 2 could be applied for the linearly dependent input points and the best known coded computation scheme from the input-oblivious setting from [21] could be applied to the remaining ones. Second, when there are disjoint union of multi-connected components, applying Theorem 2 to each multi-connected component separately may lead to more savings in the number of workers; for example, if most basis elements of a coordinate have a high degree but one has a small degree, it will dominate the $\min$ function of just one application of the theorem. Whether to apply the theorem multiple times depends on the linear dependencies and parameters (e.g., d and s). Third, the coded computing scheme presented in the proof of Theorem 2 involves evaluating f at w points where many pairs of input points have the same value in a coordinate. This may allow for a reduced communication cost compared to evaluating f at arbitrary points if the model of coded computation were extended; for example, if workers each complete multiple evaluations of f.

Recall that one restriction of Theorem 2 is that it required each coordinate of each input point to be a multiple of a *single* basis element (property 3). Next, we show how to extend Theorem 2 to the following more general notion of a multi-connected component that allows some coordinates of input points to be linear combinations of multiple basis elements.

Definition 3 (Extended multi-connected component)**.** *An* extended multi-connected component *is a connected component of the computational graph restricted to a nonempty set of coordinates* $J \subseteq [m]$ *satisfying Definition 2 with property 3 replaced with the following: For any* $l \in [k]$, X_l *is included only if for all* $j \in J$ *one of the below two properties are satisfied*

1) *There is at most one basis element of coordinate* j *that* X_l *is connected to (i.e.,* $\sum_{i=1}^{\lambda_j} \mathbb{1}[\alpha_{l,i,j} \neq 0] = 1$*).*
2) *For all* $i \in [\lambda_j]$ *that* X_i *is connected to, for any* $l' \in [k] \setminus \{l\}$ *where* $X_{l'}$ *is connected to* $\mathrm{B}_i^{(j)}$, $X_{l'}$ *is not connected to any other basis elements of the* j*th coordinate; formally,*

$$\forall l' \in [k] \setminus \{l\}, \alpha_{l',i,j} \neq 0 \rightarrow \sum_{r\in[\lambda_j]\setminus\{i\}} \mathbb{1}[\alpha_{l',r,j} \neq 0] = 0. \tag{2}$$

Remark 1. *Theorem 2 also applies to any disjoint union of extended multi-connected components rather than a disjoint union of multi-connected components.*

Proof: The proof is shown in [43]. ■

Finally, we connect the notion of a disjoint union of multi-connected components to unions of cycles in the computational graph for the practically-motivated case of $m = 2$ (e.g., for matrix multiplication).

Lemma 1. *For* $m = 2$*, if for any* $l \in [k], j \in [m]$ $\sum_{i=1}^{\lambda_j} \mathbb{1}[\alpha_{l,i,j} \neq 0] = 1$*, a union of cycles in the computational graph is a disjoint union of multi-connected components.*

Proof: The proof is shown in [43]. ■

2) Algorithm to identify multi-connected components: Section IV-1 showed how to exploit linear dependencies in coordinates of the input points via the properties of each multi-connected component of the computational graph. In settings where Theorem 2 applies to the set of all k input points, the proof of Theorem 2 provides an explicit construction that can be used with no extra computational cost compared to state-of-the-art alternatives. Otherwise, realizing the benefits of Theorem 2 may require partitioning the input points into sets corresponding to multi-connected components. Section IV-2 presents computationally efficient techniques to identify multi-connected components.

Algorithms to find connected components are insufficient to find a disjoint union of multi-connected components because connected components may include basis elements of degree 1. In this section, we present an algorithm to find a disjoint union of multi-connected components.

Before defining the algorithm, we need some notation. Let the computational graph be $G = (V, E)$ where V is the set of vertices, and E is the set of edges. The computational graph has $v = \big(k+\sum_{j\in[m]} \lambda_j\big)$ vertices (i.e., for each $l \in [k]$ one for X_l and for each $j \in [m], i \in [\lambda_j]$ and one for $\mathrm{B}_i^{(j)}$). The graph has $e \leq k\sum_{j\in[m]} \lambda_j$ edges (i.e., the product of the number of input points and number of basis elements).

The algorithm applies to graphs that satisfy the following condition: **Condition 1**: Each input point X_l has exactly one neighbor among the basis elements of each coordinate. We note that a graph satisfying Condition 1 has at most $e = km$ edges (since the at most k input points are connected to at most one basis element in each of the m coordinates).

Next, we introduce Algorithm 1 to identify a disjoint union of multi-connected components where each multi-connected component has edge connectivity at least 2 (i.e., remains connected after removing any one edge) as follows. It identifies a set of connected components, and then iteratively removes each basis element with degree zero or one along with its neighbor and neighbor's edges (if degree one) until no such basis elements remain.

Algorithm 1 Algorithm to identify a disjoint union of multi-connected components of edge connectivity at least 2 of any graph satisfying Condition 1 if one exists or returns $\emptyset$.

```
Input: $G$.
Use BFS on $G$ to identify a set of connected components, $C$.
For each $c \in C$:
  While $\exists \mathrm{B}_i^{(j)} \in c$ of degree at most 1:
    Remove any neighbor of $\mathrm{B}_i^{(j)}$ and all its edges from $c$.
    Remove $\mathrm{B}_i^{(j)}$ from $c$.
Output: $C$.
```

Lemma 2. *For any graph satisfying Condition 1, Algorithm 1 runs in $\Theta(ve+v^2)$ and identifies a nontrivial disjoint union of multi-connected components of edge connectivity at least 2 if one exists and otherwise returns an empty set.*

Proof: The proof is shown in [43]. ∎

V. Applications of the new toolkit of locality

Recall that Section IV presented new tools to exploit locality properties of each coordinate of the input points to reduce the worker threshold. Next, we highlight a concrete, practical application: multiplying upper triangular matrices. Due to taking the transpose of the matrix product, the methodology likewise applies to multiplying lower triangular matrices.

Consider upper triangular matrices U and Z, respectively, over a field, $\mathbb{F}$. Let the dimensions of U and Z be $eg \times eg$ and $eg \times er$, respectively, for positive integers e, r, and g where $r \geq g$. Suppose that U and Z are each partitioned into $e \times e$ sized blocks,

$$\begin{bmatrix} U_{1,1} & U_{1,2} & \dots & U_{1,g} \\ 0 & U_{2,2} & \ddots & U_{2,g} \\ \vdots & \ddots\ddots & \vdots & \\ 0 & 0 & \dots & U_{g,g}, = \end{bmatrix}, \begin{bmatrix} Z_{1,1} & Z_{1,2} & \dots & Z_{1,r} \\ 0 & Z_{2,2} & \ddots & Z_{2,r} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & Z_{g,r} \end{bmatrix},$$

where for any $i, j \in [g]$ and $l \in [r]$ each of $U_{i,j}$, and $Z_{j,l}$ is an $e \times e$ matrix. To compute $U * Z$, there are $g^2 r$ pairs to consider: $\{U_{i,j} Z_{j,l} \mid i, j \in [g], l \in [r]\}$. Since U and Z are upper triangular matrices, for $i, j \in [g]$ and $l \in [r]$, $U_{i,j} \neq 0$ only if $i \leq j$ and $Z_{j,l} \neq 0$ only if $j \leq l$. Thus, the number of nonzero products is at most

$$\begin{aligned} k &= \sum_{i=1}^{g} \sum_{j=i}^{g} \sum_{l=j}^{r} 1 \\ &= (g^2 r)/2 - g^3/3 + (gr)/2 + g/3 \end{aligned}$$

Next, we discuss how this computation fits into our model. Computing one of the smaller matrix products (i.e., $U_{i,j} * Z_{j,l}$ for $i, j \in [g], l \in [r]$) can be viewed as evaluating a $2-$multihomogeneous polynomial function, f, at $(U_{i,j}, Z_{j,l})$ where the first coordinate is $U_{i,j}$ and the second coordinate is $Z_{j,l}$. The input points are the pairs of matrices to be multiplied. The basis elements for the first coordinate are the nonzero blocks of U, and the basis elements for the second coordinate are the nonzero blocks of Z. Each basis element of the first coordinate (i.e., any $U_{i,j}$ for $i \leq j \in [g]$) is the first coordinate of $(1+r-j) \geq (1+r-g)$ input points (i.e., nonzero matrix products). Hence, Theorem 2 shows that coded computation can be accomplished with $\big(2k-1+s-(r-g)\big)$ workers by interpolating $f(p(z))$ for a curve, $p(z)$, defined in the proof of Theorem 2. Because $p(z)$ is known, this approach has no extra cost compared to alternatives like the Matdot Code [12] or Entangled Polynomial Code [17], which would require $(2k-1+s)$ or $(g^2 r + g - 1 + s)$ workers, respectively.

As a concrete example, consider $g = 2$ and $r = 3$. The Entangled Polynomial Code requires $(13+s)$ workers, the Matdot code requires $(13+s)$ workers, and the code obtained using Theorem 2 requires $(12+s)$ workers. In this example, when $s = 1$, the overhead of coded computation is **reduced by up to** 14% **compared to state-of-the-art.**

VI. Conclusion and future work

Coded computation alleviates the tail latency of distributed computations such as matrix multiplication by providing robustness to stragglers. The locality of codes has recently been shown as a viable technique to reduce the overhead of using coded computation. Our work expands the toolkit of locality-based coded computations for the broad class of m-homogeneous polynomial functions to leverage the structure of each coordinate of the input points. We highlight a structure that arises in many computations and how to represent it with a graph. One potential avenue for future work is to use graph theory to study new exploitable properties of the coordinates of the input points.

References

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] 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, 2017.

[3] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *2016 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2016, pp. 1–6.

[4] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[5] 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.

[6] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2017, pp. 2403–2407.

[7] ——, ""short-dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, 2019.

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

[9] 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.

[10] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2017, pp. 2418–2422.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *NeurIPS*, 2017, pp. 4406–4416.

[12] 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.

[13] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5152–5160.

[14] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2018, pp. 1993–1997.

[15] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2018, pp. 1988–1992.

[16] H. Jeong, F. Ye, and P. Grover, "Locally recoverable coded matrix multiplication," in *56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018, pp. 715–722.

[17] 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.

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

[19] 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.

[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] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

[30] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *IEEE Int. Symp. Inf. Theory*. IEEE, 2018, pp. 2027–2031.

[31] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *IPDPSW*. IEEE, 2018, pp. 857–866.

[32] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *International Conference on Machine Learning*. PMLR, 2018, pp. 903–912.

[33] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5610–5619.

[34] 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.

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

[36] 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.

[37] J. Kosaian, K. Rashmi, and S. Venkataraman, "Learning a code: Machine learning for approximate non-linear coded computation," *arXiv preprint arXiv:1806.01259*, 2018.

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

[39] J. Kosaian, K. Rashmi, and S. Venkataraman, "Parity models: erasure-coded resilience for prediction serving systems," in *SOSP*, 2019, pp. 30–46.

[40] 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 *ICDCS*, December 2020.

[41] 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*, pp. 1–1, 2021.

[42] N. Charalambides, M. Pilanci, and A. O. Hero, "Approximate weighted cr coded matrix multiplication," in *ICASSP*. IEEE, 2021, pp. 5095–5099.

[43] M. Rudow, V. Guruswami, and K. Rashmi, "On expanding the toolkit of locality-based coded computation to the coordinates of inputs," http://www.cs.cmu.edu/~rvinayak/papers/locality_of_coordinates_ISIT_2023.pdf, 2023.