2021 IEEE International Symposium on Information Theory (ISIT) | 978-1-5386-8209-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISIT45174.2021.9518056

A locality-based lens for coded computation

Michael Rudow, K.V. Rashmi, and Venkatesan Guruswami
Carnegie Mellon University, Pittsburgh PA, 15213
Email: mrudow@andrew.cmu.edu, rvinayak@cs.cmu.edu, venkatg@cs.cmu.edu

***Abstract*— Coded computation is an emerging paradigm for robustness in large-scale distributed computing, which applies principles from coding theory to provide robustness against slow or otherwise unavailable workers. We propose a new approach to view coded computation via the lens of locality of codes. We do so by defining a new notion of locality, called *computational locality*, via the locality properties of an appropriately defined code for the function being computed. This notion of locality incorporates the unique aspects of locality arising in the context of coded computation. Using this new approach, (1) We demonstrate how to design a coded computation scheme for a function using the local decoding scheme of an appropriately defined code. This rederives the best-known coded computation scheme for multivariate polynomial functions via the viewpoint of locality of the Reed Muller code. (2) We show that the proposed locality-based approach enables coded computation schemes with significantly lower resource overhead than existing schemes. Specifically, matrix multiplication over complex numbers, a common workload in high performance computing, is achieved with 33.3% fewer workers than state-of-the-art coded computation schemes.**

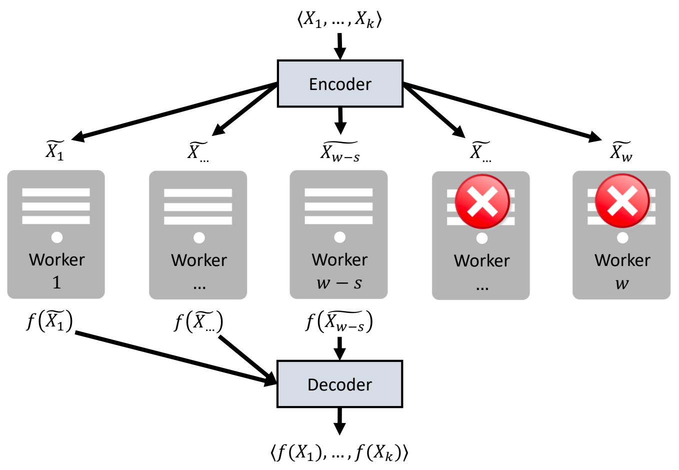

Fig. 1. Overview of the coded computation setting. A master encoder/decoder uses w workers to compute the value of the function f at k input points subject to at most s straggling workers.

I. Introduction

Large scale computations involving smaller modular pieces are widely prevalent. Such computations suffer from unavailabilities due to "straggling" servers (i.e. servers that fail or are slow to respond) [1], which increase the tail latency. One natural approach to address server unavailabilities is to add redundancy, such as replicating the computations on multiple servers. However, replication entails high resource overhead. The alternative approach of employing erasure codes can significantly reduce this overhead and is termed "coded computation" [2].

The setting is as shown in Figure 1, consisting of one master encoder/decoder node and many workers. The master's objective is to compute the value of a function f at k input points $\langle X_1, \ldots, X_k \rangle$ using the workers, while being tolerant to some of the workers straggling and not returning any outputs. Under coded computation, the master queries workers with *encoded* inputs so that the missing outputs from unavailable workers can be decoded using the received worker outputs. The setting also applies to scenarios where a large computation is broken down into smaller ones. For example, multiplication of two large matrices where the matrix product is partitioned into k smaller matrix products, where the function f computes the smaller matrix products. Lee et. al., introduced coded computation for straggler tolerance in matrix multiplication operations [2]. Several subsequent works have further studied coded computation for matrix multiplication and a variety of other computations, including for multilinear functions [3]–[20] and multivariate polynomial functions [21].

This work was funded in part by NSF grants CNS-1850483, CCF-1563742 and CCF-1814603, and in part by the AIDA project (POCI-01-0247-FEDER-045907).

In this paper, we propose a new approach to view coded computation *via the lens of locality of codes*. We define a new notion of locality, called *computational locality*, via the locality properties of an appropriately defined code for the function being computed. There is rich literature on the topic of locality of codes [22]–[28]. However, the conventional notion of locality of codes is insufficient to capture the unique aspects that arise in the coded computation setting (as detailed in Section III). We overcome this limitation by carefully defining a code corresponding to the function being computed.

The proposed locality-based lens connects the concept of coded computation to the concept of locality of code, and enables one to design coded computation schemes for a function using the local decoding schemes for an appropriately defined code. Specifically, we show how the well-known local decoding scheme for the Reed-Muller code [29], [30] can be used to obtain a coded computation scheme for multivariate polynomial functions. The scheme so obtained rederives state-of-the-art coded computation schemes for multivariate polynomials and matrix multiplication from an alternative viewpoint, that of locality of Reed Muller codes.

In addition to providing a unified perspective, the proposed locality-based lens enables new tools for designing coded computation schemes with significantly lower resource overhead than existing schemes for certain, widely computed functions. Specifically, we present a coded computation scheme to multiply complex-valued matrices with 33.3% fewer workers than that of the state-of-the-art scheme for the practically motivated case of tolerating a single straggler. We achieve this by first showing that the locality-based approach enables one to exploit linear dependencies in input points to make do with fewer workers. We then show that matrix-matrix multiplication over complex numbers involves linearly dependent data points at the workers, and that these linear dependencies does not require any additional computation to extract.

II. Setting, Background and Related Works

We now discuss the problem setting, related works and define some notation used throughout the paper.

A. Setting

The setting is shown in Figure 1 and consists of one master with an encoder/decoder and many workers. We adopt the notation from the coded computation literature, e.g., [21]. The master's objective is to evaluate a function f at k input points $\langle X_1, \ldots, X_k \rangle$ using w workers. The master "queries" each worker $i \in \{1, \ldots, w\}$ with the evaluation point $\widetilde{X_i}$ which is a function of the input points. The worker then computes $f(\widetilde{X_i})$. f can be any function in a set of functions $\mathcal{F}$, which is denoted as a "function class." Up to s arbitrary workers may straggle and return nothing to the master.[1] The remaining $(w - s)$ workers return their computed values to the master. The master uses the outputs of the $(w - s)$ reliable workers to decode $\langle f(X_1), \ldots, f(X_k) \rangle$. The minimum number of workers needed to do so is defined as follows:

Definition 1 (Worker threshold)**.** *The* $(\mathcal{F}, \langle X_1, \ldots, X_k \rangle, s)$*-worker threshold is the minimum number of workers, denoted by* $w_{\mathcal{F}, \langle X_1, \ldots, X_k \rangle, s}$*, for which coded computation for any function* f *in a function class* $\mathcal{F}$ *at the input points* $\langle X_1, \ldots, X_k \rangle$ *tolerating up to* s *straggling workers is possible.*

B. Related works

The domain of coded computation has received considerable attention. The function classes studied in most existing works on coded computation fall under the umbrella of multivariate polynomial functions. For example, many works have considered linear settings such as matrix-vector multiplication [3]–[5], [8], [9], [20], convolution [6], linear transforms [7], etc. Coded computation has also been studied for matrix-matrix multiplication [10]–[15], [17]–[19] and the matrix product of more than two matrices [12].

When computing a matrix product involving large matrices, the computation is partitioned into smaller matrix products to be computed by the workers, and the outputs from the workers are combined at the master to produce the final output [4], [5], [9]–[18], [20]. Tolerance to unavailabilities, and hence coded computation, would be needed for the smaller matrix products computed by the workers. Thus, matrix multiplication fits into the setting described in Section II-A *after* the partitioning is specified. The specific partitioning can impose extra structure on the smaller computations, as is discussed in Section V-B.

In [21], the authors propose a coded computation scheme for multivariate polynomials called the Lagrange Coded Computing (LCC). The model studied in [21] requires that the same linear encoding and decoding schemes are used for *any* k input points. We denote this setting as *input-oblivious*. The authors prove that the worker threshold for k input points and s stragglers is at least $\min(k(s+1), (k-1)deg(f)+s+1)$ in the input-oblivious setting. The LCC scheme meets this bound.

A different notion of locality in the context of coded computation than what we study in this paper was studied in [16]. It considers recovering the output of a worker "locally" by querying only a few reliable workers. Recovering the output of workers locally is an extra requirement in addition to recovering the desired function output using all reliable workers and incurs a higher worker threshold. In contrast, our work presents an approach to view the objective of computing the desired function output under the lens of "locality of codes." This new viewpoint helps to reduce the worker threshold in several scenarios, as will be discussed in Section V.

Robust distributed computation for machine learning has also received substantial interest. Most existing work focuses on the training phase, designing coded approaches for gradient computations during gradient descent [31]–[38]. In [39], [40], the authors introduced coded computation for neural network inference, a task which is highly non-linear, by presenting a learning-based approach. This approach has been studied further in [41], [42]. These learning-based approaches approximately reconstruct the unavailable function outputs.

C. Notation and Preliminaries

We use $[i]$ to denote $\{1, \ldots, i\}$. Let $c = \langle c_1, \ldots, c_n \rangle$ and c' be codewords of a code C. The **Hamming distance** $\Delta(c, c')$ is the number of symbols in which c and c' differ. The **Hamming ball** $B(c, r)$ equals $\{c_* \in C \mid \Delta(c, c_*) \leq r\}$. For $I = \{i_1, \ldots, i_l\} \subseteq [n]$, the **punctured** codeword is $c_I = \langle c_{i_1}, \ldots, c_{i_l} \rangle$ and the **punctured** code is $C_I = \{c_I \mid c \in C\}$.

III. Locality-Based Lens for Coded Computation

In this section, we present a new approach to view coded computation via the lens of locality of codes. In order to do so, we define a new notion of locality called *computational locality*. The direct use of the conventional notion of locality of codes is insufficient for the coded-computation setting due two main aspects: (1) the coded computation setting admits replicated queries, and (2) the coded computation setting handles arbitrarily many inputs whereas traditional local decoding schemes usually consider one (or sometimes two) inputs. To overcome this limitation, we first introduce and define several entities which enable us to define an appropriate code associated with the function being computed. We then use the so defined code to define a new notion of locality which captures the unique aspects of coded computation. We limit our focus here to the function class of multivariate polynomials and defer the discussion of more general functions to [43].

[1]The model can be extended to include adversarial workers, as in [21]. Thus, the results of this work extend to the adversarial setting as well.

A. Computational Locality

Let s be a positive integer and $\mathbb{F}_q = \{\alpha_1, \dots, \alpha_q\}$ be a finite field. Let $\mathcal{F} = \mathbb{F}_q[x_1, \dots, x_m]$ be the function class of all multivariate polynomials of total degree at most d.

Definition 2 (Associated codeword and code)**.** *For a function f in the function class $\mathcal{F}$ over domain $\mathbb{F}_q^m$, the* ***associated codeword*** *is $c^f = \langle f(v) \mid v \in \mathbb{F}_q^m \rangle$. The* ***associated code*** *for the function class $\mathcal{F}$ is defined as $C^{\mathcal{F}} = \{c^f \mid f \in \mathcal{F}\}$.*

A Reed-Muller code is defined over a set of degree bounded multivariate polynomials (e.g., $\mathcal{F}$). Each polynomial is associated with a codeword comprising the evaluations of the polynomial over all points in the domain.[2]

Observation 1. *The associated code for the function class of multivariate polynomials is a Reed-Muller code.*

Existing literature on locality of codes [22]–[27] usually considers a code where each code symbol appears exactly once. The associated codeword reflects this convention. However, the coded computation domain admits replicating a computation over multiple workers, motivating the next definition.

Definition 3 (Repeated codeword and code)**.** *The* ***repeated codeword*** *for a codeword $c = \langle c_1, \dots, c_n \rangle$ of a code C is defined as $c^{\langle s+1 \rangle} = c \times [s+1]$. The* ***repeated code*** *for C is $C^{\langle s+1 \rangle} = \{c^{\langle s+1 \rangle} \mid c \in C\}$.*

We next define the notion of computational locality.

Definition 4 (*Computational locality of code symbols and codes*)**.** *Let $k \leq n$ and s be non-negative integers, $I = \{i_1, \dots, i_k\} \subseteq [n]$, and $C \subseteq \mathbb{F}_q^n$ be a code. The* ***computational locality*** *$L_{I,s}$ of code symbols C_I is the minimum integer for which there exists a size $L_{I,s}$ set $J \subseteq [(s+1)n]$ such that $\forall c, c' \in C^{\langle s+1 \rangle}, c'_J \in B(c_J, s) \Rightarrow c'_I = c_I$. The* ***computational locality*** *$L_{k,s}$ of code C is $\max_{I=\{i_1,\dots,i_k\}\subseteq[n]}(L_{I,s})$*

At a high level, Definition 4 says that the code symbols C_I can be uniquely decoded using any $(L_{I,s} - s)$ code symbols of a length $L_{I,s}$ punctured code of $C^{\langle s+1 \rangle}$. The structure of C_I can be exploited for decoding. In contrast, under an input-oblivious setting, the same linear decoding scheme must apply for all C_I. Next, this notion is extended to a function class.

Definition 5 (*Computational locality of a function class*)**.** *Let $k \leq n$ and s be non-negative integers. The* ***computational locality*** *of a function class $\mathcal{F}$ equals $L_{k,s}(\mathcal{F})$ where $L_{k,s}(\mathcal{F})$ is the computational locality of the associated code $C^{\mathcal{F}}$.*

Corollary 1. *The computational locality for the function class of multivariate polynomials is the computational locality of the Reed-Muller code.*

B. Connecting Computational Locality to Coded Computation

We now connect computational locality to coded computation. For any $f \in \mathcal{F}$, consider arbitrary input points $\langle X_1, \dots, X_k \rangle$ over domain $\mathbb{F}_q^m$. Now consider the associated codeword for f, $c^f = \langle f(v) \mid v \in \mathbb{F}_q^m \rangle$. There is a set I of size k corresponding to $c_I^f = \langle f(X_1), \dots, f(X_k) \rangle$. Now consider the repeated codeword $(c^f)^{\langle s+1 \rangle} = c^f \times [s+1]$. The minimum number of codeword symbols of $(c^f)^{\langle s+1 \rangle}$ which must be queried to uniquely decode c_I^f when up to s queries are lost is the computational locality, $L_{I,s}$. Consider any scheme which decodes c_I^f using $L_{I,s}$ queries to the codeword symbols of $(c^f)^{\langle s+1 \rangle}$ indexed by $J \subseteq [(s+1)q^m]$. We label the codeword symbols $(c^f)_J^{\langle s+1 \rangle} = \langle f(\widetilde{X_1}), \dots, f(\widetilde{X_{L_{I,s}}}) \rangle$ for some $\langle \widetilde{X_1}, \dots, \widetilde{X_{L_{I,s}}} \rangle$. The analog under coded computation for this scheme uses $L_{I,s}$ workers to compute $\langle f(\widetilde{X_1}), \dots, f(\widetilde{X_{L_{I,s}}}) \rangle$ and use the same algorithm to decode $\langle f(X_1), \dots, f(X_k) \rangle$ at the master. The number of workers used for coded computation equals the number of codeword symbols which are queried. Hence, the computational locality equals the worker threshold.

Theorem 1 (*Coded computation via computational locality*)**.** *Let $k \leq q^m$ and s be non-negative integers. The computational locality of the function class $\mathcal{F}$ equals the $(\mathcal{F}, k, s)$-worker threshold (i.e. $L_{k,s}(\mathcal{F}) = w_{\mathcal{F},k,s}$).*

Proof sketch: Follows from Definitions 2, 3, 4, and 5 and the connection between computational locality and worker threshold discussed above. More details are shown in [43]. ■

Theorem 1 enables design of coded computation schemes using results on locality, as is discussed in subsequent sections.

IV. Coded Computation of Multivariate Polynomials via the Lens of Locality

In this section, we discuss how to adapt existing locality results into the coded computation setting. Specifically, we show how to convert local decoding schemes for the Reed-Muller code into a statement on the computational locality of the function class of degree bounded multivariate polynomials. The techniques presented apply to evaluating a vector where each coordinate is a polynomial by applying them component-wise. Hence, the techniques apply to coded computation of functions such as matrix multiplication.

We will use the same notation discussed in Section III. Let k be a non-negative integer and the polynomial $f \in \mathcal{F}$ have degree at most d. Recall that the associated code $C^{\mathcal{F}}$ is a Reed-Muller code.

Reed-Muller codes have been extensively studied and exhibit many useful algebraic properties. The most relevant such property to this work is that certain puncturings of the associated codeword c^f are *univariate* polynomials. Specifically, for any *univariate* polynomial $p : \mathbb{F}_q \to \mathbb{F}_q^m$, the corresponding vector $\langle f(p(\alpha)) \mid \alpha \in \mathbb{F}_q \rangle$ is a puncturing of c^f. The composition $f(p(z))$ is a degree $d' = deg(f)deg(p)$ univariate polynomial. One can recover a degree d' univariate polynomial via polynomial interpolation of $(d'+1)$ distinct points.

Several prior works in the literature on the locality of codes have exploited this property in designing local decoding schemes for Reed-Muller codes [22], [44]–[47]; the regimes of when p is a line or a quadratic curve have been especially

[2]Reed-Muller codes over non-binary alphabets (i.e. $|\mathbb{F}_q| > 2$) are usually called "generalized Reed-Muller codes." We omit the word "generalized" in this work for conciseness.

well-studied (such as in [22], [44]–[46] and [47]). We extend this local decoding technique to prove a computational locality result for multivariate polynomial function by considering the univariate polynomial $p(z)$ which passes through k code symbols and interpolating the univariate polynomial $f(p(z))$. The key property is that every set of k code symbols is part of a corresponding group of $((k-1)d+s+1)$ code symbols wherein every group of s symbols is redundant.[3]

Theorem 2. *The computational locality for multivariate polynomials of total degree at most d is $L_{k,s}(\mathcal{F}) \leq \min(k(s+1),(k-1)d+s+1)$.*

Proof sketch: Follows from interpolating the curve of degree at most $(k-1)$ passing through the k input points and $(s+1)$-wise replication. For more details, see [43]. ■

Recall from Section III-B that coded computation for a polynomial at k input points corresponds to the computational locality of k corresponding code symbols. This mapping directly translates the variant of local decoding for Reed-Muller codes employed in Theorem 2 into a coded computation scheme for multivariate polynomials. *The so-obtained coded computation scheme rederives well-known, state-of-the-art coded computation schemes for polynomials and matrix multiplication, from the viewpoint of locality of Reed-Muller codes*. For the function class of polynomials, this rederives the LCC scheme [21]. In the setting of matrix multiplication of a pair of matrices, after the partitioning of the computation, the systematic Matdot construction [12] is likewise rederived. Other coded computation schemes, such as the "Polynomial code" from [11], can also be viewed under the lens of computational locality, as is discussed in [43].

V. Coded computation with lower overhead via computational locality

In Section IV, we discussed computational locality properties of multivariate polynomials which do not use the structure of the specific input points. The overhead in the number of workers is high for coded computation schemes which likewise are oblivious to the structure of the input points, as shown in [21]. Next, we use the lens of locality to exploit the computational locality of specific input points to make do with fewer workers than existing coded computation schemes.

We first show how linear dependencies in the input points lead to improved computational locality for homogeneous multivariate polynomial functions. Such linear dependencies naturally arise in several applications. For example, a technique for complex matrix multiplication induces linear dependencies which can be exploited via the proposed locality-based model to reduce the number of workers by 33.3%, as will be shown in Section V-B. The results are extended to nonhomogeneous multivariate polynomial functions in [43]. Due to Theorem 1, this establishes a lower worker threshold for coded computation for linearly dependent input points.

[3]This requires a sufficiently large field size (i.e. $(k-1)d+s+1 \leq q$).

A. Computational locality for linearly dependent input points

In this section, we focus on the class of homogeneous multivariate polynomial functions. Such polynomials have the fundamental property that multiplying the input by a field element is equivalent to multiplying the output by an associated power of the field element. The results shown in this section are extended to nonhomogeneous polynomials in [43].

A homogeneous polynomial $f \in \mathbb{F}[x_1,\ldots,x_m]$ is defined as $f(X)=\sum_{i\in I} p_i(X)$ for an index set I where $\forall i \in I, p_i \in \mathbb{F}[x_1,\ldots,x_m]$ is a monomial of degree $deg(f)$. For any homogeneous polynomial f and $\alpha \in F$, $f(\alpha X)=\alpha^{deg(f)}f(X)$. One can evaluate f at any input point by evaluating f at a *nonzero multiple of the input point*. One can use this property to make do with fewer workers for coded computation.

We now discuss a toy example of a coded computation scheme for a homogeneous polynomial f subject to s straggling workers where $s=(deg(f)-1)$ and $deg(f)>1$. There are three linearly dependent input points in the x-y plane, $\langle(1,0),(0,1),(1,1)\rangle$. The line $l(z)=(2-z,z)$ intersects a nonzero multiple of each input, and by homogeneity, $f(\alpha X)=\alpha^{deg(f)}f(X)$ for $\alpha\in\mathbb{R}$. Hence, computing $f(l(z))$ determines $\langle f((2,0)),f((0,2)),f((1,1))\rangle$. This requires $2(s+1)$ workers, whereas the best-known scheme would use $3(s+1)$ workers.

Generalizing the above scheme leads to the following result.

Theorem 3 (Computational locality and coded computation of homogeneous polynomials at linearly dependent input points)**.** *Let $\mathcal{F}$ be the class of homogeneous multivariate polynomials of degree at most d over a vector space $\mathbb{F}^m$ for a field $\mathbb{F}$. Let k and s be non-negative integers where $|\mathbb{F}| \geq (k-2)d+s+1$. Let $\langle X_1,\ldots,X_k\rangle$ be linearly dependent input points such that $\sum_{i=1}^{k-1}\alpha_i X_i = X_k$ for $\alpha_1,\ldots,\alpha_{k-1}\in\mathbb{F}\setminus\{0\}$. The computational locality, $L_{s,\{X_1,\ldots,X_k\}}(\mathcal{F})$, and $(\mathcal{F},\langle X_1,\ldots,X_k\rangle,s)$-worker threshold are each at most $(k-2)deg(f)+s+1$.*

Proof sketch: Follows from homogeneity, component-wise interpolation of a univariate polynomials passing through a nonzero multiple of each of the k vectors, and Theorem 1. A complete proof is included in [43]. ■

In contrast, $\min((k-1)deg(f)+s+1, k(s+1))$ workers are used by the state-of-the-art scheme. When the linear dependencies involve a small number of input points, the improvement is substantial, as will be discussed in Section V-B.

Note that Theorem 3 can also be separately applied to individual linearly dependent subsets of a set input points.

B. Exploiting the lens of locality for matrix multiplication

In this section, we use the lens of computational locality to design coded computation schemes with significantly lower worker threshold than state-of-the-art schemes for matrix multiplication over complex numbers. Complex matrix multiplication is an important workload for scientific computing and engineering. Several works have studied efficient methods for such computations [48]–[51]. Recall from Section II that for distributed multiplication of large matrices, the computation is partitioned into smaller matrix products to be computed by the workers and the outputs from the workers are combined at the master to produce the final output. The technique employed by the workers to multiply the smaller matrices can lead to computing the matrix product of linearly dependent pairs of matrices (as shown below). These linear dependencies can be exploited, via the locality-based model, to achieve straggler tolerant coded computation using 33.3% *fewer workers than the state-of-the-art Matdot code [12]*. The partitioning of the matrix product into smaller components leads to several different trade-offs, including (a) the number of workers needed, (b) the storage used by the workers, and (c) the per-worker computational load. We will discuss each of these trade-offs in turn. A similar approach applies to multivariate polynomial functions, as will be discussed at the end of the section.

We start by describing how the systematic Matdot code [12] computes the matrix product of matrices A and B, each of size $(t \times t)$, using the notation of the proposed locality-based model of coded computation. Let $*_{\mathbb{C}}$ denote matrix multiplication of complex matrices of sizes $(t \times \frac{t}{k})$ and $(\frac{t}{k} \times t)$ respectively for a positive integer k. The computation is broken down into $\begin{bmatrix} A_1 & \cdots & A_k \end{bmatrix} \begin{bmatrix} B_1 & \cdots & B_k \end{bmatrix}^T = \sum_{j=1}^{k} A_j *_{\mathbb{C}} B_j$. The scheme uses $(2k - 1 + s)$ workers and tolerates up to s stragglers. Each worker $j \in [2k - 1 + s]$ evaluates $f(\tilde{A}_j, \tilde{B}_j) = \tilde{A}_j *_{\mathbb{C}} \tilde{B}_j$. The first k workers respectively compute $\langle A_1 *_{\mathbb{C}} B_1, \ldots, A_k *_{\mathbb{C}} B_k \rangle$, whose sum equals $A *_{\mathbb{C}} B$.

Next, we examine the technique used by each worker to compute the complex-valued matrix product $*_{\mathbb{C}}$. The worker may do so by computing real-valued matrix products, $*_{\mathbb{R}}$, due to the availability of specialized hardware and software optimizing real-valued matrix multiplication [48]. The trade-offs for this methodology are discussed in in detail [48]. The following are used to compute $\tilde{A}_j *_{\mathbb{C}} \tilde{B}_j$: $R_A^{(j)} := RE[\tilde{A}_j], I_A^{(j)} := IM[\tilde{A}_j], R_B^{(j)} := RE[\tilde{B}_j]$, and $I_B^{(j)} := IM[\tilde{B}_j]$. The so-called "4M" method computes $\tilde{A}_j *_{\mathbb{C}} \tilde{B}_j$ with 4 evaluations of $*_{\mathbb{R}}$: (1) $R_A^{(j)} *_{\mathbb{R}} R_B^{(j)}$, (2) $I_A^{(j)} *_{\mathbb{R}} I_B^{(j)}$, (3) $R_A^{(j)} *_{\mathbb{R}} I_B^{(j)}$, and (4) $I_A^{(j)} *_{\mathbb{R}} R_B^{(j)}$. Alternatively, it is well-known that what is dubbed the "3M method" [50] only uses 3 evaluations of $*_{\mathbb{R}}$, by replacing (3) and (4) above with $(R_A^{(j)} + I_A^{(j)}) *_{\mathbb{R}} (R_B^{(j)} + I_B^{(j)})$ as follows:

$$\begin{aligned} f(\tilde{A}_j, \tilde{B}_j) =& R_A^{(j)} *_{\mathbb{R}} R_B^{(j)} - I_A^{(j)} *_{\mathbb{R}} I_B^{(j)} - i R_A^{(j)} *_{\mathbb{R}} R_B^{(j)} - \\ & i I_A^{(j)} *_{\mathbb{R}} I_B^{(j)} + i (R_A^{(j)} + I_A^{(j)}) *_{\mathbb{R}} (R_B^{(j)} + I_B^{(j)}). \end{aligned}$$

Next, we exploit computational locality properties introduced by the 3M method to reduce the worker threshold. For simplicity of exposition, we present results for $k = 3$ and one straggler. The results we obtain can be easily extended to larger values for k, as will be discussed later. The Matdot code uses 6 workers, each of whom computes the product of complex matrices of sizes $(t \times \frac{t}{3})$ and $(\frac{t}{3} \times t)$ respectively. Recall that $A *_{\mathbb{C}} B = \sum_{j=1}^{3} A_j *_{\mathbb{C}} B_j$ where $A_j *_{\mathbb{C}} B_j$ is computed by worker j by evaluating $*_{\mathbb{R}}$ for 3 *linearly dependent* pairs of matrices. To leverage the linear dependencies via the proposed locality-based model, one can redistribute the task of computing the matrix product of the 9 pairs of matrices over the workers. Instead of computing f, each worker will compute the function $f_{\dagger}(\langle (C_1, D_1), (C_2, D_2), (C_3, D_3) \rangle) = \langle C_1 *_{\mathbb{R}} D_1, C_2 *_{\mathbb{R}} D_2, C_3 *_{\mathbb{R}} D_3 \rangle$. The per-worker computational load remains that of evaluating $*_{\mathbb{R}}$ 3 times. However, each worker stores 3 pairs of real-valued matrices rather than 1 pair of complex-valued matrices (i.e. 50% more data).

Locality-based scheme: Given f and $\langle (A_j, B_j) \mid j \in [3] \rangle$, the master will use the workers to compute $f_{\dagger}$ over $X_1 = \langle (R_A^{(j)}, R_B^{(j)}) \mid j \in [3] \rangle$, $X_2 = \langle (I_A^{(j)}, I_B^{(j)}) \mid j \in [3] \rangle$, and $X_3 = \langle (R_A^{(j)} + I_A^{(j)}, R_B^{(j)} + I_B^{(j)}) \mid j \in [3] \rangle$. These input points reflect the known structure of the $3M$ method, and the fact that $(X_1 + X_2) = X_3$ is extracted for free. Applying Theorem 3 to these *linearly dependent input points* shows that the worker threshold to compute $\langle f_{\dagger}(X_j) \mid j \in [3] \rangle$ is at most 4. The span of $\{f_{\dagger}(X_j) \mid j \in [3]\}$ contains $\{A_j *_{\mathbb{C}} B_j \mid j \in [3]\}$ and by extension $A *_{\mathbb{C}} B$. The locality-based scheme computes $A *_{\mathbb{C}} B$ while **reducing the number of workers by** 33.3%. More generally, when $k = (3k_1 + k_2)$ where $0 \le k_2 < 3$, the Matdot code uses $2k = (6k_1 + 2k_2)$ workers while one can make do with $(4k_1 + 2k_2)$ workers by using the locality-based scheme k_1 times and replicating the final k_2 matrix products. When workers use the 4M method, one can similarly exploit the linear dependency $\Big((R_A^{(j)}, R_B^{(j)}) + (I_A^{(j)}, I_B^{(j)}) = (R_A^{(j)}, I_B^{(j)}) + (I_A^{(j)}, R_B^{(j)})\Big)$.

We next compare three state-of-the-art coded computation schemes for matrix multiplication to the locality-based scheme in terms of (a) the number of workers, (b) the per-worker storage, and (c) the per-worker computational load. First, the computation of $A *_{\mathbb{C}} B$ could be partitioned into $k = 2$ matrix products using the Matdot code [12] rather than $k = 3$ matrix products (as described above). The scheme so-obtained uses the same number of workers and per-worker storage as the locality-based scheme but has a higher per-worker computational load (specifically, each worker multiplies complex matrices of dimensions $t \times \frac{t}{2}$ and $\frac{t}{2} \times t$). The locality-based scheme can also be viewed as reducing the per-worker computational load for this variant of Matdot without changing the other two metrics. Second, the Polynomial code [11] with parameter settings corresponding to the same per-worker storage requires one extra worker (i.e., 5 workers) and has a lower per-worker computational load (specifically, each worker multiplies complex matrices of dimensions $\frac{t}{2} \times t$ and $t \times \frac{t}{2}$). Third, an MDS code [2] with a 33% higher per-worker storage uses the same number of workers and has a higher per-worker computational load (specifically, each worker multiplies complex matrices of dimensions $\frac{t}{3} \times t$ and $t \times t$).

Similar computational locality properties to those discussed above exist when each worker evaluates a multivariate polynomial by computing a different polynomial (of lower computational load) at linearly dependent points and returning a linear combination of these values. Reallocating the evaluations of the polynomial over the workers and using linear decoding at the master reduces the worker threshold. Examples include workers multiplying field elements with the Karatsuba algorithm [52], extensions of it [53], or multiplying floating point numbers with the technique of [54].

References

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

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

[6] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *ISIT*. IEEE, 2017.

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

[8] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *ISIT*. IEEE, 2018.

[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 *ISIT*. IEEE, 2018.

[10] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *ISIT*. IEEE, 2017.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[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, 2019.

[13] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *ICML*, 2018.

[14] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *ISIT*. IEEE, 2018.

[15] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *ISIT*. IEEE, 2018.

[16] H. Jeong, F. Ye, and P. Grover, "Locally recoverable coded matrix multiplication," in *Allerton*. IEEE, 2018.

[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, 2020.

[18] H. Park, K. Lee, J.-y. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *ISIT*. IEEE, 2018.

[19] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch computation," *arXiv e-prints*, pp. arXiv–1909, 2019.

[20] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication," in *ISIT*. IEEE, 2019.

[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*, 2019.

[22] S. Yekhanin, "Locally decodable codes," *Foundations and Trends®in Theoretical Computer Science*, vol. 6, no. 3, October 2012. [Online]. Available: http://dx.doi.org/10.1561/0400000030

[23] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, 2012.

[24] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, 2014.

[25] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 8, 2014.

[26] N. Prakash, V. Lalitha, S. Balaji, and P. V. Kumar, "Codes with locality for two erasures," *IEEE Trans. Inf. Theory*, vol. 65, no. 12, 2019.

[27] P. Ramakrishnan and M. Wootters, "On taking advantage of multiple requests in error correcting codes," in *ISIT*. IEEE, 2018.

[28] R. Cramer, C. Xing, and C. Yuan, "Efficient multi-point local decoding of reed-muller codes via interleaved codex," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, 2020.

[29] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection," *Transactions of the IRE professional group on electronic computers*, no. 3, 1954.

[30] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, 1954.

[31] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *ICML*, 2017.

[32] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *ISIT*. IEEE, 2018.

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

[34] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *ICML*, 2018.

[35] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *ICML*, 2018.

[36] 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, 2020.

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

[38] 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, 2017.

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

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

[41] 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.

[42] 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.

[43] M. Rudow, K. Rashmi, and V. Guruswami, "A locality-based approach for coded computation," *arXiv preprint arXiv:2002.02440*, 2020.

[44] R. J. Lipton, "Efficient checking of computations," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 1990.

[45] D. Beaver and J. Feigenbaum, "Hiding instances in multioracle queries," in *STACS 90*, C. Choffrut and T. Lengauer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990.

[46] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, "Self-testing/correcting for polynomials and for approximate functions," in *STOC*, vol. 91. Citeseer, 1991.

[47] P. Gemmell and M. Sudan, "Highly resilient correctors for polynomials," *Information processing letters*, vol. 43, no. 4, 1992.

[48] F. G. Van Zee and T. M. Smith, "Implementing high-performance complex matrix multiplication via the 3m and 4m methods," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, no. 1, 2017.

[49] A. E. Eichenberger, M. K. Gschwind, and J. A. Gunnels, "Complex matrix multiplication operations with data pre-conditioning in a high performance computing architecture," Feb. 11 2014, uS Patent 8,650,240.

[50] 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, 1992.

[51] F. G. Van Zee, "Implementing high-performance complex matrix multiplication via the 1m method," *SIAM Journal on Scientific Computing*, vol. 42, no. 5, 2020.

[52] A. Karatsuba, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, 1963.

[53] A. Weimerskirch and C. Paar, "Generalizations of the karatsuba algorithm for efficient implementations." *IACR Cryptol. ePrint Arch.*, vol. 2006, 2006.

[54] S. Arish and R. Sharma, "An efficient floating point multiplier design for high speed applications using karatsuba algorithm and urdhva-tiryagbhyam algorithm," in *ICSC*. IEEE, 2015.