This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

On the Optimal Recovery Threshold of Coded Matrix Multiplication

Sanghamitra Dutta*, Mohammad Fahim*, Farzin Haddadpour*, Haewon Jeong*, Viveck Cadambe, Pulkit Grover

***Abstract*—We provide novel coded computation strategies for distributed matrix-matrix products that outperform the recent "Polynomial code" constructions in recovery threshold, i.e., the required number of successful workers. When a fixed $1/m$ fraction of each matrix can be stored at each worker node, Polynomial codes require m^2 successful workers, while our MatDot codes only require $2m-1$ successful workers. However, MatDot codes have higher computation cost per worker and higher communication cost from each worker to the fusion node. We also provide a systematic construction of MatDot codes. Further, we propose "PolyDot" coding that interpolates between Polynomial codes and MatDot codes to trade off computation/communication costs and recovery thresholds. Finally, we demonstrate a novel coding technique for multiplying n matrices ($n \geq 3$) using ideas from MatDot and PolyDot codes.**

I. Introduction

As the era of Big Data advances, massive parallelization has emerged as a natural approach to overcome limitations imposed by saturation of Moore's law (and thereby of single processor compute speeds). However, massive parallelization leads to computational bottlenecks due to faulty nodes and stragglers [2]. Stragglers refer to a few slow or delay-prone processors that can bottleneck the entire computation because one has to wait for all the parallel nodes to finish. The issue of straggling [2] and faulty nodes has been a topic of active interest in the emerging area of "coded computation" with several interesting works, e.g. [3]–[39]. Coded computation not only advances on coding approaches in classical works in Algorithm-Based Fault Tolerance (ABFT) [40], [41], but also provides novel analyses of required computation time (e.g. expected time [3] and deadline exponents [42]). Perhaps most importantly, it brings an information-theoretic lens to the problem by examining fundamental limits and comparing them with existing strategies. A broader survey of results and techniques of coded computation is provided in [43].

In this paper, we focus on the problem of coded matrix multiplication. Matrix multiplication is central to many modern computing applications, including machine learning and scientific computing. Not surprisingly, there is a lot of interest in classical ABFT literature (starting from [40], [41]) and more recently in coded computation literature (e.g. [6], [44]) to make matrix multiplications resilient to faults and delays. In particular, Yu, Maddah-Ali, and Avestimehr [6] provide novel coded matrix-multiplication constructions called *Polynomial codes* that outperform classical work from ABFT literature in terms of the *recovery threshold*, the minimum number of successful (non-delayed, non-faulty) processing nodes required for completing the computation.

In this work, we consider the standard setup used in [6], [44] with P worker nodes that perform the computation in a distributed manner. A master node helps coordinate the computation by performing some low complexity pre-processing on the inputs and distributes the inputs to the workers. A fusion node aggregates the results of the workers.[1] We propose MatDot codes that advance on existing constructions in scaling sense. More precisely, when an m-th fraction of each matrix can be stored in each worker node, Polynomial codes have the recovery threshold of m^2, while the recovery threshold of MatDot is only $2m-1$. However, as we note in Section III-B, this comes at an increased per-worker computation and communication cost[2]. We also propose PolyDot codes that interpolate between MatDot and Polynomial code constructions in terms of recovery thresholds and per-worker computation/communication costs.

Our main contributions in this work are as follows:

- We present our system model in Section II, and describe MatDot codes in Section III. While Polynomial codes have a recovery threshold of $\Theta(m^2)$, MatDot codes have a recovery threshold of $\Theta(m)$ when each node stores only a fixed $1/m$ fraction of each matrix multiplicand.
- We present a *systematic* version of MatDot codes, where the operations of the first m worker nodes may be viewed as multiplication in uncoded form, in Section IV.
- In Section V, we propose "PolyDot codes," a unified view of MatDot and Polynomial codes that leads to

*Author ordering in alphabetical order. The first four authors contributed equally to the work.

Manuscript received May 14, 2018; revised April 16, 2019.

This work was presented in part at the Annual Allerton Conference on Communication, Control, and Computing (Allerton) in October 2017 [1].

This work was supported by Systems on Nanoscale Information fabriCs (SONIC) which is one of the six SRC STARnet Centers sponsored by MARCO and DARPA, as well as NSF Awards 1350314, 1464336, 1553248 and 1763657.

Sanghamitra Dutta (sanghamd@andrew.cmu.edu), Haewon Jeong (haewon@cmu.edu) and Pulkit Grover (pgrover@andrew.cmu.edu) are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213.

Mohammad Fahim (fahim@psu.edu), Farzin Haddadpour (fxh18@psu.edu) and Viveck Cadambe (vxc12@psu.edu) are with the Department of Electrical Engineering, Pennsylvania State University, University Park, PA 16802.

[1]This separation of a master node from a fusion node is only conceptual, and it makes our exposition easier throughout the paper. One can think of a master node and a fusion node as one physical machine.

[2]Note that, the total computational cost, i.e., the per-worker computational cost multiplied by the total number of workers required to finish (recovery threshold) is similar in scaling sense for both MatDot and Polynomial codes. This is because the per-worker computational cost is increased in MatDot codes by a factor of $\Theta(m)$, while the recovery threshold is reduced by a factor of $\Theta(m)$.

a trade-off between recovery threshold and per-worker computation/communication costs.
- In Section VI, we apply the constructions of Section III to study coded computation for multiplying more than two matrices.

We note that following the publication of an initial version of this paper [1], the works of Yu, Maddah-Ali, and Avestimehr [45] and Dutta, Bai, Jeong, Low and Grover [46] obtained constructions that can reduce the recovery threshold achieved by PolyDot codes within a factor of 2. Nevertheless, MatDot codes continue to have the best known recovery threshold for distributed matrix multiplication under storage constraints. Importantly, Yu *et al.* [45] also provide fundamental limits that show that MatDot codes are, in fact, optimal for the chosen partitioning of the matrices under storage constraints when using linear codes.

II. System Model and Problem Statement

A. System Model

We consider a *computation system* that consists of three different types of nodes as follows: (i) a master node; (ii) worker nodes; and (iii) a fusion node. The roles of the master, fusion and worker nodes are illustrated in Fig. 1.

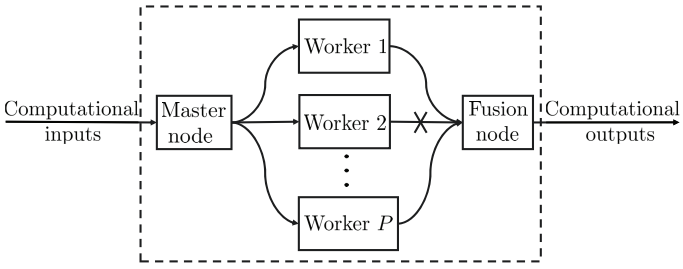

Fig. 1. A computational system: The master node receives the computational inputs and sends appropriate tasks to the workers. The workers are prone to faults and delays. The fusion node aggregates the computational outputs from the subset of successful workers and produces the desired computational outputs.

The following definition specifies a computational system for computing the matrix multiplication: $\mathbf{C} = \mathbf{AB}$.

Definition II.1. [An (N, k, P, m) Computational system for Matrix Multiplication] A computational system consists of the following:

(i) A *master node* that receives computational inputs, i.e., two $N \times N$ matrices $\mathbf{A}$ and $\mathbf{B}$ and obtains, via *linear* pre-processing, $2P$ matrices as follows:

$$\widetilde{\mathbf{A}}_i = f_i(\mathbf{A}) \text{ and } \widetilde{\mathbf{B}}_i = g_i(\mathbf{B}) \text{ for } i = 1, 2, \ldots, P.$$

Here, f_i and g_i are two functions such that $f_i : \mathbb{F}^{N\times N} \to \mathbb{F}^{N/t\times N/s}$ and $g_i : \mathbb{F}^{N\times N} \to \mathbb{F}^{N/s\times N/t}$. Each $\widetilde{\mathbf{A}}_i$ for $i = 1, 2, \ldots, P$ is an $N/t \times N/s$ matrix and each $\widetilde{\mathbf{B}}_i$ for $i = 1, 2, \ldots, P$ is an $N/s \times N/t$ matrix, where s and t are two integers that satisfy $st = m$ and m is an integer that divides N. Specifically, each entry of $\widetilde{\mathbf{A}}_i$ (respectively $\widetilde{\mathbf{B}}_i$) is restricted to be an $\mathbb{F}$-linear combination[3] of the entries of $\mathbf{A}$ (respectively $\mathbf{B}$).

(ii) P *worker nodes* that perform the following operations: For $i = 1, \cdots, P$, the i-th worker node receives $\widetilde{\mathbf{A}}_i, \widetilde{\mathbf{B}}_i$ from the master node, and performs some computation on these matrices. A *successful worker* sends the resulting computation to the fusion node. A *failed* worker does not send the result to the fusion node.

(iii) A *fusion node* that receives outputs from the subset of successful worker nodes. If the number of successful workers is atleast k, the fusion node performs post-processing (e.g., decoding) and produces the computational output $\mathbf{AB}$. Otherwise, it declares a "computation failure."

An (N, k, P, m) computation system for matrix multiplication is one where there are master pre-processing, worker computation and fusion post-processing algorithms such that, if the number of successful workers is at least k, then the output is $\mathbf{AB}$.

We make some informal remarks on the system model before describing our problem statement.

- For a given computation system, the parameter k is referred to as its *recovery threshold*. Note that as per the definition, the recovery threshold is a worst-case evaluation, i.e., over the worst possible choice of inputs $\mathbf{A}, \mathbf{B}$ as well as the worst set of worker failures.
- The parameter m controls the memory of each worker in the model, i.e., each worker node can store only upto a $1/m$ fraction of each of the input matrices.
- For convenience, we simply refer to an (N, k, P, m) computation system for matrix multiplication as a *computation system* in this paper; the parameters N, k, P, m can be inferred from context.
- A worker node can fail due to various reasons such as: (i) straggling due to other jobs in the queue; (ii) straggling due to network congestion; (iii) temporary unavailability (e.g., system updates or power outage). In particular, while our model states that the failed worker nodes do not send their computational outputs to the fusion node, in practice, a straggling worker node that sends its result later than an acceptable deadline may also be considered as a failure in our model. We use the term failed nodes interchangeably with the term *straggling nodes* in this paper. The parameter $P - k$ represents fault-tolerance, or equivalently, the straggler-tolerance of the system.
- Elementary coding theory also implies that an (N, k, P, m) computation systems can correct $\lfloor \frac{P-k}{2} \rfloor$ erroneous worker nodes, i.e., nodes that can output incorrect computations, though we do not focus explicitly on error correction in this paper.
- For a given computation system, the computational complexities of the master, workers, and the fusion node are referred to as the pre-processing, online, and decoding

[3]We restrict pre-processing to be linear to capture memory constraints of each worker node. Note that, allowing for non-linear pre-processing with infinite precision can allow the master node to encode the entire input $\mathbf{A}, \mathbf{B}$ into smaller dimensional matrices over real or complex fields.

complexities. In addition to recovery thresholds, we also evaluate various computation schemes in terms of these computation complexities, as well as the communication cost from the worker nodes to the fusion node. The communication cost between the master node and worker nodes is constant in all the strategies because of the storage constraint, i.e., the master sends upto N^2/m symbols to each worker node.

- Our strategies also extend when the matrices $\widetilde{\mathbf{A}}_i$ and $\widetilde{\mathbf{B}}_i$ are allowed to be of dimensions $N/t_1 \times N/s$ and $N/s \times N/t_2$ (discussed in Remark V.2 later), i.e., asymmetric storage constraints for the two inputs. Our system model also assumes that $\mathbf{A}, \mathbf{B}$ are square matrices with equal dimensions for simplicity of notation. Our ideas and results will naturally apply for cases where $\mathbf{A}, \mathbf{B}$ are non-square matrices as well, as long as the product $\mathbf{AB}$ is defined.

B. Problem Statement

We consider an (N, k, P, m) computation system where the computational complexities of the master, worker and fusion nodes, when evaluated in terms of parameter N, P, m, are all less than the complexity of any sequential algorithm that takes inputs $\mathbf{A}, \mathbf{B}$ and computes the product $\mathbf{AB}$ as the output[4]. Given parameters N, P, m, among these considered systems, our problem is to to determine the computation system with the smallest achievable recovery threshold.

Although the problem stated here remains open, we will present non-trivial coding strategies that achieve significantly smaller recovery threshold than previously known systems. For simplicity, we report results assuming naive matrix multiplication with complexity $\Theta(N^3)$ in our paper; our ideas and results extend, with minor modifications, to include lower complexity algorithms such as Strassen's algorithm [47].

Finally, our problem can be naturally extended to multiplying more than two matrices; we study such extensions in Section VI.

C. Some Notations and Definitions

We now provide some notation used throughout this paper.

- P: The total number of worker nodes used.
- N: The dimension (row/column) of each of the square matrices being multiplied.
- $\mathbf{A}$ and $\mathbf{B}$: The two square matrices being multiplied. Each of them belong to $\mathbb{F}^{N \times N}$ where $\mathbb{F}$ can be any field such that $|\mathbb{F}| > P$.
- m: The storage parameter that denotes that a fixed $1/m$ fraction of each of the input matrices can be stored at each node.

For $f(n)$ and $g(n)$ that are two functions of the variable n, $f(n) = \mathcal{O}(g(n))$ if there exists an n_0 and a constant c such that for all $n > n_0$, $f(n) \leq cg(n)$. Similarly, $f(n) = o(g(n))$ if for any chosen $\epsilon > 0$, one can find an n_0 such that for all $n > n_0$, $f(n) \leq \epsilon g(n)$. Lastly, $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$.

We will be using the term "row-block" to denote the sub-matrices formed when we split a matrix $\mathbf{A}$ horizontally as follows: $\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \end{bmatrix}$. Similarly, we will be using the term "column-block" to denote the sub-matrices formed when we split a matrix vertically into sub-matrices as follows: $\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_1 \end{bmatrix}$.

III. MatDot Codes

In this section, we will describe the distributed matrix-matrix multiplication strategy using MatDot codes, and then examine the computation and communication costs of the proposed strategy. Before proceeding further into the detailed construction and analyses of MatDot codes, we will first give some motivating examples which contrast MatDot codes with existing techniques.

A. Motivating Examples and Summary of Previous Results

Consider the problem statement described in Section II. We describe three different strategies as possible solutions to the problem: (i) ABFT matrix multiplication [40] (also called *product-coded matrices* in [44]), (ii) Polynomial codes [6] and then (iii) our proposed construction, MatDot codes, each progressively improving, i.e., reducing the recovery threshold. We will evaluate the straggler tolerance of a strategy by its recovery threshold, k. For all the examples, we consider the most simple case with $m = 2$. Let us begin by describing the first strategy, namely, ABFT matrix multiplication.

Example III.1. [ABFT codes [40] ($m = 2$, $k = 2\sqrt{P}$)]

Consider two $N \times N$ matrices $\mathbf{A}$ and $\mathbf{B}$ that are split as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_0 & \mathbf{B}_1 \end{bmatrix}$$

where $\mathbf{A}_0, \mathbf{A}_1$ are sub-matrices (row-blocks) of $\mathbf{A}$ of dimension $N/2 \times N$ and $\mathbf{B}_0, \mathbf{B}_1$ are sub-matrices (column-blocks) of $\mathbf{B}$ of dimension $N \times N/2$. Using ABFT, it is possible to compute $\mathbf{AB}$ over P nodes such that, (**i**) each node uses $N^2/2$ linear combinations of the entries of $\mathbf{A}$ and $N^2/2$ linear combinations of the entries of $\mathbf{B}$ and (**ii**) the overall computation is tolerant to $P - 2\sqrt{P}$ stragglers in the worst case. Thus, any $P - (P - 2\sqrt{P}) = 2\sqrt{P}$ worker nodes suffice to recover $\mathbf{AB}$.

ABFT codes use the following strategy: P processors are arranged in a $\sqrt{P} \times \sqrt{P}$ grid. ABFT codes encode two row-blocks of $\mathbf{A}$ and two column-blocks of $\mathbf{B}$ separately using two systematic $(\sqrt{P}, 2)$ MDS codes. Then, we distribute the i-th encoded row-block of $\mathbf{A}$ to all the worker nodes on the i-th row of the grid, and the j-th encoded column-block of $\mathbf{B}$ to all the worker nodes on the j-th column of the grid. Note that here the grid indexing is $i = 1, 2, \ldots, \sqrt{P}$ and $j = 1, 2, \ldots, \sqrt{P}$. An example for $P = 9$ is shown in Fig. 2. The worst case arises when all but one worker node in the lower right $(\sqrt{P}-1) \times (\sqrt{P}-1)$ part of the grid fail. Thus, the worst

[4]The computational complexity requirement is necessary. Without this requirement, it is easy to design a $(N, k = m, P, m)$ computation system by simply storing $\mathbf{A}, \mathbf{B}$ using a (P, m) Maximum Distance Separable code at the workers, which sends the stored symbols to the fusion node which then decodes $\mathbf{A}, \mathbf{B}$ and then performs the multiplication. However, in practice, this is not parallelizing the matrix-multiplication task.

case recovery threshold is $P-(\sqrt{P}-1)^2+1=2\sqrt{P}$. For the example given in Fig. 2 where $P=9$, recovery threshold is $2\sqrt{P}=6$. ■

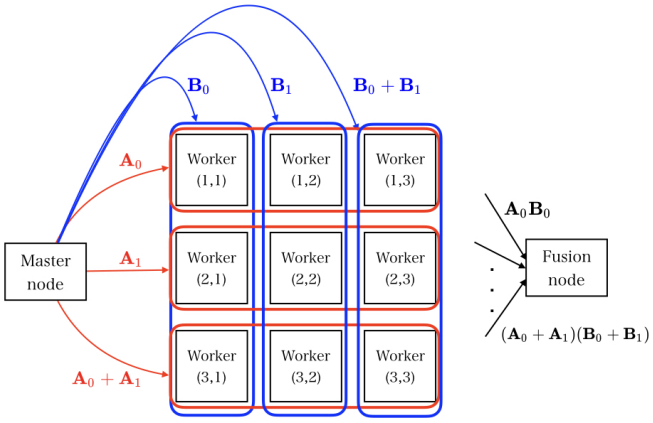

Fig. 2. ABFT matrix multiplication [40] for $P=9$ worker nodes with $m=2$, where $\mathbf{A}=\begin{bmatrix}\mathbf{A}_0\\ \mathbf{A}_1\end{bmatrix}$, $\mathbf{B}=[\mathbf{B}_0 \quad \mathbf{B}_1]$. The recovery threshold is 6.

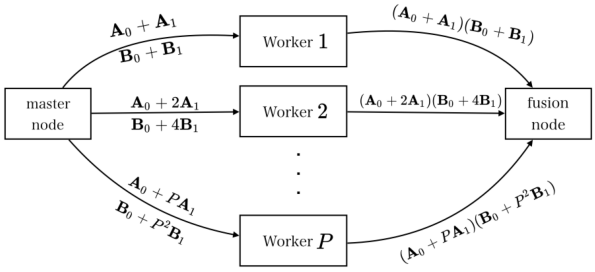

Fig. 3. Polynomial Codes [6] with $m=2$. The recovery threshold is 4.

In the previous example, the recovery threshold was a function of P and thus it requires more successful worker nodes as we use more processors. However, as we will show in the next example, Polynomial codes [6] provide a superior recovery threshold that does not depend on P.

Remark III.1. In the worst-case ABFT codes might require $\Theta(\sqrt{P})$ nodes to finish, but in the best-case only m^2 nodes might suffice, e.g., if all the systematic nodes finish first. Therefore, some specific subsets of nodes of size smaller than the recovery threshold can sometimes suffice for reconstruction, even though not all subsets of this size suffice. For a detailed discussion on best-case and average-case recovery, the reader is referred to [44].

Example III.2. [Polynomial codes [6] ($m=2$, $k=4$)] Consider two $N\times N$ matrices $\mathbf{A}$ and $\mathbf{B}$ that are split as follows:

$$\mathbf{A}=\begin{bmatrix}\mathbf{A}_0\\ \mathbf{A}_1\end{bmatrix}, \mathbf{B}=\begin{bmatrix}\mathbf{B}_0 & \mathbf{B}_1\end{bmatrix}.$$

Polynomial codes compute $\mathbf{AB}$ over P nodes such that, (**i**) each node uses $N^2/2$ linear combinations of the entries of $\mathbf{A}$ and $N^2/2$ linear combinations of the entries of $\mathbf{B}$ and (**ii**) the overall computation is tolerant to $P-4$ stragglers, i.e., any 4 nodes suffice to recover $\mathbf{AB}$. Polynomial codes use the following strategy: Node i computes $(\mathbf{A}_0+\mathbf{A}_1 i)(\mathbf{B}_0+\mathbf{B}_1 i^2), i=1,2,\dots P$, so that from any 4 of the P nodes, the polynomial $p(x)=(\mathbf{A}_0\mathbf{B}_0+\mathbf{A}_1\mathbf{B}_0 x+\mathbf{A}_0\mathbf{B}_1 x^2+\mathbf{A}_0\mathbf{B}_1 x^3)$ can be interpolated. Having interpolated the polynomial, $\mathbf{AB}$ as $\begin{bmatrix}\mathbf{A}_0\mathbf{B}_0 & \mathbf{A}_0\mathbf{B}_1\\ \mathbf{A}_1\mathbf{B}_0 & \mathbf{A}_1\mathbf{B}_1\end{bmatrix}$ can be obtained from the coefficients (matrices) of the polynomial. ■

Our novel *MatDot* construction achieves a smaller recovery threshold as compared with Polynomial codes. Unlike ABFT and Polynomial codes, MatDot divides matrix $\mathbf{A}$ vertically into column-blocks and matrix $\mathbf{B}$ horizontally into row-blocks.

Example III.3. [MatDot codes ($m=2$, $k=3$)]

MatDot codes compute $\mathbf{AB}$ over P nodes such that, (**i**) each node uses $N^2/2$ linear combinations of the entries of $\mathbf{A}$ and $N^2/2$ linear combinations of the entries of $\mathbf{B}$ and (**ii**) the overall computation is tolerant to $P-3$ stragglers, i.e., 3 nodes suffice to recover $\mathbf{AB}$. The proposed MatDot codes use the following strategy: Matrix $\mathbf{A}$ is split vertically and $\mathbf{B}$ is split horizontally as follows:

$$\mathbf{A}=[\mathbf{A}_0 \quad \mathbf{A}_1], \quad \mathbf{B}=\begin{bmatrix}\mathbf{B}_0\\ \mathbf{B}_1\end{bmatrix}, \tag{1}$$

where $\mathbf{A}_0, \mathbf{A}_1$ are column-blocks of $\mathbf{A}$ of dimension $N\times N/2$ and $\mathbf{B}_0, \mathbf{B}_1$ are row-blocks of $\mathbf{B}$ of dimension $N/2\times N$.

Let $p_{\mathbf{A}}(x)=\mathbf{A}_0+\mathbf{A}_1 x$ and $p_{\mathbf{B}}(x)=\mathbf{B}_0 x+\mathbf{B}_1$. Let $x_1, x_2, \cdots, x_P$ be distinct elements in $\mathbb{F}$. The master node sends $p_{\mathbf{A}}(x_r)$ and $p_{\mathbf{B}}(x_r)$ to the r-th worker node where the r-th worker node performs the multiplication $p_{\mathbf{A}}(x_r)p_{\mathbf{B}}(x_r)$ and sends the output to the fusion node. The exact computations at each worker node are depicted in Fig. 4. We can observe that the fusion node can obtain the product $\mathbf{AB}$ using the output of any three successful workers as follows: Let the worker nodes 1, 2, and 3 be the first three successful worker nodes, then the fusion node obtains the following three matrices:

$$p_{\mathbf{A}}(x_1)p_{\mathbf{B}}(x_1)=\mathbf{A}_0\mathbf{B}_1+(\mathbf{A}_0\mathbf{B}_0+\mathbf{A}_1\mathbf{B}_1)x_1+\mathbf{A}_1\mathbf{B}_0 x_1^2,$$
$$p_{\mathbf{A}}(x_2)p_{\mathbf{B}}(x_2)=\mathbf{A}_0\mathbf{B}_1+(\mathbf{A}_0\mathbf{B}_0+\mathbf{A}_1\mathbf{B}_1)x_2+\mathbf{A}_1\mathbf{B}_0 x_2^2,$$
$$p_{\mathbf{A}}(x_2)p_{\mathbf{B}}(x_3)=\mathbf{A}_0\mathbf{B}_1+(\mathbf{A}_0\mathbf{B}_0+\mathbf{A}_1\mathbf{B}_1)x_3+\mathbf{A}_1\mathbf{B}_0 x_3^2.$$

Since these three matrices can be seen as three evaluations of the matrix polynomial $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ of degree 2 at three distinct evaluation points x_1, x_2, x_3, the fusion node can obtain the coefficients of x in $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ using polynomial interpolation. This includes the coefficient of x, which is $\mathbf{A}_0\mathbf{B}_0+\mathbf{A}_1\mathbf{B}_1=\mathbf{AB}$. Therefore, the fusion node can recover the matrix product $\mathbf{AB}$. ■

In the example, we have seen that for $m=2$, the recovery threshold of MatDot codes is $k=3$ which is lower than Polynomial codes as well as ABFT matrix multiplication. The following theorem shows that for any integer m, the recovery threshold of MatDot codes is $k=2m-1$.

Theorem III.1. *For the matrix multiplication problem specified in Section II-B computed on the system defined in*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

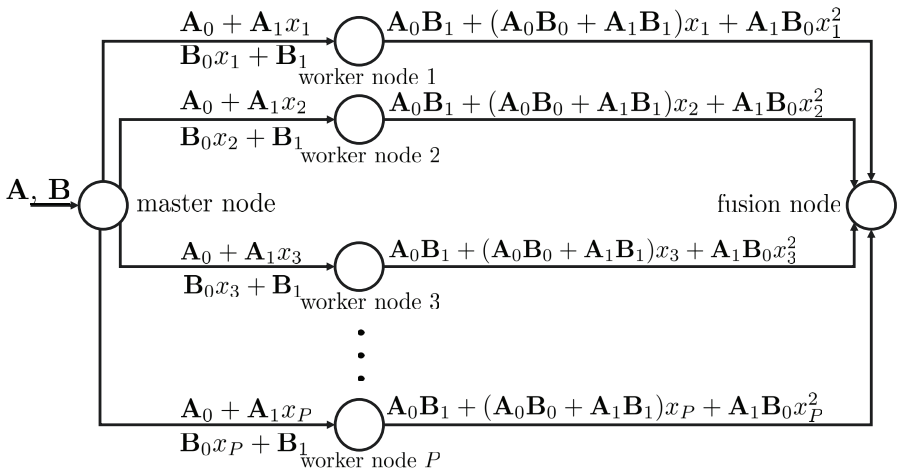

Fig. 4. An illustration of the computational system with four worker nodes and applying MatDot codes with $m = 2$. The recovery threshold is 3.

Definition II.1, a recovery threshold of $2m-1$ is achievable where $m \geq 2$ is a positive integer that divides N.

Before we prove Theorem III.1, we first describe the construction of MatDot codes.

Construction III.1. *[MatDot Codes]*

***Splitting of input matrices**: The matrix $\mathbf{A}$ is split vertically into m equal column-blocks (of N^2/m symbols each) and $\mathbf{B}$ is split horizontally into m equal row blocks (of N^2/m symbols each) as follows:*

$$\mathbf{A} = [\mathbf{A}_0 \ \mathbf{A}_1 \ \ldots \ \mathbf{A}_{m-1}], \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{m-1} \end{bmatrix}, \tag{2}$$

where, for $i \in \{0, \ldots, m-1\}$, and $\mathbf{A}_i, \mathbf{B}_i$ are $N \times N/m$ and $N/m \times N$ dimensional sub-matrices, respectively.

***Master node (encoding)**: Let $x_1, x_2, \ldots, x_P$ be distinct elements in $\mathbb{F}$. Let $p_{\mathbf{A}}(x) = \sum_{i=0}^{m-1} \mathbf{A}_i x^i$ and $p_{\mathbf{B}}(x) = \sum_{j=0}^{m-1} \mathbf{B}_j x^{m-1-j}$. The master node sends to the r-th worker the evaluations of $p_{\mathbf{A}}(x), p_{\mathbf{B}}(x)$ at $x = x_r$, that is, it sends $p_{\mathbf{A}}(x_r), p_{\mathbf{B}}(x_r)$ to the r-th worker.*

***Worker nodes**: For $r \in \{1, 2, \ldots, P\}$, the r-th worker node computes the matrix product $p_{\mathbf{C}}(x_r) = p_{\mathbf{A}}(x_r)p_{\mathbf{B}}(x_r)$ and sends it to the fusion node on successful completion.*

***Fusion node (decoding)**: The fusion node uses outputs of any $2m-1$ successful workers to compute the coefficient of x^{m-1} in the product $p_{\mathbf{C}}(x) = p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ (the feasibility of this step will be shown later in the proof of Theorem III.1). If the number of successful workers is smaller than $2m-1$, the fusion node declares a failure.*

Notice that in MatDot codes, we have

$$\mathbf{AB} = \sum_{i=0}^{m-1} \mathbf{A}_i\mathbf{B}_i, \tag{3}$$

where $\mathbf{A}_i$ and $\mathbf{B}_i$ are as defined in (2). The simple observation of (3) leads to a different way of computing the matrix product as compared with Polynomial-codes-based computation. In particular, to compute the product, we only require, for each i, the product of $\mathbf{A}_i$ and $\mathbf{B}_i$. We do not require products of the form $\mathbf{A}_i\mathbf{B}_j$ for $i \neq j$ unlike Polynomial codes, where, after splitting the matrices $\mathbf{A}, \mathbf{B}$ in to m parts, *all* m^2 cross-products are required to evaluate the overall matrix product. This leads to a significantly smaller recovery threshold for our construction.

Proof of Theorem III.1. To prove the theorem, it suffices to show that in the MatDot code construction described above, the fusion node is able to reconstruct $\mathbf{C}$ from any $2m-1$ worker nodes. Observe that the coefficient of x^{m-1} in:

$$p_{\mathbf{C}}(x) = p_{\mathbf{A}}(x)p_{\mathbf{B}}(x) = \left(\sum_{i=0}^{m-1} \mathbf{A}_i x^i\right)\left(\sum_{j=0}^{m-1} \mathbf{B}_j x^{m-1-j}\right) \tag{4}$$

is $\mathbf{AB} = \sum_{i=0}^{m-1} \mathbf{A}_i\mathbf{B}_i$ (from (3)), which is the desired matrix-matrix product. Thus it is sufficient to compute this coefficient at the fusion node as the computation output for successful computation. Now, because the polynomial $p_{\mathbf{C}}(x)$ has degree $2m-2$, evaluation of the polynomial at any $2m-1$ distinct points is sufficient to compute all of the coefficients of powers of x in $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ using polynomial interpolation. This includes $\mathbf{AB} = \sum_{i=0}^{m-1} \mathbf{A}_i\mathbf{B}_i$, the coefficient of x^{m-1}. ■

In Section III-B, we provide a complexity analysis that shows that using this strategy, the master and fusion nodes have a lower computational complexity as compared to the worker nodes in the regime where $m, P \ll N$.

B. Complexity Analysis of MatDot codes

Encoding/decoding complexity: Encoding for each worker requires evaluating two polynomials $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$, each of degree $m-1$, at a unique value of x where the coefficients of these polynomials are sub-matrices of size N^2/m. We examine the encoding complexity using two algorithms here. One encoding algorithm could be to take a linear

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

combination of m sub-matrices of size N^2/m, leading to an overall encoding complexity of $\mathcal{O}(mN^2/m) = \mathcal{O}(N^2)$ for *each worker*. Thus, the overall computational complexity of encoding for P workers is $\mathcal{O}(N^2P)$. Alternatively, one could also use fast polynomial evaluation algorithms [48], [49] which allow one to evaluate a polynomial (of degree $m-1$) at $P(>m)$ arbitrary points within a time complexity of $\mathcal{O}(P\log^2 m)$ (or more practically $\mathcal{O}(P\log^2 m \log\log m)$). Because this evaluation has to be repeated N^2/m times, the overall encoding complexity using fast polynomial evaluation algorithms becomes $\mathcal{O}\left(N^2P\frac{\log^2 m\log\log m}{m}\right)$.

Next, we examine the decoding complexity. Decoding requires interpolating the coefficient of x^{m-1} (of size N^2) in the polynomial $p_{\mathbf{C}}(x)$ of degree $2m-2$. Because we are interested in only one coefficient of the polynomial $p_{\mathbf{C}}(x)$ and not all of them, we consider the problem of inverting the corresponding Vandermonde matrix for polynomial interpolation and then computing the corresponding coefficient of x^{m-1} separately.

Let $p_{\mathbf{C}}(x) = \mathbf{C}_0+\mathbf{C}_1x+\ldots+\mathbf{C}_{k-1}x^{k-1}$ where $k=2m-1$ and we are interested in interpolating only $\mathbf{C}_{m-1}$. Also, let $\tilde{x}_1,\tilde{x}_2,\ldots,\tilde{x}_k$ denote the $k(=2m-1)$ unique values at which the k fastest workers evaluated the polynomial $p_{\mathbf{C}}(x)$ and $\mathbf{V}$ denote the $k\times k$ Vandermonde matrix given by:

$$\mathbf{V} = \begin{bmatrix} 1 & \tilde{x}_1 & \tilde{x}_1^2 & \ldots & \tilde{x}_1^{k-1} \\ 1 & \tilde{x}_2 & \tilde{x}_2^2 & \ldots & \tilde{x}_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \tilde{x}_k & \tilde{x}_k^2 & \ldots & \tilde{x}_k^{k-1} \end{bmatrix}. \tag{5}$$

Observe that

$$(\mathbf{V}\otimes I_{N\times N})\begin{bmatrix}\mathbf{C}_0\\ \mathbf{C}_1\\ \vdots\\ \mathbf{C}_{k-1}\end{bmatrix} = \begin{bmatrix}p_{\mathbf{C}}(\tilde{x}_1)\\ p_{\mathbf{C}}(\tilde{x}_2)\\ \vdots\\ p_{\mathbf{C}}(\tilde{x}_k)\end{bmatrix}$$

$$\implies \begin{bmatrix}\mathbf{C}_0\\ \mathbf{C}_1\\ \vdots\\ \mathbf{C}_{k-1}\end{bmatrix} = \left(\mathbf{V}^{-1}\otimes I_{N\times N}\right)\begin{bmatrix}p_{\mathbf{C}}(\tilde{x}_1)\\ p_{\mathbf{C}}(\tilde{x}_2)\\ \vdots\\ p_{\mathbf{C}}(\tilde{x}_k)\end{bmatrix}, \tag{6}$$

where $\otimes$ denotes the Kronecker product and $I_{N\times N}$ denotes an identity matrix of dimensions $N\times N$. The decoder first inverts the matrix $\mathbf{V}$ (complexity is at most $\mathcal{O}(k^3)$ using naive inversion algorithms[5]) and then picks the m-th row of $\mathbf{V}^{-1}$ which corresponds to the linear combination of evaluations leading to the coefficient of x^{m-1}. Next, it linearly combines these k evaluations $p_{\mathbf{C}}(\tilde{x}_1),p_{\mathbf{C}}(\tilde{x}_2),\ldots,p_{\mathbf{C}}(\tilde{x}_k)$ (of size N^2 each) using the k values in [m-th row of $\mathbf{V}^{-1}$], effectively performing the computation

$$\mathbf{C}_{m-1} = \left([m\text{-th row of }\mathbf{V}^{-1}]\otimes I_{N\times N}\right)\begin{bmatrix}p_{\mathbf{C}}(\tilde{x}_1)\\ p_{\mathbf{C}}(\tilde{x}_2)\\ \vdots\\ p_{\mathbf{C}}(\tilde{x}_k)\end{bmatrix}.$$

This second step is of complexity $\mathcal{O}(N^2k)$. Thus, the total decoding complexity is $\mathcal{O}(N^2k+k^3)$, of which, the first term dominates as we are interested in regimes where $k(=2m-1)\ll N$.

Each worker's computational cost: Each worker multiplies two matrices of dimensions $N\times N/m$ and $N/m\times N$, requiring N^3/m operations (using standard matrix multiplication algorithms[6]). Hence, the computational complexity for each worker is $\mathcal{O}(N^3/m)$. Thus, as long as P and m are sufficiently small compared to N, *the encoding and decoding complexity is smaller than per-worker computational complexity in a scaling sense.* More specifically, for the decoding complexity to be negligible, we need $m^2 = o(N)$ (derived from $N^2(2m-1)=o(N^3/m)$). Similarly, for the encoding complexity to be negligible, we need $mP=o(N)$ (derived from $N^2P=o(N^3/m)$), again sticking to the conservative estimate of encoding complexity.

Communication cost: The master node communicates $\mathcal{O}(PN^2/m)$ symbols, and the fusion node receives $\mathcal{O}(mN^2)$ symbols from the successful worker nodes. While the master node communication cost is identical to that in Polynomial codes, the fusion node there only receives $\mathcal{O}(m^2N^2/m^2) = \mathcal{O}(N^2)$ symbols.

Remark III.2. We note that in addition to communication costs, the computational cost per node is also higher for MatDot codes ($\mathcal{O}(N^3/m)$) as compared to Polynomial codes ($\mathcal{O}(N^3/m^2)$). This is suggestive of a trade-off. Thus, we also propose PolyDot codes which provide a trade-off between MatDot codes (lowest recovery threshold, higher communication and computation cost) and Polynomial codes (higher recovery threshold, lower communication and computation cost), with these two codes being its two special cases. These trade-offs are also pictorially illustrated later in Fig. 6 and Fig. 7.

Discussion on applicability of MatDot codes:

- In our recent work [56], we demonstrate the potential advantages of MatDot codes in practice. Reference [56] presents a distributed implementation of Fast approximate k-Nearest Neighbor computation using MatDot codes. The problem reduces to the online multiplication of only a set of few selected rows of a large matrix with another matrix/vector in real-time. Encoding and storing sub-matrices in advance is allowed, but the index set of rows of the first matrix is only available in the online phase. It is difficult to apply horizontal splitting in this case as the index set of rows is not known apriori, and vertical splitting of the first matrix, as done in MatDot codes, is better suited.
- In several large-scale computing settings, storage is the primary cause that necessitates parallelizing or distributing the computation across multiple nodes. The actual computation cost is often cheap, and in fact often cheaper than communication costs too. The main cause of latency

[5]Note that, it might be possible to reduce the term k^3 to k^2 using improved methods of inverting Vandermonde matrices [50]–[54]. However, since this is not the dominant term in this decoding complexity analysis, we stick with the most conservative estimate k^3.

[6]More sophisticated algorithms [55] also require super-quadratic complexity in N, and so a similar conclusion can be derived here if those algorithms are used at workers as well, as long as the complexity is super-quadratic in N.

or straggling is attributed to several factors, which also include queuing of other tasks or limitations of communication bandwidth [8], [9]. Thus, the actual time that each worker node takes is a combination of three terms: the delay-free computation cost, the delay-free communication cost and the unpredictable delay or straggling, which could even be higher than the first two terms depending on the nature of the queuing in the system. In several models in existing literature, the total time has also been modeled with distributions which do not depend on the computation cost or communication cost [8], [9]. In such scenarios, MatDot codes would be significantly beneficial in reducing latency as compared to existing techniques as it requires the fusion node to wait for the fewest workers. Alternatively, when the computation and communication costs dominate storage costs, one could use Polynomial codes, or interpolate between these two codes using our proposed PolyDot framework (see Section V).

- MatDot codes can also be written in a systematic form (see Sec. IV), which thereby opens the door for more flexible straggler management strategies. This is discussed in Remark IV.1 in the next section.

C. How do MatDot codes compare with the fundamental limits in [6]?

The statement of [6, Theorem 1] says that when a fixed $1/m$ fraction of the first matrix and $1/n$ fraction of the second matrix are allowed to be stored at each node, the fundamental limit on recovery threshold is mn. When each node is allowed to store a fixed $1/m$ fraction of both the matrices, i.e., $m = n$, the expression of this fundamental limit takes the value m^2. Interestingly, MatDot codes also store only a fixed $1/m$ fraction of each matrix but achieve a *lower* recovery threshold of $2m-1$. This might seem to contradict the fundamental limits of [6].

To understand why this is possible, one needs to carefully examine **the system model and assumptions** in [6, Section 2], and the derivation of the fundamental limit in [6, Theorem 1], which uses a cut-set argument to count the number of bits/symbols required for computing the product $\mathbf{AB}$. In doing so, the authors make the assumption that the number of symbols communicated by each worker to the fusion node is N^2/m^2. This is a fallout of storing an encoded sub-matrix of $\mathbf{A}$ of dimensions $\frac{N}{m} \times N$, and an encoded sub-matrix of $\mathbf{B}$ of dimensions $N \times \frac{N}{m}$ (the opposite of the dimensions used here) and then multiplying the two encoded sub-matrices of dimensions $\frac{N}{m} \times N$ and $N \times \frac{N}{m}$ with each other.

The bound in [6], therefore, *does not apply* to our construction: each worker now communicates N^2 symbols to the fusion node, an outcome of the novel partitioning of the two matrices proposed in this work. Note that, while the amount of information in each worker's transmissions is less, i.e., $\mathcal{O}(N^2/m)$ (because the $N \times N$ matrices communicated by the workers can have rank N^2/m), this is still significantly larger than N^2/m^2 assumption made in the fundamental limits in [6].

From a communication viewpoint, MatDot codes require communicating a total of $(2m-1)N^2$ symbols, which is larger than the N^2 symbols in the product $\mathbf{AB}$. Similarly, the per-node computation cost of MatDot codes is also $\mathcal{O}(N^2/m)$ which is larger than the computation cost of Polynomial codes ($\mathcal{O}(N^2/m^2)$). This is suggestive of a trade-off between minimal number of workers and minimal (sum-rate) communication from non-straggling workers as well as minimum computation per-node. Thus, in Section V, we describe a unified view of MatDot and Polynomial codes, which describes the trade-off between worker-fusion communication cost and per-node computation cost with the recovery threshold.

In practice, whether this increased worker-fusion node communication cost and the increased per-node computation cost using MatDot codes is worth paying for will depend on the specific computational fabric and system implementation choices. Even in systems where communication costs may be significant, it is possible that more communication from fewer successful workers is less expensive than requiring more successful workers as required in Polynomial codes. Also note that if $P = \Omega(m^2)$ (e.g. when the system is highly fault prone or the deadline [42] is very short), communication complexity at the master node will dominate, and hence MatDot codes may not impose a substantial computing overhead.

IV. Systematic MatDot code construction

In this section, we provide a systematic code construction for MatDot codes. As the notion of systematic codes in the context of the matrix multiplication problem is ambiguous, we will first define systematic codes in our context.

Definition IV.1. [Systematic code in distributed matrix-matrix multiplication] For the problem stated in Section II-B computed on the system defined in Definition II.1 such that the matrices $\mathbf{A}$ and $\mathbf{B}$ are split as in (2), a code is called *systematic* if the output of the r-th worker node is the product $\mathbf{A}_{r-1}\mathbf{B}_{r-1}$, for all $r \in \{1, \cdots, m\}$. We refer to the first m worker nodes, that output $\mathbf{A}_{r-1}\mathbf{B}_{r-1}$ for $r \in \{1, \cdots, m\}$, as *systematic worker nodes*.

Note that the final output $\mathbf{AB}$ can be obtained by summing up the outputs from the m systematic worker nodes:

$$\mathbf{AB} = \sum_{r=1}^{m} \mathbf{A}_{r-1}\mathbf{B}_{r-1}.$$

The presented systematic code, named *"systematic MatDot code"*, is advantageous over MatDot codes in two aspects. Firstly, even though both MatDot and systematic MatDot codes have the same recovery threshold, systematic MatDot codes can recover the output as soon as the m systematic worker nodes successfully finish, this is unlike MatDot codes which *always* require $2m-1$ workers to successfully finish to recover the final result. Furthermore, when the m systematic worker nodes successfully finish first, the decoding complexity using systematic MatDot codes is $O(mN^2)$, which is slightly less than the decoding complexity of MatDot codes, i.e., $O(kN^2 + k^3)$ where $k = 2m-1$. Another advantage for systematic MatDot codes over MatDot codes is that the systematic MatDot approach may be useful for *backward-compatibility with current practice*. What this means is that,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

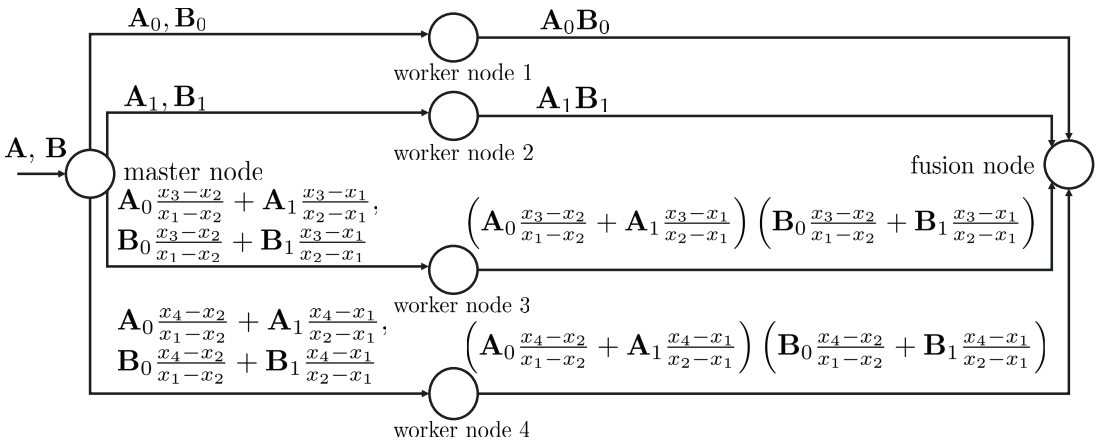

Fig. 5. An illustration of the computational system with four worker nodes and applying systematic MatDot codes with $m = 2$. The recovery threshold is 3.

for systems that are already established and operating with no straggler tolerance, but do an m-way parallelization, it is easier to apply the systematic approach as the infrastructure could be appended to additional worker nodes without modifying what the first m nodes are doing.

The following theorem shows that there exists a systematic MatDot code construction that achieves the same recovery threshold as MatDot codes.

Theorem IV.1. *For the matrix-matrix multiplication problem specified in Section II-B computed on the system defined in Definition II.1, there exists a systematic code, where the product* $\mathbf{AB}$ *is the summation of the output of the first* m *worker nodes, that solves this problem with a recovery threshold of* $2m-1$*, where* $m \geq 2$ *is any positive integer that divides* N.

Before we describe the construction of systematic MatDot codes, that will be used to prove Theorem IV.1, we first present a simple example to illustrate the idea of systematic MatDot codes.

Example IV.1. [Systematic MatDot code, $m = 2, k = 3$]

Matrix $\mathbf{A}$ is split vertically into two sub-matrices (column-blocks) $\mathbf{A}_0$ and $\mathbf{A}_1$, each of dimension $N \times \frac{N}{2}$ and matrix $\mathbf{B}$ is split horizontally into two sub-matrices (row-blocks) $\mathbf{B}_0$ and $\mathbf{B}_1$, each of dimension $\frac{N}{2} \times N$ as follows:

$$\mathbf{A} = [\ \mathbf{A}_0 \quad \mathbf{A}_1\ ], \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix}. \tag{7}$$

Now, we define the encoding functions $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$ as $p_{\mathbf{A}}(x) = \mathbf{A}_0 \frac{x-x_2}{x_1-x_2} + \mathbf{A}_1 \frac{x-x_1}{x_2-x_1}$ and $p_{\mathbf{B}}(x) = \mathbf{B}_0 \frac{x-x_2}{x_1-x_2} + \mathbf{B}_1 \frac{x-x_1}{x_2-x_1}$, for distinct $x_1, x_2 \in \mathbb{F}$. Let $x_3, \cdots, x_P$ be elements of $\mathbb{F}$ such that $x_1, x_2, x_3, \cdots, x_P$ are distinct. The master node sends $p_{\mathbf{A}}(x_r)$ and $p_{\mathbf{B}}(x_r)$ to the r-th worker node, for all $r \in \{1, \cdots, P\}$, where the r-th worker node performs the multiplication $p_{\mathbf{A}}(x_r)p_{\mathbf{B}}(x_r)$ and sends the output to the fusion node. The exact computations at each worker node are depicted in Fig. 5.

We can observe that the outputs of the worker nodes $1, 2$ are $\mathbf{A}_0\mathbf{B}_0, \mathbf{A}_1\mathbf{B}_1$, respectively, and hence this code is systematic. Let us consider a scenario where the systematic worker nodes, i.e., worker nodes 1 and 2, complete their computations first. In this scenario, the fusion node does not require a decoding step and can obtain the product $\mathbf{AB}$ by simply performing the summation of the two outputs it has received: $\mathbf{A}_0\mathbf{B}_0 + \mathbf{A}_1\mathbf{B}_1$. Now, let us consider a different scenario where worker nodes $1, 3, 4$ are the first three successful workers. Then, the fusion node receives three matrices, $p_{\mathbf{A}}(x_1)p_{\mathbf{B}}(x_1), p_{\mathbf{A}}(x_3)p_{\mathbf{B}}(x_3)$, and $p_{\mathbf{A}}(x_4)p_{\mathbf{B}}(x_4)$. Since these three matrices can be seen as three evaluations of the polynomial $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ of degree 2 at three distinct evaluation points x_1, x_3, x_4, the coefficients of the polynomial $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ can be obtained using polynomial interpolation. Finally, to obtain the product $\mathbf{AB}$, we evaluate $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ at $x = x_1, x_2$ and sum them up:

$$p_{\mathbf{A}}(x_1)p_{\mathbf{B}}(x_1) + p_{\mathbf{A}}(x_2)p_{\mathbf{B}}(x_2) = \mathbf{A}_0\mathbf{B}_0 + \mathbf{A}_1\mathbf{B}_1 = \mathbf{AB}.$$

■

We now describe the general construction of the systematic MatDot codes for matrix-matrix multiplication. As all the code constructions in this paper follow the polynomial format given in Construction III.1, in our subsequent constructions, we will only highlight major differences, such as, encoding polynomials.

Construction IV.1. *[Systematic MatDot codes]*

Splitting of input matrices*:* $\mathbf{A}$ *and* $\mathbf{B}$ *are split as in (2).*

Master node (encoding)*: The master node encodes matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *using the following polynomials:*

$$p_{\mathbf{A}}(x) = \sum_{i=1}^{m} \mathbf{A}_{i-1} L_i(x), \quad p_{\mathbf{B}}(x) = \sum_{i=1}^{m} \mathbf{B}_{i-1} L_i(x), \tag{8}$$

where

$$L_i(x) = \prod_{j \in \{1,\cdots,m\} \setminus \{i\}} \frac{x - x_j}{x_i - x_j}. \tag{9}$$

Fusion node (decoding)*: For any* k *such that* $m \leq k \leq 2m-1$*, whenever the outputs of the first* k *successful workers contain the outputs of the systematic worker nodes* $1, \cdots, m$*, i.e.,* $\{p_{\mathbf{C}}(x_r)\}_{r \in \{1,\cdots,m\}}$ *is contained in the set of the first* k *outputs received by the fusion node, the fusion node performs the summation* $\sum_{r=1}^{m} p_{\mathbf{C}}(x_r)$*. Otherwise, if* $\{p_{\mathbf{C}}(x_r)\}_{r \in \{1,\cdots,m\}}$ *is not contained in the set of the first*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

$2m-1$ *evaluations received by the fusion node, the fusion node performs the following steps: (i) interpolates the polynomial* $p_{\mathbf{C}}(x) = p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ *(the feasibility of this step will be shown later in the proof of Theorem IV.1), (ii) evaluates* $p_{\mathbf{C}}(x)$ *at* $x_1, \cdots, x_m$*, (iii) performs the summation* $\sum_{r=1}^{m} p_{\mathbf{C}}(x_r)$.

If the number of successful worker nodes is smaller than $2m-1$ *and the first* m *worker nodes are not included in the successful worker nodes, the fusion node declares a failure.*

The following lemma proves that the construction given here is systematic.

Lemma IV.1. *For Construction IV.1, the output of the* r*-th worker node, for* $r \in \{1, \cdots, m\}$*, is the product* $\mathbf{A}_{r-1}\mathbf{B}_{r-1}$*. That is, Construction IV.1 is a systematic code for distributed matrix-matrix multiplication as defined in Definition IV.1*

Proof of Lemma IV.1. The lemma follows from the fact that $p_{\mathbf{A}}(x_r) = \mathbf{A}_{r-1}$, and $p_{\mathbf{B}}(x_r) = \mathbf{B}_{r-1}$, for $r \in \{1, \cdots, m\}$. Thus, $p_{\mathbf{C}}(x_r) = p_{\mathbf{A}}(x_r)p_{\mathbf{B}}(x_r) = \mathbf{A}_{r-1}\mathbf{B}_{r-1}$, for any $r \in \{1, \cdots, m\}$. ■

Now, we proceed with the proof of Theorem IV.1.

Proof of Theorem IV.1. Since Construction IV.1 is a systematic code for matrix-matrix multiplication (Lemma IV.1), in order to prove the theorem, it suffices to show that Construction IV.1 is a valid construction with a recovery threshold $k = 2m-1$. From (9), observe that the polynomials $L_i(x)$, $i \in \{1, \cdots, m\}$, have degrees $m-1$ each. Therefore, each of $p_{\mathbf{A}}(x) = \sum_{i=1}^{m} \mathbf{A}_{i-1}L_i(x)$ and $p_{\mathbf{B}}(x) = \sum_{i=1}^{m} \mathbf{B}_{i-1}L_i(x)$ has a degree of $m-1$ as well. Consequently, $p_{\mathbf{C}}(x) = p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ has a degree of $2m-2$. Now, because the polynomial $p_{\mathbf{C}}(x)$ has degree $2m-2$, evaluation of the polynomial at any $2m-1$ distinct points is sufficient to interpolate $\mathbf{C}(x)$ using polynomial interpolation algorithm. Now, since Construction IV.1 is systematic (Lemma IV.1), the product $\mathbf{AB}$ is the summation of the outputs of the first m workers, i.e., $\mathbf{AB} = \sum_{r=1}^{m} p_{\mathbf{C}}(x_r)$. Therefore, after the fusion node interpolates $\mathbf{C}(x)$, evaluating $p_{\mathbf{C}}(x)$ at $x_1, \cdots, x_m$, and performing the summation $\sum_{r=1}^{m} p_{\mathbf{C}}(x_r)$ yields the product $\mathbf{AB}$. ■

A. Complexity Analysis of Systematic MatDot codes

Apart from the encoding/decoding complexity, the complexity analyses of systematic MatDot codes are the same as their MatDot counterpart. In the following, we investigate the encoding/decoding complexity of Construction IV.1.

Encoding/Decoding Complexity: Encoding for each worker first requires performing evaluations of polynomials $L_i(x)$ for all $i \in \{1, \cdots, m\}$, with each evaluation requiring $\mathcal{O}(m)$ operations. This gives $\mathcal{O}(m^2)$ operations for all polynomial evaluations. Afterwards, two linear combinations of m sub-matrices of size N^2/m each is taken, which is of complexity $\mathcal{O}(mN^2/m) = \mathcal{O}(N^2)$. Therefore, the overall encoding complexity for *each non-systematic worker* is $\mathcal{O}(\max(N^2, m^2)) = \mathcal{O}(N^2)$ because $m \ll N$. For the systematic workers, no further encoding is required on the sub-matrices of $\mathbf{A}$ and $\mathbf{B}$. Thus, the overall computational complexity of encoding for P workers is $\mathcal{O}(N^2(P-m))$. This is similar to the encoding for MatDot codes given in Section III.

For decoding, two cases would arise depending on whether all the m systematic nodes finished first or not. When all the m systematic nodes finish first, the decoding is equivalent to taking the sum of the m systematic evaluations and is thus of complexity $\mathcal{O}(N^2m)$. Alternatively, when the m systematic nodes do not finish first, the decoder waits for the first $k(=2m-1)$ nodes to send their evaluations of $p_{\mathbf{C}}(x)$. Then it is required to interpolate the coefficients of $p_{\mathbf{C}}(x)$, evaluate it at the systematic points $x_1, x_2, \ldots, x_m$, and then take the sum of the systematic evaluations. Because we are interested in only the final sum of the systematic evaluations and not in the individual systematic evaluations or coefficient interpolations, we again consider the problem of deriving the appropriate linear combination and taking the final linear combination on the matrices separately.

Recall from Section III-B that $p_{\mathbf{C}}(x) = \mathbf{C}_0 + \mathbf{C}_1 x + \ldots + \mathbf{C}_{k-1}x^{k-1}$ where $k = 2m-1$ but now we are interested in computing the sum of the systematic evaluations of $p_{\mathbf{C}}(x)$ at $x_1, x_2, \ldots, x_m$. Also let $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k$ denote the $k(=2m-1)$ unique values at which the k fastest workers evaluated the polynomial $p_{\mathbf{C}}(x)$ and $\mathbf{V}$ denote the $k \times k$ Vandermonde matrix as defined in (5). Recall that,

$$\begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_{k-1} \end{bmatrix} = \left(\mathbf{V}^{-1} \otimes I_{N\times N}\right) \begin{bmatrix} p_{\mathbf{C}}(\tilde{x}_1) \\ p_{\mathbf{C}}(\tilde{x}_2) \\ \vdots \\ p_{\mathbf{C}}(\tilde{x}_k) \end{bmatrix}.$$

Let $\hat{\mathbf{V}}$ denote the $m \times k$ Vandermonde matrix for evaluation, consisting of increasing powers of the m systematic values $x_1, x_2, \ldots, x_m$, as follows:

$$\hat{\mathbf{V}} = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{k-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \ldots & x_m^{k-1} \end{bmatrix}.$$

Now, the evaluation of $p_{\mathbf{C}}(x)$ at the systematic values $x_1, x_2, \ldots, x_m$ is equivalent to the following operation:

$$\left(\hat{\mathbf{V}} \otimes I_{N\times N}\right) \begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_{k-1} \end{bmatrix} = \left((\hat{\mathbf{V}}\mathbf{V}^{-1}) \otimes I_{N\times N}\right) \begin{bmatrix} p_{\mathbf{C}}(\tilde{x}_1) \\ p_{\mathbf{C}}(\tilde{x}_2) \\ \vdots \\ p_{\mathbf{C}}(\tilde{x}_k) \end{bmatrix}.$$

Finally, the summation of these m systematic evaluations can be written as:

$$([1,1,\ldots,1]_{1\times m} \otimes I_{N\times N}) \left(\hat{\mathbf{V}} \otimes I_{N\times N}\right) \begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_{k-1} \end{bmatrix}$$

$$= \left(([1,1,\ldots,1]_{1\times m}\hat{\mathbf{V}}\mathbf{V}^{-1}) \otimes I_{N\times N}\right) \begin{bmatrix} p_{\mathbf{C}}(\tilde{x}_1) \\ p_{\mathbf{C}}(\tilde{x}_2) \\ \vdots \\ p_{\mathbf{C}}(\tilde{x}_k) \end{bmatrix}.$$

The decoder first computes the final row-vector $([1,1,\ldots,1]_{1\times m}\hat{\mathbf{V}}\mathbf{V}^{-1})$ (complexity is at most $\mathcal{O}(k^3)$ as it is

dominated by the inversion of the matrix $\mathbf{V}$). Next, it linearly combines the k evaluations $p_{\mathbf{C}}(\tilde{x}_1), p_{\mathbf{C}}(\tilde{x}_2), \ldots, p_{\mathbf{C}}(\tilde{x}_k)$ (of size N^2 each) using the k values in the final row vector (complexity is $\mathcal{O}(N^2k)$). Thus, the total decoding complexity is $\mathcal{O}(N^2k + k^3) = \mathcal{O}(N^2k)$ when $k(= 2m-1) \ll N$. This is similar to MatDot codes.

Note that, these encoding and decoding complexities may be improved further in functions of m and P in different scenarios, e.g., using alternate methods of faster evaluation, or using the outputs of the systematic nodes more efficiently during decoding if at least some of them are in the set of k fastest workers (if not all) that will be pursued as a future work. Here, we restrict ourselves to somewhat conservative estimates for our proposed strategy as our main goal is to explore dependence on N in the regime where $m, P \ll N$.

Remark IV.1. The flexibility offered by systematic MatDot codes makes them more amenable to straggler mitigation in practice. In several modern distributed computing systems [57], [58], stragglers are handled in a "reactive" manner where the computation is initially performed over a number of workers without redundancy. The strategy of issuing redundant computations is *reactively speculated* based on the delay pattern of the systematic workers. Specifically, after a certain time, the master or fusion node speculates which nodes are straggling based on how much of the computation they have performed so far, and then issues clones or redundant computations corresponding to these speculated stragglers. After this, the master or fusion node waits for a sufficient set of nodes to finish out of the total nodes that are still computing, which includes the speculated stragglers as well as the newly issued redundant workers. Reactive schemes, where nodes that appear to be straggling, are selectively replicated have been widely studied in distributed computing systems research under the term "speculative execution" (See, e.g., [59], [60] and references therein). Since reactive strategies have fewer unnecessary redundant computations, they incur lower computation overhead in the system than *proactive* strategies, where all the computations (including the redundant ones) are issued at the beginning, independent of the run-time delay pattern of worker nodes.

An advantage of systematic MatDot codes, unlike their non-systematic counterpart, is that they may be used in reactive straggler mitigation strategies, thus lowering the overall computation burden on the system. As an example, consider the case where the m systematic (uncoded) worker nodes issue their computation as follows: for $i \in \{1, 2, \ldots, m\}$, the i-th systematic worker node receives $\mathbf{A}_i$ and $\mathbf{B}_i$, and computes the product $\mathbf{A}_i\mathbf{B}_i$. A centralized scheduling node (*e.g.*, master or fusion node) waits for a prespecified time after which it speculates that the workers that did not complete their computation are stragglers. Suppose that the speculated set of straggling systematic nodes are $\mathcal{R} = \{r_1, \cdots, r_s\} \subseteq \{1, \cdots, m\}$. The master node reactively applies systematic MatDot codes only on the inputs of the speculated stragglers. That is, $\ell > s-1$ redundant parity computations are encoded using systematic MatDot for $\mathbf{A}_j, \mathbf{B}_j, j \in \mathcal{R}$. Among the worker nodes that are still computing (the s systematic nodes in $\mathcal{R}$ and the ℓ newly-issued redundant computations), the fusion node can recover the output from any $2s-1$ worker nodes to obtain $\sum_i^s \mathbf{A}_{r_i}\mathbf{B}_{r_i}$, and hence compute the product $\mathbf{AB}$ by adding the result to the outputs of the non-straggler systematic worker nodes.

In contrast, a replication-based scheme would issue one or more clones of each of these s straggling jobs. Suppose the replication strategy issues ℓ (where s divides ℓ) redundant jobs that have $\frac{\ell}{s}$ clones of each of the s original jobs. This strategy would require certain specific subsets of s workers (among the ℓ clones and the s original jobs) to complete the job but it cannot recover the output from *every* subset of $2s-1$ workers. In the worst case, the replication strategy might have to wait for $l + s - \frac{l}{s}$ nodes to finish which is its recovery threshold.

It must be noted that it is unclear whether the replication-based scheme or the systematic MatDot scheme would lead to greater speed-up. We suspect that a good engineering solution would require a careful choice based on the available resources, e.g., number of redundant nodes ℓ, system response characteristics etc. A detailed performance comparison of replication versus systematic MatDot via reactive straggler mitigation is an open question worthy of future systems performance evaluation research.

V. Unifying MatDot and Polynomial Codes: Trade-off between Per-Worker Computation/Communication Costs and Recovery Threshold

In this section, we present a code construction, named *PolyDot*, that provides a trade-off between per-worker computation/communication costs and recovery thresholds. Polynomial codes [6] have a higher recovery threshold of m^2, but have a lower per-worker computation cost of $\mathcal{O}(N^3/m^2)$ and communication cost of $\mathcal{O}(N^2/m^2)$ per worker node. On the other hand, MatDot codes have a lower recovery threshold of $2m-1$, but have a higher per-worker computation cost of $\mathcal{O}(N^3/m)$ and a higher communication cost of $\mathcal{O}(N^2)$ per-worker. This section constructs a code that bridges the gap between Polynomial codes and MatDot codes so that we can get intermediate per-worker computation/communication costs and recovery thresholds, with Polynomial and MatDot codes being two special cases. To achieve this goal, we propose PolyDot codes, which may be viewed as an interpolation of MatDot codes and Polynomial codes, with one extreme being MatDot codes and the other extreme being Polynomial codes.

We follow the same problem setup and system assumptions in II-B. In the following theorem, we obtain the recovery threshold achieved by PolyDot codes.

Theorem V.1. *For the matrix multiplication problem specified in Section II-B computed on the system defined in Definition II.1, there exist codes with a recovery threshold of $t^2(2s-1)$ and a communication cost from each worker node to the fusion node bounded by $\mathcal{O}(N^2/t^2)$ for any positive integers s, t such that $st = m$ and both s and t divide N.*

Before we move on to describe the PolyDot code construction and prove Theorem V.1, we first introduce PolyDot codes with a simple example for $m = 4$ and $s = t = 2$.

Example V.1. [PolyDot codes ($m = 4, s = 2, k = 12$)]

Matrix $\mathbf{A}$ is split into sub-matrices $\mathbf{A}_{0,0}, \mathbf{A}_{0,1}, \mathbf{A}_{1,0}, \mathbf{A}_{1,1}$, each of dimension $N/2 \times N/2$. Similarly, matrix $\mathbf{B}$ is split into sub-matrices $\mathbf{B}_{0,0}, \mathbf{B}_{0,1}, \mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ each of dimension $N/2 \times N/2$ as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} \\ \mathbf{B}_{1,0} & \mathbf{B}_{1,1} \end{bmatrix}. \tag{10}$$

Notice that, from (10), the product $\mathbf{AB}$ can be written as

$$\mathbf{AB} = \begin{bmatrix} \sum_{i=0}^{1} \mathbf{A}_{0,i}\mathbf{B}_{i,0} & \sum_{i=0}^{1} \mathbf{A}_{0,i}\mathbf{B}_{i,1} \\ \sum_{i=0}^{1} \mathbf{A}_{1,i}\mathbf{B}_{i,0} & \sum_{i=0}^{1} \mathbf{A}_{1,i}\mathbf{B}_{i,1} \end{bmatrix}. \tag{11}$$

Now, we define the encoding functions $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$ as

$$p_{\mathbf{A}}(x) = \mathbf{A}_{0,0} + \mathbf{A}_{1,0}x + \mathbf{A}_{0,1}x^2 + \mathbf{A}_{1,1}x^3,$$

$$p_{\mathbf{B}}(x) = \mathbf{B}_{0,0}x^2 + \mathbf{B}_{1,0} + \mathbf{B}_{0,1}x^8 + \mathbf{B}_{1,1}x^6.$$

Observe the following:

(i) the coefficient of x^2 in $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ is $\sum_{i=0}^{1} \mathbf{A}_{0,i}\mathbf{B}_{i,0}$,
(ii) the coefficient of x^8 in $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ is $\sum_{i=0}^{1} \mathbf{A}_{0,i}\mathbf{B}_{i,1}$,
(iii) the coefficient of x^3 in $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ is $\sum_{i=0}^{1} \mathbf{A}_{1,i}\mathbf{B}_{i,0}$, and
(iv) the coefficient of x^9 in $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ is $\sum_{i=0}^{1} \mathbf{A}_{1,i}\mathbf{B}_{i,1}$.

Let $x_1, \cdots, x_P$ be distinct elements of $\mathbb{F}$. The master node sends $p_{\mathbf{A}}(x_r)$ and $p_{\mathbf{B}}(x_r)$ to the r-th worker node for $r \in \{1, \cdots, P\}$. The r-th worker node performs the multiplication $p_{\mathbf{A}}(x_r)p_{\mathbf{B}}(x_r)$ and sends the result to the fusion node.

Let worker nodes indexed from 1 to 12 be the first 12 worker nodes that send their results to the fusion node. Then the fusion node obtains the matrices $p_{\mathbf{A}}(x_r)p_{\mathbf{B}}(x_r)$ for all $r \in \{1, \cdots, 12\}$. Since these 12 matrices are essentially twelve distinct evaluations of the matrix polynomial $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ of degree 11 at twelve distinct points $x_1, \cdots, x_{12}$, the coefficients of the matrix polynomial $p_{\mathbf{A}}(x)p_{\mathbf{B}}(x)$ can be obtained using polynomial interpolation. This includes the coefficients of x^{i+2+6j} for all $i, j \in \{0, 1\}$, i.e., $\sum_{k=0}^{1} \mathbf{A}_{i,k}\mathbf{B}_{k,j}$ for all $i, j \in \{0, 1\}$. Once the matrices $\sum_{k=0}^{1} \mathbf{A}_{i,k}\mathbf{B}_{k,j}$ for all $i, j \in \{0, 1\}$ are obtained, the product $\mathbf{AB}$ is obtained by (11). ■

The recovery threshold for $m = 4$ in Example V.1 is $k = 12$. This is larger than the recovery threshold of MatDot codes, which is $k = 2m - 1 = 9$, and smaller then the recovery threshold of Polynomial codes, which is $k = m^2 = 16$. Hence, we can see that the recovery thresholds of PolyDot codes are between those of MatDot codes and Polynomial codes.

Construction V.1 describes the general construction of PolyDot(m, s, t) codes. Note that, although two parameters m and s are sufficient to characterize a PolyDot code, we include t in the parameters for better readability.

Construction V.1. *[PolyDot(m, s, t) codes]*

Splitting of input matrices: *$\mathbf{A}$ and $\mathbf{B}$ are split both horizontally and vertically:*

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \cdots & \mathbf{A}_{0,s-1} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{t-1,0} & \cdots & \mathbf{A}_{t-1,s-1} \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{0,0} & \cdots & \mathbf{B}_{0,t-1} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{s-1,0} & \cdots & \mathbf{B}_{s-1,t-1} \end{bmatrix}, \tag{12}$$

where, for $i = 0, \cdots, s-1, j = 0, \cdots, t-1$, $\mathbf{A}_{j,i}$'s are $N/t \times N/s$ sub-matrices of $\mathbf{A}$ and $\mathbf{B}_{i,j}$'s are $N/s \times N/t$ sub-matrices of $\mathbf{B}$. We choose s and t such that both s and t divide N and $st = m$.

Master node (encoding): *Define the encoding polynomials as:*

$$p_{\mathbf{A}}(x, y) = \sum_{i=0}^{t-1} \sum_{j=0}^{s-1} \mathbf{A}_{i,j}x^i y^j,$$
$$p_{\mathbf{B}}(y, z) = \sum_{k=0}^{s-1} \sum_{l=0}^{t-1} \mathbf{B}_{k,l}y^{s-1-k}z^l. \tag{13}$$

The master node sends the evaluations of $p_{\mathbf{A}}(x, y), p_{\mathbf{B}}(y, z)$ at $x = x_r, y = x_r^t, z = x_r^{t(2s-1)}$ to the r-th worker where x_r's are all distinct for $r \in \{1, 2, \ldots, P\}$. By this substitution, we are transforming the three-variable polynomial to a single-variable polynomial as follows[7]*:*

$$p_{\mathbf{C}}(x, y, z) = p_{\mathbf{C}}(x) = \sum_{i,j,k,l} \mathbf{A}_{i,j}\mathbf{B}_{k,l}x^{i+t(s-1+j-k)+t(2s-1)l},$$

and evaluate the polynomial $p_{\mathbf{C}}(x)$ at x_r for $r = 1, \cdots, P$. In Lemma V.1, we show that this transformation is one-to-one.

Worker nodes: *For $r \in \{1, 2, \ldots, P\}$, the r-th worker node computes the matrix product $p_{\mathbf{C}}(x_r, y_r, z_r) = p_{\mathbf{A}}(x_r, y_r)p_{\mathbf{B}}(y_r, z_r)$ and sends it to the fusion node on successful completion.*

Fusion node (decoding): *The fusion node uses outputs of the first $t^2(2s-1)$ successful workers to compute the coefficient of $x^{i-1}y^{s-1}z^{l-1}$ in $p_{\mathbf{C}}(x, y, z) = p_{\mathbf{A}}(x, y)p_{\mathbf{B}}(y, z)$, i.e., it computes the coefficient of $x^{i-1+(s-1)t+(2s-1)t(l-1)}$ of the transformed single-variable polynomial. The proof of Theorem V.1 shows that this is indeed possible. If the number of successful workers is smaller than $t^2(2s-1)$, the fusion node declares a failure.*

Discussion on applicability of PolyDot codes: Before we prove the theorem, let us discuss the utility of PolyDot codes. Under a fixed storage constraint ($1/m$), as t increases and s decreases while keeping $st(= m)$ fixed, the recovery threshold keeps increasing and the computation and communication costs keep decreasing. By choosing different s and t, we can trade off communication/computation cost and recovery threshold. For $s = m$ and $t = 1$, PolyDot($m, s = m, t = 1$) code is a MatDot code which has a low recovery threshold

[7] An alternate substitution can reduce the recovery threshold further as mentioned in subsequent works [45], [46]. We will clarify this in Remark V.1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

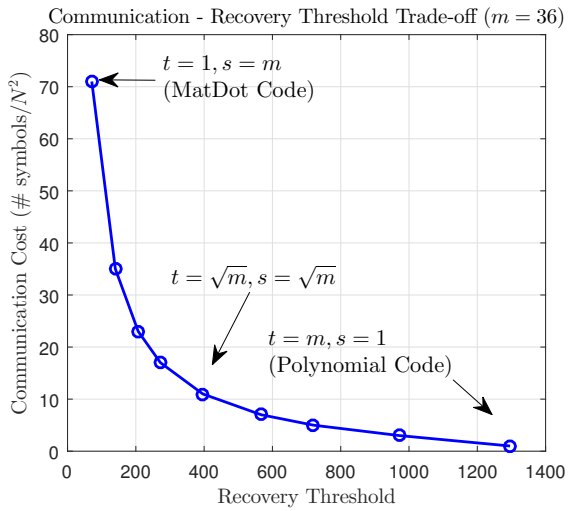

Fig. 6. An illustration of the trade-off between communication cost (from the workers to the fusion node) and the recovery threshold of PolyDot codes by varying s and t for a fixed m ($m = 36$). The minimum communication cost is N^2, corresponding to polynomial codes, that have the largest recovery threshold. It is important to note here that in the above, we are *only including the communication cost from the workers to the fusion node*. The communication from the master node to the workers is not included, and it can dominate in situations when the workers are highly unreliable.

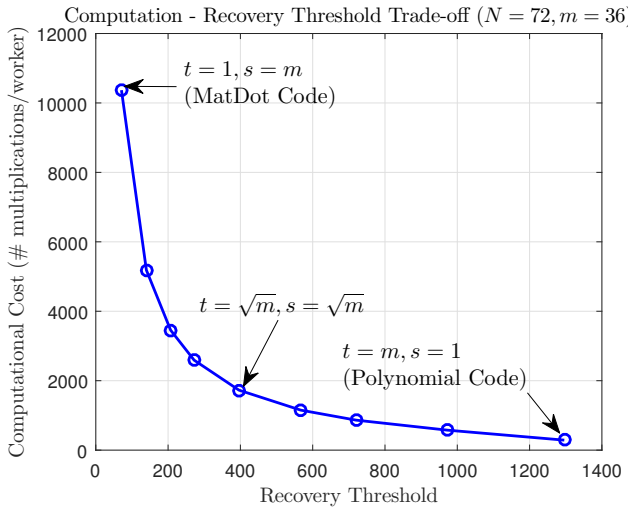

Fig. 7. An illustration of the trade-off between the computation cost per worker and the recovery threshold of PolyDot codes by varying s and t for a fixed N, m ($N = 72, m = 36$). The minimum computation cost per worker is 288 multiplication operations per worker, corresponding to polynomial codes, that have the largest recovery threshold.

but a high communication/computation cost. At the other extreme, for $s = 1$ and $t = m$, PolyDot$(m, s = 1, t = m)$ code is a Polynomial Code. Now, let us consider a code with intermediate s and t values, such as, $s = \sqrt{m}$ and $t = \sqrt{m}$. A PolyDot$(m, s = \sqrt{m}, t = \sqrt{m})$ code has a recovery threshold of $m(2\sqrt{m} - 1) = \Theta(m^{1.5})$, and the total number of symbols to be communicated to the fusion node is $\Theta\left((N/\sqrt{m})^2 \cdot m^{1.5}\right) = \Theta(\sqrt{m}N^2)$, which is smaller than $\Theta(mN^2)$ as required by MatDot codes but larger than $\Theta(N^2)$ as required by Polynomial codes. This trade-off between communication cost and recovery threshold is illustrated in Fig. 6 for $m = 36$. Similarly, in terms of computational cost per worker node, a PolyDot$(m, s = \sqrt{m}, t = \sqrt{m})$ code requires $\mathcal{O}(N^3/m^{1.5})$ operations, which is less than the $\mathcal{O}(N^3/m)$ operations required by MatDot codes but higher than the $\mathcal{O}(N^3/m^2)$ operations required by Polynomial codes. This trade-off between the computation per worker and the recovery threshold is illustrated in Fig. 7 for $N = 72, m = 36$.

In regimes where the storage-constraint is more critical than the computation or communication time, PolyDot codes with the MatDot configuration (or at least closer to MatDot codes, *i.e.*, higher s, lower t) is more appropriate. Alternatively, in settings where computation and communication time dominate significantly, PolyDot codes with Polynomial codes configuration (or at least close to Polynomial codes, *i.e.*, higher t, lower s) may be more preferable. Interestingly though, even in systems where communication costs may be significant, it is possible that more communication from fewer successful workers is less expensive than requiring more successful workers as required in Polynomial codes, which we hope to explore experimentally in future work.

Now, we proceed to prove Theorem V.1. We need the following lemma.

Lemma V.1. *The following function*

$$\begin{aligned} f :&\{0, \cdots, t-1\} \times \{0, \cdots, 2s-2\} \times \{0, \cdots, t-1\} \\ &\to \{0, \cdots, t^2(2s-1) - 1\} \\ &(\alpha, \beta, \gamma) \mapsto \alpha + t\beta + t(2s-1)\gamma \end{aligned} \tag{14}$$

is a bijection.

Proof. Let us assume, for the sake of contradiction, that for some $(\alpha', \beta', \gamma') \neq (\alpha, \beta, \gamma)$, $f(\alpha', \beta', \gamma') = f(\alpha, \beta, \gamma)$. Then $(f(\alpha, \beta, \gamma) \bmod t) = \alpha = (f(\alpha', \beta', \gamma') \bmod t) = \alpha'$ and hence $\alpha = \alpha'$. Similarly, $(f(\alpha, \beta, \gamma) \bmod t(2s-1)) = (f(\alpha', \beta', \gamma') \bmod t(2s-1))$ gives $\alpha + t\beta = \alpha' + t\beta'$, and thus $\beta = \beta'$ (because $\alpha = \alpha'$). Now, because $\alpha = \alpha'$ and $\beta = \beta'$, as we just established, $f(\alpha, \beta, \gamma) = f(\alpha', \beta', \gamma')$ from our assumption, it follows that $\gamma = \gamma'$. This contradicts our assumption that $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$. ∎

Proof of Theorem V.1. The product of $p_{\mathbf{A}}(x, y)$ and $p_{\mathbf{B}}(y, z)$ can be written as follows:

$$\begin{aligned} p_{\mathbf{C}}(x, y, z) &= p_{\mathbf{A}}(x, y) p_{\mathbf{B}}(y, z) \\ &= \left(\sum_{i=0}^{t-1} \sum_{j=0}^{s-1} \mathbf{A}_{i,j} x^i y^j \right) \left(\sum_{k=0}^{s-1} \sum_{l=0}^{t-1} \mathbf{B}_{k,l} y^{s-1-k} z^l \right) \\ &= \sum_{i,j,k,l} \mathbf{A}_{i,j} \mathbf{B}_{k,l} x^i y^{s-1+j-k} z^l. \end{aligned} \tag{15}$$

Note that the coefficient of $x^{i-1} y^{s-1} z^{l-1}$ in $p_{\mathbf{C}}(x, y, z)$ is equal to $\mathbf{C}_{i,l} = \sum_{k=0}^{s-1} \mathbf{A}_{i,k} \mathbf{B}_{k,l}$. By our choice of $y = x^t$ and $z = x^{t(2s-1)}$ we can further simplify $p_{\mathbf{C}}(x, x^t, x^{t(2s-1)})$:

$$p_{\mathbf{C}}(x, y, z) = p_{\mathbf{C}}(x) = \sum_{i,j,k,l} \mathbf{A}_{i,j} \mathbf{B}_{k,l} x^{i+t(s-1+j-k)+t(2s-1)l}. \tag{16}$$

The maximum degree of this polynomial is when $i = t-1, j-k = s-1$ and $l = t-1$, which is $(t-1) + (2s-2)t + t(2s-1)(t-1) = t^2(2s-1) - 1$.

Furthermore, if we let $\alpha = i, \beta = s-1+j-k, \gamma = l$, the function $f(\alpha,\beta,\gamma)$ in Lemma V.1 is the degree of x in (16). This implies that for different pairs of $(i, j-k, l)$, we get different powers of x. When $j-k=0$, we obtain $(\sum_{k=0}^{s-1}\mathbf{A}_{i,k}\mathbf{B}_{k,l})x^{i+t(s-1)+t(2s-1)l} = \mathbf{C}_{i,l}x^{i+t(s-1)+t(2s-1)l}$ which is the desired product we want to recover.

This implies that if we have $t^2(2s-1)$ successful worker nodes, we can compute all the coefficients in (16) by polynomial interpolation. Hence, we can recover all $\mathbf{C}_{i,l}$'s, i.e., the coefficients of $x^{i+t(s-1)+t(2s-1)l}$, for $i,l=0,\cdots,t-1$. ■

Remark V.1. PolyDot codes essentially introduce a general framework which transforms the matrix-matrix multiplication problem into a polynomial interpolation problem with three variables x, y, z. For the PolyDot codes proposed in the initial version of this work [1], we used the substitution $y = x^t$ and $z = x^{t(2s-1)}$ to convert the polynomial in three variables to a polynomial in a single variable, and obtained the recovery threshold of $t^2(2s-1)$. However, based on subsequent works [45], [46], by using a different substitution, $x = y^t, z = y^{st}$, a smaller recovery threshold of st^2+s-1 can be achieved for this problem. This is an improvement within a factor of 2.

Remark V.2. We first introduce the novel PolyDot framework for *matrix-matrix* multiplication which block-partitions the two matrices $\mathbf{A}$ and $\mathbf{B}$ into $t \times s$ and $s \times t$ respectively, using two multivariate polynomials:

$$p_{\mathbf{A}}(x,y) = \sum_{i=0}^{t-1}\sum_{j=0}^{s-1}\mathbf{A}_{i,j}x^i y^j,$$
$$p_{\mathbf{B}}(y,z) = \sum_{k=0}^{s-1}\sum_{l=0}^{t-1}\mathbf{B}_{k,l}y^{s-1-k}z^l. \quad (17)$$

It is trivial to see that for an asymmetric partitioning, e.g., where $\mathbf{A}$ is split in $t_1 \times s$ and $\mathbf{B}$ is split in $s \times t_2$ blocks, the encoding polynomials in the PolyDot framework change as:

$$p_{\mathbf{A}}(x,y) = \sum_{i=0}^{t_1-1}\sum_{j=0}^{s-1}\mathbf{A}_{i,j}x^i y^j,$$
$$p_{\mathbf{B}}(y,z) = \sum_{k=0}^{s-1}\sum_{l=0}^{t_2-1}\mathbf{B}_{k,l}y^{s-1-k}z^l. \quad (18)$$

In this work, the novelty lies in cleverly choosing $p_{\mathbf{A}}(x,y)$ and $p_{\mathbf{B}}(y,z)$, such that, in the product of the two multivariate polynomials, i.e., in $p_{\mathbf{C}}(x,y,z)(= p_{\mathbf{A}}(x,y)p_{\mathbf{B}}(y,z))$ some coefficients correspond to parts of the required resultant matrix $\mathbf{AB}$. After this, we convert the multivariate polynomial $p_{\mathbf{C}}(x,y,z)$ into a polynomial of a single variable in [1] using a substitution which *preserves bijection between all the coefficients (including the ones that are not required).*

Because only some of the coefficients of $p_{\mathbf{C}}(x,y,z)$ are actually required for reconstructing $\mathbf{AB}$, it is not necessary to preserve bijection between all the coefficients in the polynomial of a single variable. In subsequent works [45], [46] a lower recovery threshold is obtained by choosing an improved substitution such that some of the garbage coefficients in $p_{\mathbf{C}}(x,y,z)$ align with each other resulting in a polynomial of a single variable with fewer coefficients.

A. Complexity Analysis of PolyDot codes

Encoding/decoding complexity: Encoding for one worker requires the evaluation of the polynomials $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$ at a unique value of x. As both the polynomials have m non-zero coefficients which are sub-matrices of $\mathbf{A}$ and $\mathbf{B}$ respectively, the encoder scales the m sub-matrices with N^2/m elements each and adds them up. This requires computational complexity of $\mathcal{O}(m \cdot N^2/m) = \mathcal{O}(N^2)$. Thus, the overall computational complexity of encoding for P worker nodes is $\mathcal{O}(N^2P)$. One could alternatively also use fast polynomial evaluation algorithms [48], [49] to evaluate the two polynomials of respective degrees $st-1$ and $t^2(2s-1)-st$ at P arbitrary points, leading to an encoding complexity of at most $\mathcal{O}\left(N^2P\frac{\log^2(st^2)\log\log(st^2)}{m}\right)$, that can be rewritten as $\mathcal{O}\left(N^2P\frac{\log^2(m^2/s)\log\log(m^2/s)}{m}\right)$ using $st=m$.

Decoding requires interpolating t^2 coefficients of the polynomial $p_{\mathbf{C}}(x)$ of degree $t^2(2s-1)-1$ where each coefficient is of size N^2/t^2. We examine a choice of two decoding algorithms here, and interestingly, again observe a trade-off between MatDot and Polynomial codes in decoding. If we use a decoding technique similar to MatDot codes by considering the problem of deriving the required t^2 linear combinations from the inverse of the $k \times k$ Vandermonde matrix $\mathbf{V}$ and then combining the k evaluated sub-matrices sent by the worker nodes using these t^2 linear combinations, then the overall decoding complexity is $\mathcal{O}(t^2 \cdot \frac{N^2}{t^2}k + k^3) = \mathcal{O}(N^2k + k^3)$ where $k = t^2(2s-1)$. Again, as $k \ll N$, the complexity is dominated by the term N^2k.

Alternatively, the decoder could also choose to solve for all the coefficients of $p_{\mathbf{C}}(x)$ from the evaluations, as a single interpolation problem. There exist fast polynomial interpolation methods that have a complexity of $\mathcal{O}(k\log^2 k)$ theoretically [48] (or more practically $\mathcal{O}(k\log^2 k\log\log k)$ [49]) for a polynomial of degree $k-1$. For this problem $k = t^2(2s-1)$. Therefore, using these fast polynomial interpolation algorithms, the decoding complexity *per coefficient matrix element is* $\mathcal{O}(t^2(2s-1)\log^2 t^2(2s-1)\log\log t^2(2s-1)) = \mathcal{O}(t^2 s\log^2(m^2/s)\log\log m^2/s)$ using $m = st$. As the interpolation is performed N^2/t^2 times for the coefficient matrices of size N^2/t^2, the overall decoding complexity is $\mathcal{O}(N^2 s\log^2(m^2/s)\log\log(m^2/s))$.

Remark V.3. Note that, when we substitute $t=1, s=m$ in the second expression of decoding complexity for PolyDot codes, we get $\mathcal{O}(N^2 m\log^2(m)\log\log(m))$ which differs from the decoding complexity of MatDot and systematic MatDot codes by a factor of $\log^2(m)\log\log(m)$ although it matches with the decoding complexity of Polynomial codes for $t=m, s=1$. This is because for MatDot codes, we only require one coefficient of the polynomial $p_{\mathbf{C}}(x)$ and hence the decoding complexity can be lowered by $\log^2(m)\log\log(m)$ by treating the matrix-inversion and the final coefficient computation separately than solving them together as a single interpolation problem as done in the second case because interpolation also produces all the other coefficients that are not required in MatDot codes. Alternatively, for Polynomial codes, it makes sense to solve a single interpolation problem

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

as all the coefficients of $p_{\mathbf{C}}(x)$ are useful. For a general PolyDot coding scheme, one can choose to invert first and then compute only the required coefficients (first decoding algorithm) or to decode as a single interpolation problem (second decoding algorithm) depending on whether $\mathcal{O}(N^2st^2)$ or $\mathcal{O}(N^2s\log^2(m^2/s)\log\log(m^2/s))$ is lower.

Each worker's computational complexity: Multiplication of matrices of size $N/t \times N/s$ and $N/s \times N/t$ requires $\mathcal{O}(\frac{N^3}{st^2}) = \mathcal{O}(\frac{N^3 s}{m^2})$ computations. For the decoding complexity to be negligible in comparison to the per-node computational complexity, we need either $m^2t^2 = m^4/s^2 = o(N)$ or $m^2\log^2(m^2/s)\log\log(m^2/s) = o(N)$. Similarly, for the encoding complexity to be negligible in comparison to the per-node computational complexity, we need $m^2P/s = o(N)$.

Communication complexity: Master node communicates $\mathcal{O}(N^2/ts) = \mathcal{O}(N^2/m)$ symbols to each worker, hence total outgoing symbols from the master node will be $\mathcal{O}(PN^2/M)$. For decoding, each node sends $\mathcal{O}(N^2/t^2)$ symbols to the fusion node and the recovery threshold is $\mathcal{O}(t^2(2s-1))$. Total number of symbols communicated to the fusion node is $\mathcal{O}((2s-1)N^2)$.

VI. Multiplying more than two matrices

In this section, we present a coding technique for multiplying n matrices (n-*matrix multiplication*), i.e., computing

$$\mathbf{C} = \mathbf{D}^{(1)}\mathbf{D}^{(2)}\cdots\mathbf{D}^{(n)}. \tag{19}$$

We state the problem formally in Section VI-A and then explain why this is different from multiplying two matrices. Then, in Section VI-B, we provide a new code construction called n-*matrix codes* which applies MatDot codes and Polynomial codes in an alternating fashion. With this construction, we show that we can achieve recovery threshold of $\Theta(m^{\lceil n/2\rceil})$ (see Theorem VI.1) followed by a complexity analysis in Section VI-C. After that, we propose a Generalized n-matrix codes in Section VI-D which allows for both horizontal and vertical partitioning of all the matrices being multiplied and again explore the trade-off between recovery threshold (see Theorem VI.2 in Section VI-D) and communication and computation complexity (Section VI-E).

A. Problem Statement

We consider a generalization of the system model of Section II with a master node, P worker nodes, and a fusion node, to multiply more than two matrices. Here the goal is to compute the product $\mathbf{C} = \prod_{i=1}^{n}\mathbf{D}^{(i)}$ of $N \times N$ square matrices, $\mathbf{D}^{(1)}, \cdots, \mathbf{D}^{(n)}$. As we will treat the matrices $\mathbf{D}^{(i)}$ with odd and even indices differently, we will denote the $\mathbf{D}^{(i)}$'s with odd indices as $\mathbf{A}^{(\lceil i/2\rceil)}$ and the $\mathbf{D}^{(i)}$'s with even indices as $\mathbf{B}^{(i/2)}$ for all $i \in \{1, \cdots, n\}$. Using this notation, $\mathbf{C}$ can be written as:

$$\mathbf{C} = \begin{cases} \prod_{i=1}^{\frac{n}{2}}\mathbf{A}^{(i)}\mathbf{B}^{(i)} & \text{if } n \text{ is even,} \\ \left(\prod_{i=1}^{\lfloor\frac{n}{2}\rfloor}\mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}^{(\lceil\frac{n}{2}\rceil)} & \text{if } n \text{ is odd.} \end{cases} \tag{20}$$

In our model, each worker can receive at most nN^2/m symbols from the master node, where each symbol is an element of $\mathbb{F}$. Specifically, for each matrix $\mathbf{D}^{(i)}$, each worker receives N^2/m symbols which are $\mathbb{F}$-linear combinations of the entries of the matrix. Similar to Section II-B, the computational complexities of the operations at master, worker and fusion nodes, in terms of the parameters N, P, m, are required to be strictly less than the computational complexity of a sequential algorithm that computes the product. The goal is to perform this matrix product utilizing faulty or straggling workers with as low recovery threshold as possible. Again, in the following discussion, we will assume that $|\mathbb{F}| > P$.

B. Codes for n-matrix multiplication

Theorem VI.1 (Recovery threshold for n-matrix codes)**.** *For the matrix multiplication problem specified in Section VI-A computed on the system defined in Definition II.1, there exists a code with a recovery threshold of*

$$k(n,m) = \begin{cases} 2m^{n/2} - 1 & \text{if } n \text{ is even,} \\ (m+1)m^{\lfloor\frac{n}{2}\rfloor} - 1 & \text{if } n \text{ is odd.} \end{cases} \tag{21}$$

Proof. See Appendix A. ∎

Discussion on applicability of n-matrix codes:

Before describing the code construction for n-matrix multiplication, we first discuss when n-matrix multiplication codes can be useful despite having a recovery threshold that grows exponentially with n. First, note that as n-matrix multiplication is a chain of $(n-1)$ matrix-matrix multiplications, one may think that we can apply the coding techniques developed in the previous sections to each pairwise matrix multiplication instead of developing a new coding technique for n-matrix multiplication. For example, let us consider computing $\mathbf{C} = \mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)}$. A master node can first encode $\mathbf{A}^{(1)}$ and $\mathbf{B}^{(1)}$ using MatDot codes and distribute encoded matrices to all the worker nodes and the fusion node can decode $\mathbf{E} = \mathbf{A}^{(1)}\mathbf{B}^{(1)}$ from the output of successful worker nodes. Then we again encode $\mathbf{E}$ and $\mathbf{A}^{(2)}$ using MatDot code and distribute encoded matrices to the worker nodes. Finally, the fusion node can reconstruct $\mathbf{C}$ by decoding the outputs of successful worker nodes. As you can see from this example, simply applying MatDot codes on each matrix-matrix multiplication requires two rounds of communication after computing $\mathbf{E} = \mathbf{A}^{(1)}\mathbf{B}^{(1)}$ and $\mathbf{C} = \mathbf{E}\mathbf{A}^{(2)}$. For n-matrix multiplication, it requires $n-1$ rounds of communication. This can be inefficient in the systems when the communication cost increases with number of rounds of communication (e.g., due to large communication setup overheads).

What we propose in this section is a coded n-matrix multiplication strategy which requires only one round of communication. Our main result in Theorem VI.1 shows that n-matrix codes need $\Theta(m^{\lceil n/2\rceil})$ successful nodes to recover the computation result. On the other hand, successively applying MatDot codes requires $\Theta(m)$ nodes to successfully recover the final result, which is is in scaling sense smaller than $\Theta(m^{\lceil n/2\rceil})$ for large n. This suggests that n-matrix codes avoid intermediate communications at the cost of larger recovery threshold. When communication start-up cost is the main source of delay, one should use n-matrix codes, and when number of computation nodes is limited, one should sequentially apply coding strategy

for two-matrix multiplication such as MatDot or PolyDot codes.

Moreover, in many applications such as power-iteration-based methods, one often prefers to compute $\mathbf{A}^n\mathbf{x}^{(0)}$ (where $\mathbf{x}^{(0)} \in \mathbb{R}^n$ is an initial vector) instead of calculating $\mathbf{A}^n$ due to higher computational complexity. Our suggested coded multiple matrix-matrix multiplications can be employed in such applications simply by letting $\mathbf{D}^{(1)} = \mathbf{D}^{(2)} = \ldots = \mathbf{D}^{(n)} = \mathbf{A}$. Further details about this idea can be found in [61]. Therefore, redundancy overhead used in our scheme can be useful in such scenarios for two main reasons: (i) Saving communication cost; and (ii) Providing robustness against stragglers.

We will now begin with simple examples for even and odd n. The first example shows the example for even n, and present a construction for general n.

Example VI.1. [Multiplying 4 matrices ($n = 4, m = 2, k = 7$)]

Here, we give an example of multiplying 4 matrices and show that a recovery threshold of 7 is achievable. For $i \in \{1, 2\}$, matrix $\mathbf{A}^{(i)}$ is split vertically into sub-matrices $\mathbf{A}_0^{(i)}, \mathbf{A}_1^{(i)}$ each of dimension $N \times \frac{N}{2}$ as follows: $\mathbf{A}^{(i)} = \begin{bmatrix}\mathbf{A}_0^{(i)} & \mathbf{A}_1^{(i)}\end{bmatrix}$, while, for $i \in \{1, 2\}$, matrix $\mathbf{B}^{(i)}$ is split horizontally into sub-matrices $\mathbf{B}_0^{(i)}, \mathbf{B}_1^{(i)}$ each of dimension $\frac{N}{2} \times N$ as follows:

$$\mathbf{B}^{(i)} = \begin{bmatrix} \mathbf{B}_0^{(i)} \\ \mathbf{B}_1^{(i)} \end{bmatrix}. \tag{22}$$

Notice that the product $\mathbf{C} = \prod_{i=1}^{2} \mathbf{A}^{(i)}\mathbf{B}^{(i)}$ can now be written as

$$\begin{aligned}\prod_{i=1}^{2} \mathbf{A}^{(i)}\mathbf{B}^{(i)} &= \left(\mathbf{A}^{(1)}\mathbf{B}^{(1)}\right)\left(\mathbf{A}^{(2)}\mathbf{B}^{(2)}\right) \\ &= \left(\mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} + \mathbf{A}_1^{(1)}\mathbf{B}_1^{(1)}\right)\left(\mathbf{A}_0^{(2)}\mathbf{B}_0^{(2)} + \mathbf{A}_1^{(2)}\mathbf{B}_1^{(2)}\right). \end{aligned} \tag{23}$$

Now, we define the encoding polynomials $p_{\mathbf{A}^{(i)}}(x), p_{\mathbf{B}^{(i)}}(x)$, $i \in \{1, 2\}$ as follows:

$$\begin{aligned} p_{\mathbf{A}^{(1)}}(x) &= \mathbf{A}_0^{(1)} + \mathbf{A}_1^{(1)}x, \\ p_{\mathbf{B}^{(1)}}(x) &= \mathbf{B}_0^{(1)}x + \mathbf{B}_1^{(1)}, \\ p_{\mathbf{A}^{(2)}}(x) &= \mathbf{A}_0^{(2)} + \mathbf{A}_1^{(2)}x, \\ p_{\mathbf{B}^{(2)}}(x) &= \mathbf{B}_0^{(2)}x + \mathbf{B}_1^{(2)}. \end{aligned} \tag{24}$$

From (24), we have

$$\begin{aligned} p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x) &= \mathbf{A}_0^{(1)}\mathbf{B}_1^{(1)} + (\mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} + \mathbf{A}_1^{(1)}\mathbf{B}_1^{(1)})x \\ &\quad + \mathbf{A}_1^{(1)}\mathbf{B}_0^{(1)}x^2, \\ p_{\mathbf{A}^{(2)}}(x)p_{\mathbf{B}^{(2)}}(x) &= \mathbf{A}_0^{(2)}\mathbf{B}_1^{(2)} + (\mathbf{A}_0^{(2)}\mathbf{B}_0^{(2)} + \mathbf{A}_1^{(2)}\mathbf{B}_1^{(2)})x \\ &\quad + \mathbf{A}_1^{(2)}\mathbf{B}_0^{(2)}x^2. \end{aligned} \tag{25}$$

From (23) along with (25), we can observe the following:

(i) the coefficient of x in $p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x)$ is $\mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} + \mathbf{A}_1^{(1)}\mathbf{B}_1^{(1)} = \mathbf{A}^{(1)}\mathbf{B}^{(1)}$,

(ii) the coefficient of x^2 in $p_{\mathbf{A}^{(2)}}(x^2)p_{\mathbf{B}^{(2)}}(x^2)$ is the product $\mathbf{A}_0^{(2)}\mathbf{B}_0^{(2)} + \mathbf{A}_1^{(2)}\mathbf{B}_1^{(2)} = \mathbf{A}^{(2)}\mathbf{B}^{(2)}$, and

(iii) the coefficient of x^3 in $p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x)p_{\mathbf{A}^{(2)}}(x^2)p_{\mathbf{B}^{(2)}}(x^2)$ is the product $\prod_{i=1}^{2}\mathbf{A}^{(i)}\mathbf{B}^{(i)}$ (our desired output).

Let $x_1, \cdots, x_P$ be distinct elements of $\mathbb{F}$, the master node sends $p_{\mathbf{A}^{(i)}}(x_r^i)$ and $p_{\mathbf{B}^{(i)}}(x_r^i)$, for all $i \in \{1, 2\}$, to the r-th worker node, $r \in \{1, \cdots, P\}$, and the r-th worker node performs the multiplication $\prod_{i=1}^{2} p_{\mathbf{A}^{(i)}}(x_r^i)p_{\mathbf{B}^{(i)}}(x_r^i)$ and sends the output to the fusion node.

Let worker nodes $1, \cdots, 7$ be the first 7 worker nodes to send their computation outputs to the fusion node, then the fusion node receives the matrices $\prod_{i=1}^{2} p_{\mathbf{A}^{(i)}}(x_r^i)p_{\mathbf{B}^{(i)}}(x_r^i)$ for all $r \in \{1, \cdots, 7\}$. Since these 7 matrices can be seen as 7 evaluations of the matrix polynomial $\prod_{i=1}^{2} p_{\mathbf{A}^{(i)}}(x^i)p_{\mathbf{B}^{(i)}}(x^i)$ of degree 6 at 7 distinct evaluation points $x_1, \cdots, x_7$, the coefficients of the matrix polynomial $\prod_{i=1}^{2} p_{\mathbf{A}^{(i)}}(x^i)p_{\mathbf{B}^{(i)}}(x^i)$ can be obtained using polynomial interpolation. This includes the coefficient of x^3, i.e., $\prod_{i=1}^{2}\mathbf{A}^{(i)}\mathbf{B}^{(i)}$. ■

Now we show an example for odd n.

Example VI.2. [Multiplying 3 matrices ($n = 3, m = 2, k = 5$)]

Here, we give an example of multiplying 3 matrices and show that a recovery threshold of 5 is achievable. In this example, we have three input matrices $\mathbf{A}^{(1)}$, $\mathbf{B}^{(1)}$, and $\mathbf{A}^{(2)}$, each of dimension $N \times N$ and need to compute the product $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)}$. First, the three input matrices are split in the same way as in Example VI.1. The product $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)}$ can now be written as

$$\mathbf{C} = \mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)} = \begin{bmatrix}\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}_0^{(2)} & \mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}_1^{(2)}\end{bmatrix}, \tag{26}$$

where $\mathbf{A}^{(1)}\mathbf{B}^{(1)} = \mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} + \mathbf{A}_1^{(1)}\mathbf{B}_1^{(1)}$.

Now, we define the encoding polynomials $p_{\mathbf{A}^{(1)}}(x), p_{\mathbf{B}^{(1)}}(x), p_{\mathbf{A}^{(2)}}(x)$ as follows:

$$\begin{aligned} p_{\mathbf{A}^{(1)}}(x) &= \mathbf{A}_0^{(1)} + \mathbf{A}_1^{(1)}x, \\ p_{\mathbf{B}^{(1)}}(x) &= \mathbf{B}_0^{(1)}x + \mathbf{B}_1^{(1)}, \\ p_{\mathbf{A}^{(2)}}(x) &= \mathbf{A}_0^{(2)} + \mathbf{A}_1^{(2)}x. \end{aligned} \tag{27}$$

From (27), we have

$$\begin{aligned} p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x)p_{\mathbf{A}^{(2)}}(x^2) &= \mathbf{A}_0^{(1)}\mathbf{B}_1^{(1)}\mathbf{A}_0^{(2)} \\ &\quad + (\mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} + \mathbf{A}_1^{(1)}\mathbf{B}_1^{(1)})\mathbf{A}_0^{(2)}x \\ &\quad + (\mathbf{A}_1^{(1)}\mathbf{B}_0^{(1)}\mathbf{A}_0^{(2)} + \mathbf{A}_0^{(1)}\mathbf{B}_1^{(1)}\mathbf{A}_1^{(2)})x^2 \\ &\quad + (\mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} + \mathbf{A}_1^{(1)}\mathbf{B}_1^{(1)})\mathbf{A}_1^{(2)}x^3 + \mathbf{A}_1^{(1)}\mathbf{B}_0^{(1)}\mathbf{A}_1^{(2)}x^4. \end{aligned} \tag{28}$$

From (28), we can observe the following:

(i) the coefficient of x in $p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x)p_{\mathbf{A}^{(2)}}(x^2)$ is the product $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}_0^{(2)}$, and

(ii) the coefficient of x^3 in $p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x)p_{\mathbf{A}^{(2)}}(x^2)$ is the product $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}_1^{(2)}$.

From (26), these two coefficients suffice to recover $\mathbf{C}$. Let $x_1, \cdots, x_P$ be distinct elements of $\mathbb{F}$, the master node sends $p_{\mathbf{A}^{(i)}}(x_r^i)$, for all $i \in \{1, 2\}$, and $p_{\mathbf{B}_1}(x_r)$ to the r-th worker node, $r \in \{1, \cdots, P\}$, where the r-th worker node performs

the multiplication $p_{\mathbf{A}^{(1)}}(x_r)p_{\mathbf{B}^{(1)}}(x_r)p_{\mathbf{A}^{(2)}}(x_r^2)$ and sends the output to the fusion node.

Let worker nodes $1,\cdots,5$ be the first 5 worker nodes to send their computation outputs to the fusion node, then the fusion node receives the matrices $p_{\mathbf{A}^{(1)}}(x_r)p_{\mathbf{B}^{(1)}}(x_r)p_{\mathbf{A}^{(2)}}(x_r^2)$ for all $r\in\{1,\cdots,5\}$. Since these 5 matrices can be seen as 5 evaluations of the polynomial $p_{\mathbf{A}^{(1)}}(x)p_{\mathbf{B}^{(1)}}(x)p_{\mathbf{A}^{(2)}}(x^2)$ of degree 4 at five distinct evaluation points $x_1,\cdots,x_5$, the coefficients of the matrix polynomial $p_{\mathbf{A}^{(1)}}(x_r)p_{\mathbf{B}^{(1)}}(x_r)p_{\mathbf{A}^{(2)}}(x_r^2)$ can be obtained using polynomial interpolation. This includes the coefficients of x and x^3, i.e., $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}_0^{(2)}$ and $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}_1^{(2)}$. ■

Next, we present a code construction for n-matrix multiplication for general n and m.

Construction VI.1. *[n-matrix codes]*

***Splitting of input matrices**: for every $i\in\{1,\cdots,\lceil\frac{n}{2}\rceil\}$ and $j\in\{1,\cdots,\lfloor\frac{n}{2}\rfloor\}$, $\mathbf{A}_i$ and $\mathbf{B}_j$ are split as follows*

$$\mathbf{A}^{(i)}=\begin{bmatrix}\mathbf{A}_1^{(i)} & \mathbf{A}_2^{(i)} & \dots & \mathbf{A}_m^{(i)}\end{bmatrix},\quad \mathbf{B}^{(j)}=\begin{bmatrix}\mathbf{B}_1^{(j)}\\ \mathbf{B}_2^{(j)}\\ \vdots\\ \mathbf{B}_m^{(j)}\end{bmatrix},\tag{29}$$

where, for $k\in\{1,\dots,m\}$, $\mathbf{A}_k^{(i)},\mathbf{B}_k^{(j)}$ are $N\times N/m$ and $N/m\times N$ dimensional matrices, respectively.

***Master node (encoding)**: Let $x_1,x_2,\dots,x_{P-1}$ be arbitrary distinct elements of $\mathbb{F}$. For $i\in\{1,\cdots,\lceil\frac{n}{2}\rceil\}$, define $p_{\mathbf{A}^{(i)}}(x)=\sum_{j=1}^m\mathbf{A}_j^{(i)}x^{j-1}$, and, for $i\in\{1,\cdots,\lfloor\frac{n}{2}\rfloor\}$, define $p_{\mathbf{B}^{(i)}}(x)=\sum_{j=1}^m\mathbf{B}_j^{(i)}x^{m-j}$. For $r\in\{1,2,\dots,P\}$, the master node sends to the r-th worker the evaluations, $p_{\mathbf{A}^{(i)}}(x_r^{m^{i-1}})$ and $p_{\mathbf{B}^{(j)}}(x_r^{m^{j-1}})$, for all $i\in\{1,\cdots,\lceil\frac{n}{2}\rceil\}$ and $j\in\{1,\cdots,\lfloor\frac{n}{2}\rfloor\}$.*

***Worker nodes**: For $i\in\{1,\cdots,\lceil\frac{n}{2}\rceil\}$, define*

$$p_{\mathbf{C}^{(i)}}(x)=\begin{cases}p_{\mathbf{A}^{(i)}}(x)p_{\mathbf{B}^{(i)}}(x) & \text{if } i\in\{1,\cdots,\lfloor\frac{n}{2}\rfloor\},\\ p_{\mathbf{A}^{(i)}}(x) & \text{if } n \text{ is odd and } i=\lceil\frac{n}{2}\rceil.\end{cases}\tag{30}$$

For $r\in\{1,2,\dots,P\}$, the r-th worker node computes the matrix product $\Pi_{i=1}^{\lceil\frac{n}{2}\rceil}p_{\mathbf{C}^{(i)}}(x_r^{m^{i-1}})$ and sends it to the fusion node on successful completion.

***Fusion node (decoding)**: If n is even, the fusion node uses outputs of any $2m^{\frac{n}{2}}-1$ successful workers to compute the coefficient of $x^{m^{n/2}-1}$ in the matrix polynomial $\Pi_{i=1}^{\frac{n}{2}}p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$, and if n is odd, the fusion node uses outputs of any $m^{\lfloor\frac{n}{2}\rfloor}(m+1)-1$ successful workers to compute the coefficients of $x^{jm^{\lfloor\frac{n}{2}\rfloor}-1}$, for all $j\in\{1,\cdots,m\}$, in the matrix polynomial $\Pi_{i=1}^{\lceil\frac{n}{2}\rceil}p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ (the feasibility of this step will be shown later in the proof of Theorem VI.1).*

If the number of successful workers is smaller than $2m^{\frac{n}{2}}-1$ for even n or smaller than $m^{\lfloor\frac{n}{2}\rfloor}(m+1)-1$ for odd n, the fusion node declares a failure.

Remark VI.1. The coefficient of $x^{m^i-m^{i-1}}$ in $p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$, for any $i\in\{1,\cdots,\lfloor\frac{n}{2}\rfloor\}$, is $\sum_{j=1}^m\mathbf{A}_j^{(i)}\mathbf{B}_j^{(i)}=\mathbf{A}^{(i)}\mathbf{B}^{(i)}$.

Remark VI.2. A reader might wonder why there is a difference between odd-valued and even-valued n, and if one can be reduced to the other by introducing an identity matrix of dimensions $N\times N$ in the n-matrix multiplication problem. In this work, we have an assumption that the matrices being multiplied are not known in advance and may even be chosen by an adversary. If it is known in advance that one of the matrices is an identity matrix or even a matrix with a special structure, e.g., a Toeplitz matrix (essentially convolution), then alternative coding techniques might be applicable altogether, which we hope to explore as a future work. Here, we assume that none of the matrices are known to us, and we aim to find a general scheme. When $n=2$, the n-matrix codes is exactly MatDot codes. When $n=3$, (e.g., multiplying $\mathbf{ABC}$), it is Polynomial codes applied to $\mathbf{AB}$ and $\mathbf{C}$, followed by MatDot codes. It reduces to simply computing $\mathbf{AB}$ when we know that the third matrix $\mathbf{C}$ is identity, but without the hindsight, we still have to encode the identity matrix, resulting in a bigger recovery threshold than multiplying two matrices.

C. Complexity Analyses of n-matrix codes (Construction VI.1)

Encoding/decoding complexity: Decoding requires interpolating a $2m^{n/2}-2$ degree polynomial if n is even or a $m^{\lfloor\frac{n}{2}\rfloor}(m+1)-2$ degree polynomial if n is odd for each element in the matrix. Using polynomial interpolation algorithms of complexity $\mathcal{O}(k\log^2 k)$ [48], or $\mathcal{O}(k\log^2 k\log\log k)$ [49], where $k=k(n,m)$ as defined in (21), complexity per matrix element is $\mathcal{O}(m^{\lceil\frac{n}{2}\rceil}\log^2 m^{\lceil\frac{n}{2}\rceil})\log\log m^{\lceil\frac{n}{2}\rceil})$. Thus, for N^2 elements, the decoding complexity is $\mathcal{O}(N^2m^{\lceil\frac{n}{2}\rceil}\log^2 m^{\lceil\frac{n}{2}\rceil}\log\log m^{\lceil\frac{n}{2}\rceil})$.

Encoding for each worker requires performing n additions, each adding m scaled matrices of size N^2/m, for an overall encoding complexity for *each worker* of $\mathcal{O}(mnN^2/m)=\mathcal{O}(nN^2)$. Thus, the overall computational complexity of encoding for P workers is $\mathcal{O}(nN^2P)$.

Each worker's computational cost: Each worker multiplies n matrices of dimensions $N\times N/m$ and $N/m\times N$. For any worker r with $r\in\{1,\cdots,P\}$, the multiplication can be performed as follows:

Case 1**:** n is even

In this case, worker r wishes to compute the product:

$$p_{\mathbf{A}^{(1)}}(x_r)p_{\mathbf{B}^{(1)}}(x_r)p_{\mathbf{A}^{(2)}}(x_r^m)p_{\mathbf{B}^{(2)}}(x_r^m)\cdots$$
$$p_{\mathbf{A}^{(n/2)}}(x_r^{m^{n/2-1}})p_{\mathbf{B}^{(n/2)}}(x_r^{m^{n/2-1}}).$$

Worker r does this multiplication in the following order:

1. Compute $p_{\mathbf{B}^{(i)}}(x_r^{m^{i-1}})p_{\mathbf{A}^{(i+1)}}(x_r^{m^i})$ for all $i\in\{1,\cdots,n/2-1\}$ with a total complexity of $\mathcal{O}(nN^3/m^2)$.
2. Compute the product of the output matrices of the previous step with a total complexity of $\mathcal{O}(nN^3/m^3)$. Call this product matrix $\mathbf{D}$. Notice that $\mathbf{D}$ has a dimension of $N/m\times N/m$.
3. Compute $p_{\mathbf{A}^{(1)}}(x_r)\mathbf{D}$ with complexity $\mathcal{O}(N^3/m^2)$. Call this product matrix $\mathbf{E}$. Notice that $\mathbf{E}$ has a dimension of $N\times N/m$.
4. Compute $\mathbf{E}\,p_{\mathbf{B}^{(n/2)}}(x_r^{m^{n/2-1}})$ with complexity $\mathcal{O}(N^3/m)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIT.2019.2929328, IEEE Transactions on Information Theory

Hence, the overall computational complexity per worker for even n is $\mathcal{O}(\max(nN^3/m^2, nN^3/m^3, N^3/m^2, N^3/m)) = \mathcal{O}(\max(nN^3/m^2, N^3/m))$.

Case 2: n is odd

In this case, worker r wishes to compute the product:

$$p_{\mathbf{A}^{(1)}}(x_r)p_{\mathbf{B}^{(1)}}(x_r)\cdots p_{\mathbf{A}^{((n-1)/2)}}(x_r^{m^{(n-3)/2}})$$
$$p_{\mathbf{B}^{((n-1)/2)}}(x_r^{m^{(n-3)/2}})p_{\mathbf{A}^{((n+1)/2)}}(x_r^{m^{(n-1)/2}}).$$

Worker r does this multiplication in the following order:

1. Compute $p_{\mathbf{B}^{(i)}}(x_r^{m^{i-1}})p_{\mathbf{A}^{(i+1)}}(x_r^{m^i})$ for all $i \in \{1, \cdots, (n-1)/2\}$ with a total complexity of $\mathcal{O}(nN^3/m^2)$.
2. Compute the product of the output matrices of the previous step with a total complexity of $\mathcal{O}(nN^3/m^3)$. Call this product matrix $\mathbf{D}$. Notice that $\mathbf{D}$ has a dimension of $N/m \times N/m$.
3. Compute $p_{\mathbf{A}^{(1)}}(x_r)\mathbf{D}$ with complexity $\mathcal{O}(N^3/m^2)$.

Hence, the overall computational complexity per worker for odd n is $\mathcal{O}(\max(nN^3/m^2, nN^3/m^3, N^3/m^2)) = \mathcal{O}(nN^3/m^2)$.

In conclusion, the computational complexity per worker is $\mathcal{O}(\max(nN^3/m^2, N^3/m))$ if n is even, and $\mathcal{O}(nN^3/m^2)$ if n is odd[8].

Communication cost: The master node communicates total of $\mathcal{O}(nPN^2/m)$ symbols to the worker nodes, and the fusion node receives $\mathcal{O}(m^{\lfloor \frac{n}{2} \rfloor}N^2)$ symbols from the successful worker nodes.

D. Codes for Generalized n-matrix multiplication

Here, we give another code construction for n-matrix multiplication which is a generalization of the code construction given in the previous section. The new construction allows us to split input matrices more flexibly and trades off communication and computation (similar to PolyDot codes in Section V for two matrices). The results presented here are an improvement over [1], and are built on techniques from [45], [46].

Theorem VI.2 (Recovery threshold for Generalized n-matrix codes)**.** *For the matrix multiplication problem specified in Section VI-A and computed on the system defined in Definition II.1, there exists a code with a recovery threshold of*

$$k(n,s,t) = \begin{cases} s^{\frac{n}{2}}t^{\frac{n}{2}+1} + s^{\frac{n}{2}}t^{\frac{n}{2}-1} - 1 & \text{if } n \text{ is even,} \\ s^{\frac{n+1}{2}}t^{\frac{n+1}{2}} + s^{\frac{n-1}{2}}t^{\frac{n-1}{2}} - 1 & \text{if } n \text{ is odd} \end{cases} \tag{31}$$

for any integers s, t that satisfy $m = st$.

Proof. See Appendix B. ∎

Remark VI.3. If we substitute $st = m$ in (31), we get:

$$k(n,s,t) = \begin{cases} m^{\frac{n}{2}}(t+1) - t & \text{if } n \text{ is even,} \\ m^{\frac{n-1}{2}}(m+t) - t & \text{if } n \text{ is odd} \end{cases} \tag{32}$$

[8]The expressions for even n and odd n are different due to the last step in the even n case where we compute the matrix multiplication of dimension $N \times N/m$ and $N/m \times N$, which has computational complexity of $\mathcal{O}(N^3/m)$

By plugging in $s = m, t = 1$, we can see that $k(n,s,t) = 2m^{n/2} - 1$ for n even, and $k(n,s,t) = m^{\frac{n+1}{2}} + m^{\frac{n-1}{2}} - 1$ for n odd. This matches the recovery threshold given in (21). Also, note that if we consider the substitutions given here for the particular case of two-matrix multiplication, i.e., $n = 2$, the recovery thresholds that we obtain are actually better than the recovery threshold of PolyDot codes as proposed in the initial version of this work in [1], and matches the recovery threshold for two-matrix multiplication in subsequent works [45], [46].

We now give a construction of Generalized n-matrix codes.

Construction VI.2 (Generalized n-matrix multiplication code)**.**

Splitting of input matrices*: We split $\mathbf{A}_i$'s and $\mathbf{B}_i$'s as follows:*

$$\mathbf{A}^{(i)} = \begin{bmatrix} \mathbf{A}_{0,0}^{(i)} & \cdots & \mathbf{A}_{0,s-1}^{(i)} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{t-1,0}^{(i)} & \cdots & \mathbf{A}_{t-1,s-1}^{(i)} \end{bmatrix},$$
$$\mathbf{B}^{(i)} = \begin{bmatrix} \mathbf{B}_{0,0}^{(i)} & \cdots & \mathbf{B}_{0,t-1}^{(i)} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{s-1,0}^{(i)} & \cdots & \mathbf{B}_{s-1,t-1}^{(i)} \end{bmatrix}, \tag{33}$$

where $\mathbf{A}_{j,k}^{(i)}$'s have dimension $N/t \times N/s$ and $\mathbf{B}_{j,k}^{(i)}$'s have dimension $N/s \times N/t$.

Master node (encoding)*: Define the encoding polynomials as*

$$p_{\mathbf{A}^{(1)}}(z_1, z_2) = \sum_{i=0}^{t-1}\sum_{j=0}^{s-1} \mathbf{A}_{i,j}^{(1)} z_1^i z_2^j,$$
$$p_{\mathbf{B}^{(1)}}(z_2, z_3) = \sum_{i=0}^{s-1}\sum_{j=0}^{t-1} \mathbf{B}_{i,j}^{(1)} z_2^{s-1-i} z_3^j,$$
$$\vdots,$$
$$p_{\mathbf{B}^{(n/2)}}(z_n, z_{n+1}) = \sum_{i=0}^{s-1}\sum_{j=0}^{t-1} \mathbf{B}_{i,j}^{(n/2)} z_n^{s-1-i} z_{n+1}^j.$$

for n even, and

$$p_{\mathbf{A}^{(1)}}(z_1, z_2) = \sum_{i=0}^{t-1}\sum_{j=0}^{s-1} \mathbf{A}_{i,j}^{(1)} z_1^i z_2^j,$$
$$\vdots,$$
$$p_{\mathbf{B}^{((n-1)/2)}}(z_{n-1}, z_n) = \sum_{i=0}^{s-1}\sum_{j=0}^{t-1} \mathbf{B}_{i,j}^{((n-1)/2)} z_{n-1}^{s-1-i} z_n^j,$$
$$p_{\mathbf{A}^{((n+1)/2)}}(z_n, z_{n+1}) = \sum_{i=0}^{t-1}\sum_{j=0}^{s-1} \mathbf{A}_{i,j}^{((n-1)/2)} z_n^{t-1-i} z_{n+1}^j,$$

for n odd.

The master node sends to the r-th worker evaluations of $p_{\mathbf{A}^{(i)}}$'s, and $p_{\mathbf{B}^{(i)}}$'s at

$$z_1 = x^{s^{n/2}t^{n/2-1}}, z_2 = x, z_3 = x^s, \cdots,$$
$$z_n = x^{s^{n/2-1}t^{n/2-1}}, z_{n+1} = x^{s^{n/2}t^{n/2}} \quad \text{for } n \text{ even}, \tag{34}$$
$$z_1 = x^{s^{(n-1)/2}t^{(n-1)/2}}, z_2 = x, x_3 = x^s, \cdots,$$
$$z_n = x^{s^{(n-1)/2}t^{(n-3)/2}}, z_{n+1} = x^{s^{(n-1)/2}t^{(n+1)/2}} \quad \text{for } n \text{ odd}. \tag{35}$$

where x_r's are all distinct for $r \in \{1, 2, \ldots, P\}$.

***Fusion node (decoding)**: The fusion node uses outputs of any $k(n, s, t)$ successful workers (given in (31)) to compute the coefficients of $p_{\mathbf{C}}(z)$. If the number of successful workers is smaller than $k(n, s, t)$, the fusion node declares a failure.*

Remark VI.4. The two strategies for n-matrix multiplication proposed in this work can be understood better in our general PolyDot framework (see Table I). Essentially, they differ in the substitutions for the variables $z_1, \cdots, z_{n+1}$ to convert the polynomial in n variables into a polynomial in a single variable for the ease of interpolation. The main intuition behind the substitutions of (34) and (35) is that for z_1 and z_{n+1}, their powers grow from 0 to $t-1$ (or $s-1$), while all the other terms have powers growing from 0 to $2s-2$ (or $2t-2$). Hence, to minimize the maximum degree of the product polynomial, it is best to assign high powers of x to z_1 and z_{n+1}. An alternate substitution could also be to start with $z_1 = x$ and then continue substituting $z_2 = x^t$, $z_3 = x^{st}$, $z_4 = x^{st^2}$, ..., $z_{n+1} = s^{\lfloor \frac{n}{2} \rfloor} t^{\lceil \frac{n}{2} \rceil}$. The recovery threshold resulting due to this substitution is given by:

$$k(n, s, t) = \begin{cases} s^{\frac{n}{2}} t^{\frac{n}{2}+1} + s^{\frac{n}{2}} t^{\frac{n}{2}} - t & \text{if } n \text{ is even}, \\ s^{\frac{n+1}{2}} t^{\frac{n+1}{2}} + s^{\frac{n-1}{2}} t^{\frac{n+1}{2}} - t & \text{if } n \text{ is odd} \end{cases} \tag{36}$$

for any integers s, t that satisfy $m = st$. This is slightly higher than the recovery threshold obtained in Theorem VI.2. Thus, for $n > 2$, we can improve the recovery threshold by delving deeper into the order of the substitution.

E. Complexity Analysis of Generalized n-matrix codes

Encoding/decoding complexity: Encoding communication cost is $\mathcal{O}(nN^2P)$ as in Section VI-C. Decoding complexity is $\mathcal{O}(\frac{N^2}{t^2} k(n, s, t) \log^2 k(n, s, t) \log\log k(n, s, t))$ (even case) or $\mathcal{O}(\frac{N^2}{ts} k(n, s, t) \log^2 k(n, s, t) \log\log k(n, s, t))$ (odd case).

Communication Complexity: The master node sends out $\mathcal{O}(nPN^2/m)$ encoded symbols in the beginning. After the completion of computation, each node has to send $\mathcal{O}(N^2/t^2)$ symbols to the fusion node. Hence, total number of symbols the fusion node receives is $k(n, s, t) \cdot N^2/t^2$. Let us first consider the case when n is even. By substituting (31), we obtain $k(n, s, t)N^2/t^2 = \mathcal{O}(m^{n/2}/t)$. This is the same trade-off we observed using PolyDot codes for single matrix-matrix multiplication. For a fixed m, recovery threshold $k(n, s, t)$ grows linearly with t while communication cost is inversely related to t (See Fig.6). When n is odd, we do not see such trade-off. Recovery threshold is always $m^{(n-1)/2}(m+t)-t = \mathcal{O}(m^{(n+1)/2})$ regardless of the choice of t. Communication cost on the other hand is $k(n, s, t)N^2/t^2 = \mathcal{O}(m^{(n+1)/2}/t^2 + m^{(n-1)/2}/t)$ which decreases with growing t. For instance, if $t = 1$, communication cost is $\mathcal{O}(m^{(n+1)/2})$, and when $t = m$, communication cost is $\mathcal{O}(m^{(n-3)/2})$. This suggests that when n is odd, it is always better to choose $t = m$ as m grows to infinity.

Each worker's computation cost: Using the similar technique shown in Section VI-C, we can show that each worker's computation complexity is at most $\mathcal{O}(\max(nN^3/m^{1.5}, N^3/m))$ for any choice of s, t. If we compare the computation complexity for encoding/decoding and the computation complexity at each worker node, we can see that as long as $N > \mathcal{O}(m^{n/2-1.5} \log m)$, encoding/decoding computation overhead is amortized.

Remark VI.5. Our result given here splits $\mathbf{A}^{(i)}$'s into $s \times t$ grid of blocks and $\mathbf{B}^{(i)}$'s into $t \times s$ grid of blocks. However, it is not necessary that all matrices have to be split in the same fashion. For instance, $\mathbf{A}^{(1)}$ can be divided into $t_1 \times s_1$ grid and $\mathbf{B}^{(1)}$ can be divided into $s_1 \times t_2$ grid, and so on. In this more general setting $\mathbf{A}^{(i)}$'s are split into $t_i \times s_i$ grid and $\mathbf{B}^{(i)}$'s are split into $s_i \times t_{i+1}$ grid. Let us denote $\mathbf{s} = [s_1, \cdots, s_{n/2}], \mathbf{t} = [t_1, \cdots, t_{n/2+1}]$. Then Theorem VI.2 can be rewritten as follows.

$$k(n, \mathbf{s}, \mathbf{t}) = \begin{cases} (t_{n/2+1} + 1/t_1) \prod_{i=1}^{n/2} s_i t_i - 1 & \text{if } n \text{ even}, \\ (t_1 s_{(n+1)/2} + 1) \prod_{i=1}^{(n-1)/2} s_i t_i - 1 & \text{if } n \text{ odd}. \end{cases} \tag{37}$$

Remark VI.6. In this work we assumed that all matrices have size $N \times N$ for simplicity. However, this assumption is not necessary in the results presented here. When we have matrices with different dimensions to multiply, splitting each matrix in a different way would be more beneficial. For example, when we multiply matrices $\mathbf{A}, \mathbf{B}$ with dimensions $N \times N$ and $N \times 2$, we can divide $\mathbf{A}$ into $t \times s$ grid and divide $\mathbf{B}$ into $s \times 1$ grid.

VII. Discussion and Future Work

We provide a novel MatDot code construction for coded matrix multiplication with a recovery threshold of $2m-1$. Currently, this is the best known recovery threshold for storage-constrained coded matrix multiplication. We also present a systematic MatDot construction achieving the same recovery threshold. Note that a recent converse of Yu et al. [45] shows that the recovery threshold of MatDot codes is optimal for the chosen partitioning of the matrices under the given storage constraints when using linear codes. In this paper, we also provide full proofs of results that appeared in [1], including PolyDot constructions which allow a trade-off between communication cost and recovery threshold. Finally, we provide code constructions for multiplying more than two matrices.

We conclude with a discussion that uses an important open problem, namely coded tensor products, to demonstrate how focusing exclusively on recovery thresholds, and ignoring encoding/decoding costs in coded computing problems, can provide impractical solutions.

TABLE I
COMPARISON OF DIFFERENT STRATEGIES FOR MULTIPLYING n MATRICES USING DIFFERENT SUBSTITUTIONS IN THE GENERAL POLYDOT FRAMEWORK WHEN n IS EVEN.

	n-matrix codes	Generalized n-matrix codes	Alternate Substitution
Substitution	$z_1 = z_2 = x, z_3 = z_4 = x^m, \cdots, z_{n-1} = x_n = x^{m^{n/2-1}}, z_{n+1} = x^{m^{n/2}}$	$z_1 = x^{s^{n/2}t^{n/2-1}}, z_2 = x, x_3 = x^s, \cdots, z_n = x^{s^{n/2-1}t^{n/2-1}}, z_{n+1} = x^{s^{n/2}t^{n/2}}$	$z_1 = x, z_2 = x^t, z_3 = x^{st}, \cdots, z_{n+1} = x^{s^{n/2}t^{n/2}}$
Recovery Threshold	$2m^{n/2} - 1$	$s^{\frac{n}{2}}t^{\frac{n}{2}+1} + s^{\frac{n}{2}}t^{\frac{n}{2}-1} - 1$	$s^{\frac{n}{2}}t^{\frac{n}{2}+1} + s^{\frac{n}{2}}t^{\frac{n}{2}} - t$

A. When is coded computing useful? An example of coded tensor products

Consider the problem of computing the tensor product of two $N \times N$ square matrices $\mathbf{A}$ and $\mathbf{B}$, i.e., $\mathbf{A} \otimes \mathbf{B}$, using P workers in the system defined in Section II. As usual, our goal is to implement this in a parallelized fashion with a low recovery threshold. For this problem, we show (below) that an application of Polynomial codes [6] yields a recovery threshold of m^2. However, we also show that this makes the decoding complexity at the fusion node comparable to (or sometimes even larger than) the overall per-worker computational complexity. This can be undesirable when coded computing is performed to address straggling because now the fusion node itself becomes the bottleneck. This leads to two interesting questions for future work:

- Is there an application where the high decoding cost at the fusion node can be justified?
- Are there alternative techniques of coding tensor products with reduced decoding overhead?

To be concrete, the Polynomial coded tensor-product strategy is as follows. We split the two matrices $\mathbf{A}$ and $\mathbf{B}$ as follows:

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_0 & \mathbf{A}_1 & \dots & \mathbf{A}_{m-1}\end{bmatrix}, \mathbf{B} = \begin{bmatrix}\mathbf{B}_0 & \mathbf{B}_1 & \dots & \mathbf{B}_{m-1}\end{bmatrix}.$$

Note that,

$$\begin{aligned}\mathbf{A} \otimes \mathbf{B} &= [\mathbf{A}_0 \otimes \mathbf{B} \quad \dots \quad \mathbf{A}_{m-1} \otimes \mathbf{B}] \\ &= [\mathbf{A}_0 \otimes \mathbf{B}_0 \;\; \mathbf{A}_0 \otimes \mathbf{B}_1 \;\; \cdots \mathbf{A}_0 \otimes \mathbf{B}_{m-1} \;\; \cdots \\ &\quad \mathbf{A}_{m-1} \otimes \mathbf{B}_0 \;\; \cdots \;\; \mathbf{A}_{m-1} \otimes \mathbf{B}_{m-1}]\end{aligned}$$

and thus it suffices to compute all terms of the form $\mathbf{A}_i \otimes \mathbf{B}_j$ for $i, j = 0, \cdots, m-1$ to obtain $\mathbf{A} \otimes \mathbf{B}$.

Let us define $p_{\mathbf{A}}(x) = \sum_{i=0}^{m-1} \mathbf{A}_i x^i$ and $p_{\mathbf{B}}(x) = \sum_{j=0}^{m-1} \mathbf{B}_j x^{mj}$ respectively. Let us also choose distinct scalars $x_1, x_2, \dots, x_P$ for each worker. Each worker receives the evaluation of $p_{\mathbf{A}}(x)$ and $p_{\mathbf{B}}(x)$ at distinct scalar values, i.e., at $x = x_1, x_2, \dots, x_P$ respectively. The worker then computes the tensor product $p_{\mathbf{A}}(x) \otimes p_{\mathbf{B}}(x)$ that we will denote as $p_{\mathbf{A}\otimes\mathbf{B}}(x)$ at a distinct scalar value of x. Thus, worker r computes $p_{\mathbf{A}}(x_r) \otimes p_{\mathbf{B}}(x_r)$ for $r = 1, 2, \dots, P$.

Observe that $p_{\mathbf{A}\otimes\mathbf{B}}(x)$ is a polynomial of degree $m^2 - 1$:

$$\begin{aligned}p_{\mathbf{A}\otimes\mathbf{B}}(x) = p_{\mathbf{A}}(x) \otimes p_{\mathbf{B}}(x) &= \Big(\sum_{i=0}^{m-1} \mathbf{A}_i x^i\Big) \otimes \Big(\sum_{j=0}^{m-1} \mathbf{B}_j x^{mj}\Big) \\ &= \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} (\mathbf{A}_i \otimes \mathbf{B}_j) x^{i+mj}.\end{aligned}$$

The coefficient of x^{i+mj} in $p_{\mathbf{A}\otimes\mathbf{B}}(x)$ is in fact $\mathbf{A}_i \otimes \mathbf{B}_j$, for $0 \le i, j \le m-1$. Thus, if the fusion node is able to interpolate all the coefficients of the polynomial $p_{\mathbf{A}\otimes\mathbf{B}}(x)$, it can successfully recover all the matrices in the set $\{\mathbf{A}_i \otimes \mathbf{B}_j, i, j = 0, 1, \dots, m-1\}$, and therefore $\mathbf{A} \otimes \mathbf{B}$. Because the polynomial is of degree $m^2 - 1$, the fusion node needs m^2 evaluations of the polynomial at distinct values. Worker node r produces an evaluation of the polynomial $p_{\mathbf{A}\otimes\mathbf{B}}(x)$ at $x = x_r$. The fusion node is thus required to wait for any m^2 successful worker nodes, and then it can interpolate $p_{\mathbf{A}\otimes\mathbf{B}}(x)$.

The computational complexity of the fusion node is $\Theta\left(\frac{N^4}{m^2} m^2 \log^2(m^2)\right) = \Theta\left(N^4 \log^2(m)\right)$, which is comparable to the complexity of the entire tensor product, i.e., $\Theta(N^4)$. From the viewpoint of applications, it is typically necessary to have computational complexity at the fusion node to be smaller than the computational complexity at each worker node.

To determine whether coded computing adds significant overhead, we can conceptually classify the computations, that are known to allow for coding in existing literature, into three different categories as follows:

- Both encoding and decoding complexity is negligible compared to the per-node computational complexity: examples include convolutions [42] and matrix multiplications in the regime where the number of nodes P is much smaller than the dimensions of the vectors being convolved or the matrices being multiplied. For computations in this category, both encoding and decoding can be done online because they both add negligible overhead.
- Only decoding complexity is negligible compared to the per-node computational complexity: examples include coded matrix-vector products in the regime where the number of nodes P is much smaller than the dimensions of the matrix. For such computations, the encoding cost can be amortized in applications where the encoding is performed only once, e.g., the same matrix is multiplied with different vectors across multiple iterations, even though decoding can be performed online with negligible overhead. An example of such an application is (non-adaptive) inference in machine learning, where the model is fixed and its encoding cost can be amortized over multiple instances of inference [10].
- The decoding complexity is comparable to the per-node computational complexity: examples may include tensor products discussed above, where decoding online would add a significant overhead to the computation. For tensor products in particular, the encoding can be done online as the encoding complexity is smaller than the per-node

computational complexity. However, finding an application where the high decoding cost can be justified is an interesting question.

B. Fully-Decentralized Implementations

It will also be useful to obtain fully decentralized realizations of coded computing techniques with no centralized master node. This often avoids a "single source of failure," particularly if the encoder or decoder are themselves prone to straggling or errors. We refer interested readers to [13], [32], [46], [62]–[64] for works on completely decentralized implementations.

Appendix A
Proof of Theorem VI.1

We will first prove Lemmas A.1 and A.2 which provide properties of coefficients of products of polynomials. Using Lemmas A.1 and A.2, we show Claims A.1 and A.2 which demonstrate that the product $\mathbf{C}$ is contained in a set of coefficients of the matrix polynomial $\Pi_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}_i}(x^{m^{i-1}})$, where $p_{\mathbf{C}_i}(x)$ is as defined in (30), for $i \in \{1, \cdots, \lceil n/2 \rceil\}$. Finally, we provide a proof of Theorem VI.1 using Claims A.1 and A.2.

Lemma A.1. *If $p(x) = \sum_{j=0}^{2d^{i-1}-2} p_j x^j$ is a polynomial with degree $2d^{i-1}-2$ for some $i \geq 2$, and $q(x) = \sum_{j=0}^{2d-2} q_j x^j$ is any other polynomial with degree $2d-2$, then $p_{d^{i-1}-1} q_{d-1}$ is the coefficient of x^{d^i-1} in $p(x)q(x^{d^{i-1}})$.*

Proof. We first expand out $p(x)$ and $q(x)$ as following:

$$p(x) = \underbrace{\sum_{j=0}^{d^{i-1}-2} p_j x^j}_{\tilde{p}_1(x)} + p_{d^{i-1}-1} x^{d^{i-1}-1} + \underbrace{\sum_{j=d^{i-1}}^{2d^{i-1}-2} p_j x^j}_{\tilde{p}_2(x)} \quad (38)$$

$$q(x) = \underbrace{\sum_{j=0}^{d-2} q_j x^j}_{\tilde{q}_1(x)} + q_{d-1} x^{d-1} + \underbrace{\sum_{j=d}^{2d-2} q_j x^j}_{\tilde{q}_2(x)} \quad (39)$$

We show that the term of degree $d^i - 1$ in $p(x)q(x^{d^{i-1}})$ is only generated by multiplication of the term of degree $d^{i-1}-1$ in $p(x)$ and the term of degree $d^{i-1}(d-1)$ in $q(x^{d^{i-1}})$. For this purpose, we consider following terms:

1. Consider the multiplication of two lowest degree terms in $\tilde{p}_1(x)$ and $\tilde{q}_2(x^{d^{i-1}})$ of equations (38) and (39). That is, $q_{d^i} x^{d^i} p_0 = p_0 q_{d^i} x^{d^i}$ which has higher degree in comparison to x^{d^i-1}. Consequently, the degree of any term in the multiplication of $\tilde{p}_1(x)$ and $\tilde{q}_2(x^{d^{i-1}})$ will be strictly greater than $d^i - 1$.
2. Consider the multiplication of two highest degree of terms in $\tilde{p}_2(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ of equations (38) and (39),
$$q_{d-2} x^{d^{i-1}(d-2)} p_{2d^{i-1}-2} x^{2d^{i-1}-2} = q_{d-2} p_{2d^{i-1}-2} x^{d^i-2}$$
is less than d^i-1. Consequently, the degree of any term in the multiplication of $\tilde{p}_2(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ will be strictly less than $d^i - 1$.
3. Since the degree of any term in the multiplication of $\tilde{p}_2(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ is strictly less than $d^i - 1$, and any term in $\tilde{p}_1(x)$ has degree less than the degree of any term in $\tilde{p}_2(x)$, we conclude that any term in the multiplication of $\tilde{p}_1(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ has degree strictly less than d^i-1.
4. Since the degree of any term in the multiplication of $\tilde{p}_1(x)$ and $\tilde{q}_2(x^{d^{i-1}})$ is strictly greater than $d^i - 1$, and any term in $\tilde{p}_2(x)$ has degree larger than the degree of any term in $\tilde{p}_1(x)$, we conclude that any term in $\tilde{p}_2(x)\tilde{q}_2(x^{d^{i-1}})$ has degree strictly greater than $d^i - 1$ which completes the proof.

■

Lemma A.2. *If $p(x) = \sum_{j=0}^{2d^{i-1}-2} p_j x^j$ is a polynomial with degree $2d^{i-1}-2$ for some $i \geq 2$, and $q(x) = \sum_{j=0}^{d-1} q_j x^j$ is any other polynomial with degree $d-1$, then, for $0 \leq j \leq d-1$, $p_{d^{i-1}-1} q_j$ are the coefficients of $x^{(j+1)d^{i-1}-1}$ in $p(x)q(x^{d^{i-1}})$.*

Proof. First, we expand out $p(x)$ as in (38), and expand $q(x)$ as follows:

$$q(x) = \underbrace{\sum_{k=0}^{j-1} q_k x^k}_{\tilde{q}_1(x)} + q_j x^j + \underbrace{\sum_{k=j+1}^{d-1} q_j x^j}_{\tilde{q}_2(x)} \quad (40)$$

In order to prove Lemma A.2, we show that $x^{(j+1)d^{i-1}-1}$ term in $p(x)q(x^{d^{i-1}})$ is produced solely by the multiplication of the term $p_{d^{i-1}-1} x^{d^{i-1}-1}$ in $p(x)$ with the term $q_j x^{j(d^{i-1})}$ in $q(x^{d^{i-1}})$. First, it is clear that the product of the term $p_{d^{i-1}-1} x^{d^{i-1}-1}$ in $p(x)$ with the term $q_j x^{j(d^{i-1})}$ in $q(x^{d^{i-1}})$ has degree $(j+1)d^{i-1}-1$. Thus, to complete the proof, we show that no other terms in $p(x)$ produce $x^{(j+1)d^{i-1}-1}$ term when multiplied with any term in $q(x^{d^{i-1}})$. To do so, we consider the following terms:

1. Consider the multiplication of two lowest degree terms in $\tilde{p}_1(x)$ and $\tilde{q}_2(x^{d^{i-1}})$ as defined in (38) and (40). That is, $p_0 q_{j+1} x^{(j+1)d^{i-1}}$ which has higher degree in comparison to $x^{(j+1)d^{i-1}-1}$. Consequently, the degree of any term in the multiplication of $\tilde{p}_1(x)$ and $\tilde{q}_2(x^{d^{i-1}})$ will be strictly greater than $(j+1)d^{i-1}-1$.
2. Consider the multiplication of two highest degree of terms in $\tilde{p}_2(x)$ and $\tilde{q}_1(x^{d^{i-1}})$, the product
$$q_{j-1} x^{(j-1)d^{i-1}} p_{2d^i-2} x^{2d^{i-1}-2} = q_{j-1} p_{2d^i-2} x^{(j+1)d^{i-1}-2}$$
has degree less than $(j+1)d^{i-1}-1$. Consequently, the degree of any term in the multiplication of $\tilde{p}_2(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ will be strictly less than $(j+1)d^{i-1}-1$.
3. Since the degree of any term in the multiplication of $\tilde{p}_2(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ is strictly less than $(j+1)d^{i-1}-1$, and the degree of any term in $\tilde{p}_1(x)$ is less than the degree of any term in $\tilde{p}_2(x)$, we conclude that any term in the multiplication of $\tilde{p}_1(x)$ and $\tilde{q}_1(x^{d^{i-1}})$ has degree strictly less than $(j+1)d^{i-1}-1$.
4. Since the degree of any term in the multiplication of $\tilde{p}_1(x)$ and $\tilde{q}_2(x^{d^{i-1}})$ is strictly greater than $(j+1)d^{i-1}-1$, and the degree of any term in $\tilde{p}_2(x)$ is larger than the degree of any term in $\tilde{p}_1(x)$, we conclude that any

term in $\tilde{p}_2(x)\tilde{q}_2(x^{d^{i-1}})$ has degree strictly greater than $(j+1)d^{i-1}-1$ which completes the proof. ■

Now, we are able to state the following claims.

Claim A.1. *The coefficient of $x^{m^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ is $\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}$, where, for $i \in \{1, \cdots, \lfloor \frac{n}{2} \rfloor\}$, $p_{\mathbf{C}^{(i)}}(x)$ is as defined in (30).*

Proof. We prove the claim iteratively. Since $p_{\mathbf{C}^{(1)}}(x)$ has degree $2m^{i-1}-2$ with $i=2$, and $p_{\mathbf{C}^{(2)}}(x)$ has degree $2m-2$, we have, by Lemma A.1, that the coefficient of x^{m^2-1} in $p_{\mathbf{C}^{(1)}}(x)p_{\mathbf{C}^{(2)}}(x^m)$ is the product of the coefficient of x^{m-1} in $p_{\mathbf{C}^{(1)}}(x)$ and the coefficient of x^{m^2-m} in $p_{\mathbf{C}^{(2)}}(x^m)$. However, from Remark VI.1, we already know that $\mathbf{A}^{(1)}\mathbf{B}^{(1)}$ is the coefficient of x^{m-1} in $p_{\mathbf{C}^{(1)}}(x)$ and that $\mathbf{A}^{(2)}\mathbf{B}^{(2)}$ is the coefficient of x^{m^2-m} in $p_{\mathbf{C}^{(2)}}(x^m)$. Therefore, $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)}\mathbf{B}^{(2)}$ is the coefficient of x^{m^2-1} in $p_{\mathbf{C}^{(1)}}(x)p_{\mathbf{C}^{(2)}}(x^m)$.

Similarly, consider the two polynomials $p'(x) = p_{\mathbf{C}^{(1)}}(x)p_{\mathbf{C}^{(2)}}(x^m)$ and $p_{\mathbf{C}^{(3)}}(x)$. Notice that $p'(x)$ has degree $2m^{i-1}-2$ with $i=3$, and $p_{\mathbf{C}^{(3)}}(x)$ has degree $2m-2$, therefore, from Lemma A.1, the coefficient of x^{m^3-1} in $p'(x)p_{\mathbf{C}^{(3)}}(x^{m^2})$ is the product of the coefficient of x^{m^2-1} in $p'(x)$ and the coefficient of $x^{m^3-m^2}$ in $p_{\mathbf{C}^{(3)}}(x^{m^2})$. However, from the previous step, we already know that $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)}\mathbf{B}^{(2)}$ is the coefficient of x^{m^2-1} in $p'(x)$. In addition, from Remark VI.1, we already know that $\mathbf{A}^{(3)}\mathbf{B}^{(3)}$ is the coefficient of $x^{m^3-m^2}$ in $p_{\mathbf{C}^{(3)}}(x^{m^2})$. Therefore, $\mathbf{A}^{(1)}\mathbf{B}^{(1)}\mathbf{A}^{(2)}\mathbf{B}^{(2)}\mathbf{A}^{(3)}\mathbf{B}^{(3)}$ is the coefficient of x^{m^3-1} in $p'(x)p_{\mathbf{C}^{(3)}}(x^{m^2}) = p_{\mathbf{C}^{(1)}}(x)p_{\mathbf{C}^{(2)}}(x^m)p_{\mathbf{C}^{(3)}}(x^{m^2})$.

Repeating the same procedure, we conclude that $\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}$ is the coefficient of $x^{m^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\prod_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$. ■

Claim A.2. *If $n \geq 3$ and odd, then, for any $j \in \{1, \cdots, m\}$, $\left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_j^{(\lceil \frac{n}{2} \rceil)}$ is the coefficient of $x^{jm^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\prod_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$, where, for $i \in \{1, \cdots, \lceil \frac{n}{2} \rceil\}$, $p_{\mathbf{C}^{(i)}}(x)$ is as defined in (30).*

Proof. First, notice that since the degree of $p_{\mathbf{C}^{(i)}}(x)$ is $2m-2$ for all $i \in \{1, \cdots, \lfloor \frac{n}{2} \rfloor\}$, the degree of $\Pi_{i=1}^{\lfloor \frac{n}{2} \rfloor} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ is $(2m-2)\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} m^{i-1} = 2m^{\lfloor \frac{n}{2} \rfloor} - 2$. In addition, the matrix polynomial $p_{\mathbf{C}^{(\lceil \frac{n}{2} \rceil)}}(x)$ has degree $m-1$. Therefore, from Lemma A.2, for $1 \leq j \leq m$, the product of the coefficient of $x^{m^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\Pi_{i=1}^{\lfloor \frac{n}{2} \rfloor} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ and the coefficient of x^{j-1} in $p_{\mathbf{C}^{(\lceil \frac{n}{2} \rceil)}}(x)$ is the coefficient of $x^{jm^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\Pi_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$. However, we already know, from Claim A.1, that the coefficient of $x^{m^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\Pi_{i=1}^{\lfloor \frac{n}{2} \rfloor} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ is $\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}$, also, by definition, the coefficient of x^{j-1} in $p_{\mathbf{C}^{(\lceil \frac{n}{2} \rceil)}}(x)$ is $\mathbf{A}_j^{(\lceil \frac{n}{2} \rceil)}$. Thus, $\left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_j^{(\lceil \frac{n}{2} \rceil)}$ is the coefficient of $x^{jm^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\prod_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$. ■

Now, we prove Theorem VI.1.

Proof of Theorem VI.1. To prove the theorem, it suffices to show that for Construction VI.1, the fusion node is able to construct $\mathbf{C}$ from any $2m^{n/2}-1$ worker nodes if n is even or from any $(m+1)m^{\lfloor \frac{n}{2} \rfloor}-1$ if n is odd.

First, for the case in which n is even, we need to compute $\mathbf{C} = \prod_{i=1}^{\frac{n}{2}} \mathbf{A}^{(i)}\mathbf{B}^{(i)}$. Notice, from Claim A.1, that the desired matrix product $\mathbf{C}$ is the coefficient of $x^{m^{n/2}-1}$ in $\prod_{i=1}^{\frac{n}{2}} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$. Thus, it is sufficient to compute this coefficient at the fusion node as the computation output for successful computation. Now, because the polynomial $\prod_{i=1}^{\frac{n}{2}} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ has degree $2m^{n/2}-2$, evaluation of the polynomial at any $2m^{n/2}-1$ distinct points is sufficient to compute all of the coefficients of powers of x in $\prod_{i=1}^{\frac{n}{2}} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ using polynomial interpolation. This includes $\mathbf{C}$, the coefficient of $x^{m^{n/2}-1}$.

Now, for the case in which n is odd, we need to compute $\mathbf{C} = \left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}^{(\lceil \frac{n}{2} \rceil)}$. First, notice that $\mathbf{C}$ is a concatenation of the matrices $\left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_j^{(\lceil \frac{n}{2} \rceil)}$, $j \in \{1, \cdots, m\}$ as follows:

$$\mathbf{C} = \left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}^{(\lceil \frac{n}{2} \rceil)} = \left[\left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_1^{(\lceil \frac{n}{2} \rceil)} \middle| \cdots \middle| \left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_m^{(\lceil \frac{n}{2} \rceil)}\right]. \tag{41}$$

From Claim A.2, for all $j \in \{1, \cdots, m\}$, the product $\left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_j^{(\lceil \frac{n}{2} \rceil)}$ is the coefficient of $x^{jm^{\lfloor \frac{n}{2} \rfloor}-1}$ in $\prod_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$. Thus, it is sufficient to compute these coefficients, for all $j \in \{1, \cdots, m\}$, at the fusion node as the computation output for successful computation. Now, because the polynomial $\prod_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ has degree $m^{\lfloor \frac{n}{2} \rfloor}(m+1)-2$, evaluation of the polynomial at any $m^{\lfloor \frac{n}{2} \rfloor}(m+1)-1$ distinct points is sufficient to compute all of the coefficients of powers of x in $\prod_{i=1}^{\lceil \frac{n}{2} \rceil} p_{\mathbf{C}^{(i)}}(x^{m^{i-1}})$ using polynomial interpolation. This includes the coefficients of $x^{jm^{\lfloor \frac{n}{2} \rfloor}-1}$, i.e. $\left(\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} \mathbf{A}^{(i)}\mathbf{B}^{(i)}\right)\mathbf{A}_j^{(\lceil \frac{n}{2} \rceil)}$, for all $j \in \{1, \cdots, m\}$. ■

Appendix B
Proof of Theorem VI.2

Proof of Theorem VI.2. Here, we only derive the proof for the case of even n. The proof for odd n can be derived in a similar manner with minor differences in the expressions. What we have to show to complete the proof are as follows:

Claim B.1. *The maximum degree of $p_{\mathbf{C}}(x)$ is $s^{\frac{n}{2}}t^{\frac{n}{2}+1} + s^{\frac{n}{2}}t^{\frac{n}{2}-1} - 1$.*

Claim B.2. *$\mathbf{C}_{i,j}$ is the coefficient of $x^{d(n,i,j)}$ for $i,j = 1, \cdots, t$ where*

$$d(n,i,j) = s-1+s(t-1)+st(s-1)+\cdots+i\cdot s^{\frac{n}{2}}t^{\frac{n}{2}-1}+j\cdot s^{\frac{n}{2}}t^{\frac{n}{2}}. \tag{42}$$

Claim B.3. *$x^{d(n,i,j)}$ term is obtained only when: i) $i_1 = i$, ii) $j_1 = i_2, \cdots, j_{n-1} = i_n$, iii) $j_n = j$.*

Let us first rewrite $p_{\mathbf{C}}(x)$ as follows:

$$p_{\mathbf{C}}(x) = \sum_{\substack{i_1=1\cdots t,\cdots,i_n=1\cdots s \\ j_1=1\cdots s,\cdots,j_n=1\cdots t}} \mathbf{A}^{(1)}_{i_1,j_1}\mathbf{B}^{(1)}_{i_2,j_2}\cdots\mathbf{A}^{(n/2)}_{i_{n-1},j_{n-1}}\mathbf{B}^{(n/2)}_{i_n,j_n} x^{(s-1+j_1-i_2)+\cdots+i_1 s^{\frac{n}{2}}t^{\frac{n}{2}-1}+j_n s^{\frac{n}{2}}t^{\frac{n}{2}}}. \quad (43)$$

Note that we get the maximum degree when $i_1 = t-1, s-1+j_1-i_2 = 2s-2, \cdots, j_n = t-1$. Hence,

$$\begin{aligned}\text{max deg of } p_{\mathbf{C}}(x) &= 2s-2+s(2t-2)+\cdots+ \\ &\quad s^{n/2-1}t^{n/2-1}(2s-2)+(t-1)s^{n/2}t^{n/2-1} \\ &\quad +(t-1)s^{n/2}t^{n/2} \\ &= s^{n/2}t^{n/2-1}+s^{n/2}t^{n/2+1}-2 \\ &= k(n,s,t)-1.\end{aligned}$$

This shows Claim B.1. To show Claim B.2, note that $\mathbf{C}_{i,j} = \sum_{j_1,j_2,\cdots,j_{n-1}} \mathbf{A}^{(1)}_{i,j_1}\mathbf{B}^{(1)}_{j_1,j_2}\mathbf{A}^{(2)}_{j_2,j_3}\mathbf{B}^{(2)}_{j_3,j_4}\cdots\mathbf{A}^{(n/2)}_{j_{n-2},j_{n-1}}\mathbf{B}^{(n/2)}_{j_{n-1},j}$. Among the terms in the sum in (43), $\mathbf{C}_{i,j}$ is the sum of terms that are from the i-th row of the first matrix $\mathbf{A}^{(1)}$ and the j-th column on the last matrix $\mathbf{B}^{(n/2)}$, and that have the second index and the first index of two adjacent matrices matching, e.g., $j_1 = i_2$ and $j_2 = i_3$. By setting these $i_1,\cdots,i_n,j_1,\cdots,j_n$ values, we obtain (42).

Lastly, we want to show Claim B.3. Let d be the degree of x in (43)

$$\begin{aligned}d = (s-1+j_1-i_2)+\cdots+s^{\frac{n}{2}-1}t^{\frac{n}{2}-1}(s-1+j_{n-1}-i_n) \\ +i_1 s^{\frac{n}{2}}t^{\frac{n}{2}-1}+j_n s^{\frac{n}{2}}t^{\frac{n}{2}}, \quad (44)\end{aligned}$$

which can be rewritten as:

$$d = d_0+d_1\cdot s+d_2\cdot st+\cdots+d_{n-1}\cdot s^{\frac{n}{2}}t^{\frac{n}{2}-1}+d_n\cdot s^{\frac{n}{2}}t^{\frac{n}{2}}, \quad (45)$$

where

$$\begin{aligned}d_0 &= d \mod s \\ d_1 &= (d-d_0)/s \mod t \\ d_2 &= (d-d_0-d_1\cdot s)/st \mod s \\ &\vdots \\ d_n &= (d-d_0-d_1\cdot t-\cdots-d_{n-1}\cdot s^{n/2}t^{n/2-1})/s^{n/2}t^{n/2}.\end{aligned}$$

We can think of this representation as a mixed radix system $\mathcal{D}$ with $n+2$ digits, $(d_0,d_1,\cdots,d_{n+1})$, which has an alternating radix $(t,s,t,s,\cdots,t,s)$. By substituting $d_0 = t-1, d_1 = s-1,\cdots,d_{n+1} = s-1$, we can confirm that the biggest number we can represent with (45) is $s^{n+1}t^{n+1}-1 > k(n,s,t)-1$. Also, from its construction, any number between 0 and $s^{n+1}t^{n+1}-1$ can be uniquely determined by the pair $(d_0,d_1,\cdots,d_{n+1})$ (for more explanation, see Theorem 1 in [65]). Hence, any $0 \le d \le k(n,s,t)-1$ can be uniquely represented with $(d_0,d_1,\cdots,d_{n+1})$.

Now, we want to show that $d = d(n,i,j)$ only when $d_0 = s-1, d_1 = t-1,\cdots,d_{n-3} = t-1, d_{n-2} = s-1$ and $d_{n-1} = i, d_n = j$. It is easy to see that $d_0 = d(n,i,j) \mod s = s-1$, and similarly $d_1 = (d(n,i,j)-d_0) \mod t = t-1$ and so on. Since i_1 varies only from 0 to $t-1$,

$$\begin{aligned}d_{n-1} &= (i\cdot s^{\frac{n}{2}}t^{\frac{n}{2}-1}+j\cdot s^{\frac{n}{2}}t^{\frac{n}{2}})/s^{n/2}t^{n/2-1} \mod t \\ &= (i+jt) \mod t \\ &= i.\end{aligned}$$

Finally, $d_n = (j\cdot s^{\frac{n}{2}}t^{\frac{n}{2}})/s^{\frac{n}{2}}t^{\frac{n}{2}} = j$. As there is only one unique representation of any d with a tuple $(d_0,d_1,\cdots,d_n)$, by comparing (44) and (45), we can conclude that $j_1 = i_2,\cdots,j_{n-1} = i_n$, and $i_1 = i, j_n = j$. This completes the proof.

■

ACKNOWLEDGMENTS

We thank Mayank Bakshi and Yaoqing Yang for helpful discussions. We acknowledge support from NSF CNS-1702694, CCF-1553248, CCF-1464336, and CCF-1350314. This work was also supported in part by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

REFERENCES

[1] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *IEEE Communication, Control, and Computing (Allerton)*, Oct 2017, pp. 1264–1270.

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

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

[4] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *IEEE Communication, Control, and Computing (Allerton)*, 2015, pp. 964–971.

[5] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," in *Machine Learning Systems Workshop, Advances in Neural Information Processing Systems (NIPS)*, 2016.

[6] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," in *Advances In Neural Information Processing Systems (NIPS)*, 2017, pp. 4403–4413.

[7] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 989–997, 2014.

[8] D. Wang, G. Joshi, and G. Wornell, "Using Straggler Replication to Reduce Latency in Large-Scale Parallel Computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[9] D. Wang, G. Joshi, and G. Wornell, "Efficient Task Replication for Fast Response Times in Parallel Computation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1, pp. 599–600, 2014.

[10] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products," in *Advances In Neural Information Processing Systems (NIPS)*, 2016, pp. 2092–2100.

[11] N. Azian-Ruhi, A. S. Avestimehr, F. Lahouti, and B. Hassibi, "Consensus-based distributed computing," in *Information Theory and Applications Workshop*, 2017.

[12] Y. Yang, P. Grover, and S. Kar, "Fault-tolerant Distributed Logistic Regression Using Unreliable Components," in *IEEE Communication, Control, and Computing (Allerton)*, 2016, pp. 940–947.

[13] Y. Yang, P. Grover, and S. Kar, "Computing Linear Transformations With Unreliable Components," *IEEE Transactions on Information Theory*, vol. 63, no. 6, pp. 3729–3756, 2017.

[14] Y. Yang, P. Grover, and S. Kar, "Fault-tolerant parallel linear filtering using compressive sensing," in *IEEE International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2016, pp. 201–205.

[15] S. Li, M. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A Fundamental Tradeoff Between Computation and Communication in Distributed Computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, 2018.

[16] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A Unified Coding Framework for Distributed Computing with Straggling Servers," in *Globecom Workshops (GC Wkshps)*, 2016, pp. 1–6.

[17] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded TeraSort," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 389–398.

[18] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded Distributed Computing: Straggling Servers and Multistage Dataflows," in *IEEE Communication, Control, and Computing (Allerton)*, 2016, pp. 164–171.

[19] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 3368–3376.

[20] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic mds codes and expander graphs," in *International Conference on Machine Learning (ICML)*, 2018, pp. 4302–4310.

[21] M. Aktas, P. Peng, and E. Soljanin, "Effective Straggler Mitigation: Which Clones Should Attack and When?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 12–14, 2017.

[22] M. Aktas, P. Peng, and E. Soljanin, "Straggler Mitigation by Delayed Relaunch of Tasks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 224–231, 2018.

[23] M. Aliasgari, J. Kliewer, and O. Simeone, "Coded computation against processing delays for virtualized cloud-based channel decoding," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 28–38, 2019.

[24] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2408–2412.

[25] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2027–2031.

[26] C. Karakus, Y. Sun, and S. Diggavi, "Encoded distributed optimization," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2890–2894.

[27] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler Mitigation in Distributed Optimization through Data Encoding," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5440–5448.

[28] Y. Yang, P. Grover, and S. Kar, "Coded Distributed Computing for Inverse Problems," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 709–719.

[29] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *IEEE Communication, Control, and Computing (Allerton)*, 2017, pp. 1256–1263.

[30] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2413–2417.

[31] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded fourier transform," in *IEEE Communication, Control, and Computing (Allerton)*, 2017, pp. 494–501.

[32] H. Jeong, T. M. Low, and P. Grover, "Masterless Coded Computing: A Fully-Distributed Coded FFT Algorithm," in *IEEE Communication, Control, and Computing (Allerton)*, 2018, pp. 887–894.

[33] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-Proofing Massive-Scale Distributed Matrix Multiplication with D-Dimensional Product Codes," in *IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1993–1997.

[34] G. Suh, K. Lee, and C. Suh, "Matrix sparsification for coded matrix multiplication," in *IEEE Communication, Control, and Computing (Allerton)*, 2017, pp. 1271–1278.

[35] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 8192–8196.

[36] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *International Conference on Machine Learning (ICML)*, 2018, pp. 5139–5147.

[37] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," *arXiv preprint arXiv: 1804.09791*, 2018.

[38] A. Severinson, A. G. i Amat, and E. Rosnes, "Block-Diagonal and LT Codes for Distributed Computing With Straggling Servers," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 1739–1753, 2019.

[39] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *International Conference on Machine Learning (ICML)*, 2018, pp. 5606–5615.

[40] K. H. Huang and J. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 518–528, 1984.

[41] T. Herault and Y. Robert, *Fault-Tolerance Techniques for High Performance Computing*. Springer, 2015.

[42] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2403–2407.

[43] V. Cadambe and P. Grover, "Codes for Distributed Computing: A Tutorial," *IEEE Information Theory Society Newsletter*, vol. 67, no. 4, pp. 3–15, Dec. 2017.

[44] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2418–2422.

[45] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2022–2026.

[46] 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 International Symposium on Information Theory (ISIT)*, 2018, pp. 1585–1589.

[47] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[48] H. T. Kung, "Fast evaluation and interpolation," Carnegie Mellon University, Tech. Rep., 1973.

[49] K. S. Kedlaya and C. Umans, "Fast polynomial factorization and modular composition," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1767–1802, 2011.

[50] I. Gohberg and V. Olshevsky, "The fast generalized Parker-Traub algorithm for inversion of Vandermonde and related matrices," *Journal of Complexity*, vol. 13, no. 2, pp. 208–234, 1997.

[51] A. Bjorck and V. Pereyra, "Solution of Vandermonde systems of equations," *Mathematics of Computation*, vol. 24, no. 112, pp. 893–903, 1970.

[52] I. Kaufman, "The inversion of the Vandermonde matrix and transformation to the Jordan canonical form," *IEEE Transactions on Automatic Control*, vol. 14, no. 6, pp. 774–777, 1969.

[53] F. Parker, "Inverses of Vandermonde matrices," *The American Mathematical Monthly*, vol. 71, no. 4, pp. 410–411, 1964.

[54] J. F. Traub, "Associated polynomials and uniform methods for the solution of linear problems," *Siam Review*, vol. 8, no. 3, pp. 277–301, 1966.

[55] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[56] U. Sheth, S. Dutta, M. Chaudhari, H. Jeong, Y. Yang, J. Kohonen, T. Roos, and P. Grover, "An Application of Storage-Optimal MatDot Codes for Coded Matrix Multiplication: Fast k-Nearest Neighbors Estimation," in *IEEE Big Data (Short Paper)*, 2018.

[57] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[58] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[59] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 954–967, 2014.

[60] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "GRASS: Trimming Stragglers in Approximation Analytics," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 289–302.

[61] F. Haddadpour, Y. Yang, V. Cadambe, and P. Grover, "Cross-Iteration Coded Computing," in *IEEE Communication, Control, and Computing (Allerton)*, 2018, pp. 196–203.

[62] S. Dutta, Z. Bai, T. M. Low, and P. Grover, "CodeNet: Training Large Scale Neural Networks in Presence of Soft-Errors," *Workshop on Coding Theory for Large-Scale Machine Learning, International Conference on Machine Learning (ICML)*, 2019.

[63] Y. Yang, P. Grover, and S. Kar, "Can a noisy encoder be used to communicate reliably?" in *IEEE Communication, Control, and Computing (Allerton)*, 2014, pp. 659–666.

[64] M. G. Taylor, "Reliable information storage in memories designed from unreliable components," *Bell System Technical Journal*, vol. 47, no. 10, pp. 2299–2337, 1968.

[65] A. S. Fraenkel, "Systems of numeration," *The American Mathematical Monthly*, vol. 92, no. 2, pp. 105–114, 1985.

Sanghamitra Dutta [S'15] received her B.Tech in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 2015.

She is a doctoral candidate in the Department of Electrical and Computer Engineering at Carnegie Mellon University, PA, USA. She was a summer research intern at the IBM TJ Watson Research Center from May 2017 to August 2017. Her main contributions to science are towards developing novel erasure-codes for reliable computing in presence of faults, stragglers and errors, and deriving fundamental information-theoretic limits on their performance. She is interested in novel algorithmic solutions for reliable and trustworthy machine learning, that address computational challenges of large-scale machine learning as well as the emerging trust issues concerning fairness and privacy.

Ms. Dutta is a recipient of the 2019 Axel Berny Presidential Graduate Fellowship, 2017 Tan Endowed Graduate Fellowship, 2016 Prabhu and Poonam Goel Graduate Fellowship and the 2014 HONDA Young Engineer and Scientist Award.

Mohammad Fahim [S'17] received his B.Sc. degree (Hons.) and M.Sc. degree in electrical engineering from Alexandria University, Alexandria, Egypt, in 2012 and 2015, respectively.

He is a Graduate Research Assistant at the School of Electrical Engineering and Computer Science, Pennsylvania State University, PA, USA since August 2015. He was a Research Assistant at the American University in Cairo between 2012 and 2015, and a Teaching Assistant at Alexandria University, Egypt between 2013 and 2015. He did an internship at Texas A&M University at Qatar on summer 2013. His previous research interests include space time codes in wireless communications and network coding. He is currently interested in providing secure, communication-efficient and fault-tolerant coding techniques in large scale distributed computing systems using tools from information theory and coding theory.

Farzin Haddadpour received the B.Sc. degree in electrical engineering from University of Tabriz, Tabriz, Iran, in 2010 and his M.Sc. degree from Sharif University of Technology, Tehran, Iran in 2012, respectively. He is currently working towards the Ph.D. degree in the Department of Electrical Engineering and Computer Science, Pennsylvania State University, State College, PA. He was awarded the Trust scholarship from the Cambridge University in 2014. His research interests include distributed optimization for machine learning problems and coding and information theory.

Haewon Jeong [M.S. CMU 16, B.S. KAIST 14] is a PhD student at Carnegie Mellon University. Her main contributions to science are towards solving a 60-year-old problem that was raised by Von Neumann: can we reliably compute in the presence of noise? By connecting classical theory of error correcting codes and large-scale parallel computing, she has developed several computing strategies using unreliable processors (coded computing), including her recent work on an optimal matrix multiplication strategy under uncertainty in computing. She is also interested in green communication, and came up with an idea of energy-adaptive codes. She received 2014 NSDI Community Award and 2014 Samsung HumanTech Paper Award.

Viveck Cadambe [M'06] received his Ph.D in electrical and computer engineering from the University of California, Irvine, CA, USA in 2011. He received his B.Tech and M.Tech in electrical engineering from the Indian Institute of Technology Madras, Chennai, India, in 2006.

He is an Assistant Professor in the Department of Electrical Engineering at Pennsylvania State University, University Park, PA USA. Between 2011 and 2014, he was a postdoctoral researcher, jointly with the Electrical and Computer Engineering (ECE) department at Boston University, and the Research Laboratory of Electronics (RLE) at the Massachusetts Institute of Technology (MIT). His research uses tools of information theory, error correcting codes and theory of distributed systems to understand fundamental engineering trade-offs in data communication, storage and computing systems.

Dr. Cadambe is a recipient of the 2009 IEEE Information Theory Society Best Paper Award, 2011 CPCC Best Dissertation Award from University of California, Irvine, the 2014 IEEE International Symposium on Network Computing and Applications (NCA) Best Paper Award, the 2015 NSF CRII Award, the 2016 NSF Career Award and a finalist for the 2016 Bell Labs Prize. He has served as an Associate Editor for the IEEE Transactions on Wireless Communications since December 2014.

Pulkit Grover [Ph.D. UC Berkeley'10, SM'16] is an Associate Professor at CMU in Electrical and Computer Engineering and Carnegie Mellon Neurosciences Institute. His main contributions to science are towards developing and experimentally validating a new theory of information (fundamental limits, practical designs) for optimizing designs of artificial, as well as understanding designs of biological, communication and computing systems. This includes developing formal mathematical tools for estimating flows of information and minimizing energy in computing. Pulkit received the 2010 best student paper award at IEEE Conference on Decision and Control; the 2011 Eli Jury Dissertation Award from UC Berkeley; the 2012 IEEE Leonard G. Abraham journal paper award; a 2014 NSF CAREER award; a 2015 Google Research Award; and the 2019 Best Tutorial Paper Award from IEEE ComSoc. In 2018, he received the inaugural award from the Chuck Noll Foundation for Brain Injury Research and the Joel and Ruth Spira Excellence in Teaching Award.