Efficient and Robust Distributed Matrix Computations via Convolutional Coding

Anindya Bijoy Das, *Graduate Student Member, IEEE*, Aditya Ramamoorthy, *Senior Member, IEEE*, and Namrata Vaswani, *Fellow, IEEE*

***Abstract*— Distributed matrix computations – matrix-matrix or matrix-vector multiplications – are well-recognized to suffer from the problem of stragglers (slow or failed worker nodes). Much of prior work in this area is (i) either sub-optimal in terms of its straggler resilience, or (ii) suffers from numerical problems, i.e., there is a blow-up of round-off errors in the decoded result owing to the high condition numbers of the corresponding decoding matrices. Our work presents a convolutional coding approach to this problem that removes these limitations. It is optimal in terms of its straggler resilience, and has excellent numerical robustness as long as the workers' storage capacity is slightly higher than the fundamental lower bound. Moreover, it can be decoded using a fast peeling decoder that only involves add/subtract operations. Our second approach has marginally higher decoding complexity than the first one, but allows us to operate arbitrarily close to the storage capacity lower bound. Its numerical robustness can be theoretically quantified by deriving a computable upper bound on the worst case condition number over all possible decoding matrices by drawing connections with the properties of large block Toeplitz matrices. All above claims are backed up by extensive experiments done on the AWS cloud platform.**

***Index Terms*— Distributed computing, straggler, convolutional coding, Toeplitz matrix, Vandermonde matrix.**

I. Introduction

DISTRIBUTED computing clusters are heavily used in domains such as machine learning where datasets are often so large that they cannot be stored in a single computer. The widespread usage of such clusters presents several opportunities and advantages over traditional computing paradigms. However, they also present newer challenges. Large scale clusters which can be heterogeneous in nature suffer from the issue of stragglers (slow or failed workers in the system). Fig. 1 shows the variation of speed of different `t2.micro` machines in AWS (Amazon Web Services) cluster, and it can be seen that for a particular job, a slow worker node may require around $40\% - 50\%$ more time than the average.

Manuscript received June 1, 2020; revised January 14, 2021; accepted June 23, 2021. Date of publication July 9, 2021; date of current version August 25, 2021. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1718470 and Grant CCF-1910840. This article was presented in part at the 2021 IEEE International Symposium on Information Theory (ISIT). *(Corresponding author: Aditya Ramamoorthy.)*

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (e-mail: abd149@iastate.edu; adityar@iastate.edu; namrata@iastate.edu).

Communicated by M. Schwartz, Associate Editor for Coding Techniques.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIT.2021.3095909.

Digital Object Identifier 10.1109/TIT.2021.3095909

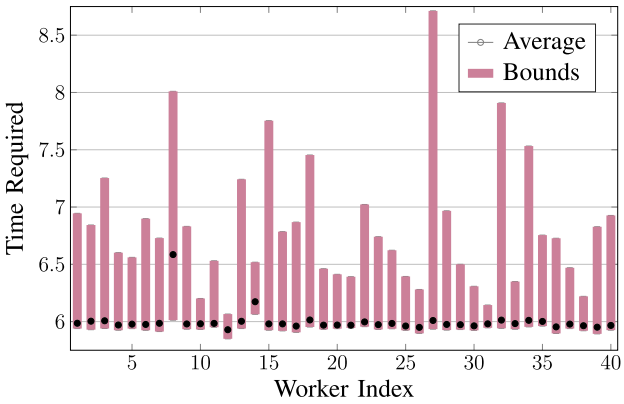

Fig. 1. Variation of worker speeds for the same job over 100 runs across 40 workers within AWS; the job involves multiplying two random matrices of size 4000×4000 twice. The average time is shown by the small circle for each worker. The upper and lower edges indicate the maximum and minimum time over the 100 runs. The required time exhibits a wide variation from 5.85 seconds to 8.71 seconds.

The conventional approach [1] to tackle stragglers has been to run multiple copies of tasks on various machines, with the hope that at least one copy finishes on time. For instance, consider matrix-vector multiplication with a matrix $\mathbf{A}$ and vector $\mathbf{x}$, where our goal is to obtain the product $\mathbf{A}^T\mathbf{x}$ in a distributed fashion. Fig. 2 shows an example where we partition $\mathbf{A}$ into four block-columns and we assign two block-columns to each of the four worker nodes. Thus each block-column has been assigned twice over all four workers and we can verify that we recover the final result if *any* three workers finish their respective jobs. In other words, we can say that this scheme is resilient to one straggler.

However, this toy example can be made even more efficient in terms of resource utilization by dividing $\mathbf{A}$ into two block-columns $\mathbf{A}_0$ and $\mathbf{A}_1$ and assigning the worker nodes appropriate linear combinations of $\mathbf{A}_0$ and $\mathbf{A}_1$ so that the required result can be decoded from any *two* workers. This is the basic idea underlying "coded computation" (introduced in the work of Lee *et al.* [2]). It leverages ideas from erasure coding to introduce redundancy in the computation performed by the worker nodes. Roughly speaking, as long as enough worker nodes complete their tasks, the central node can decode the intended result by appropriate post-processing.

The central problem within coded distributed matrix computation can be explained as follows. Suppose that we have

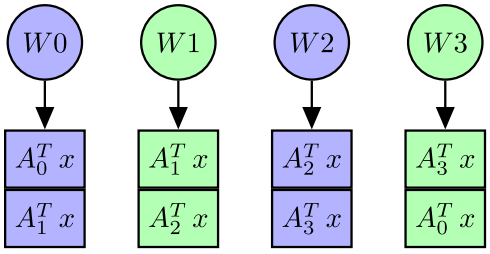

Fig. 2. Matrix A is divided into four submatrices. Each worker is assigned two of the submatrices and the vector x.

large matrices $\mathbf{A} \in \mathbb{R}^{t\times r}, \mathbf{B} \in \mathbb{R}^{t\times w}$ and a vector $\mathbf{x} \in \mathbb{R}^t$. The goal is to either compute $\mathbf{A}^T\mathbf{x}$ (matrix-vector multiplication) or $\mathbf{A}^T\mathbf{B}$ (matrix-matrix multiplication) in a distributed fashion using n worker nodes while being resistant to any s stragglers. Redundancy is introduced in the computation by coding across appropriately chosen submatrices of $\mathbf{A}$ and $\mathbf{B}$ and assigning the worker nodes appropriate computation responsibilities.

The main finding of several recent works in this area is that it is possible to embed distributed matrix computations into the structure of an equivalent erasure code, where the failed nodes play the role of erasures [3]–[9] (we discuss related work in detail shortly). A given coded computation scheme is said to have threshold τ if the desired result can be decoded as long as any τ worker nodes return their results to the central node. This has been the focus of many works in the literature.

In this work, we consider the important issue of numerical stability within coded computation (in addition to threshold). We point out that several of the existing schemes in the literature suffer from significant numerical issues in the decoding process. In particular, the system of equations that is solved by the central node in the decoding step can have a very high condition number which in turn results in a large error in the decoded result. We present a novel scheme based on convolutional codes (operating over the reals) that simultaneously addresses numerical stability, the threshold, and possesses easy encoding/decoding. An overview of the properties of most of the known schemes in the literature is presented in Table I.

This paper is organized as follows. Section II explains the problem formulation and Section III describes the background and related work and summarizes of the contributions of our work. Section IV discusses our main ideas on how convolutional codes can be used to address distributed matrix computations, Section V overviews the analysis of numerical stability for our codes and Section VI discusses the experimental performance of our proposed methods and shows the comparison with other available approaches. We conclude the paper with a discussion about future work in Section VII. For the sake of readability several of the proofs appear in the Appendix.

II. Problem Formulation

In the matrix-vector case we partition $\mathbf{A}$ into submatrices of equal size and $\mathbf{x}$ into subvectors and distribute a certain number of "coded" versions of these submatrices and subvectors to the n workers (subject to a storage constraint). Every worker computes the product of its assigned submatrices and subvectors and sends the computed result back to the central node. The central node then "decodes" to recover $\mathbf{A}^T\mathbf{x}$.

In the matrix-matrix multiplication scenario, each worker node receives coded versions of submatrices of $\mathbf{A}$ and coded versions of the submatrices of $\mathbf{B}$, respectively[1]. It computes pairwise products (either all or some subset thereof) of these and sends them to the central node which needs to decode to recover $\mathbf{A}^T\mathbf{B}$.

In the discussion below we discuss the matrix-matrix scenario; it applies in a natural way to the matrix-vector case as well. We consider a $p \times u$ and $p \times v$ block decomposition of $\mathbf{A}$ and $\mathbf{B}$ respectively as shown below.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \cdots & \mathbf{A}_{0,u-1} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{p-1,0} & \cdots & \mathbf{A}_{p-1,u-1} \end{bmatrix};$$

$$\text{and } \mathbf{B} = \begin{bmatrix} \mathbf{B}_{0,0} & \cdots & \mathbf{B}_{0,v-1} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{p-1,0} & \cdots & \mathbf{B}_{p-1,v-1} \end{bmatrix}.$$

The central node encodes by computing appropriate scalar linear combinations of the $\mathbf{A}_{i,j}$ submatrices and respectively the $\mathbf{B}_{i,j}$ submatrices. This implies that the central node only performs scalar multiplications and additions. It is not responsible for any of the computationally intensive matrix operations. Following this, it sends the corresponding coded submatrices to each of the workers.

We assume that a worker node cannot store the whole matrix $\mathbf{A}$ or $\mathbf{B}$. Each worker can store the equivalent of γ_A fraction of matrix $\mathbf{A}$ and γ_B fraction of matrix $\mathbf{B}$; this is referred to as the storage fraction. The assumption is that some nodes will fail or will be too slow, the maximum number of such nodes is assumed to be s or less. Here we also assume that all the results suffer from some numerical errors (for example, round-off errors due to limited numerical precision); this is inevitable in real computations. In our work we do not consider adversarial errors introduced by the worker nodes that may be of much larger magnitudes and limited to only a subset of the nodes. The goal is to design the coding scheme so that (i) the decoding is possible using the output of any $k = (n - s)$ workers (k is often called the recovery threshold of the scheme), (ii) it is robust to noise (both numerical precision errors and other sources of noise); and (iii) it is efficiently decodable. We say that the threshold of a scheme is *optimal* if it is the lowest possible given the storage constraints.

III. Background, Related Work and Summary of Contributions

In recent years, several coded computation schemes have been proposed for matrix multiplication [3]–[9], [13]–[15]. We illustrate the basic idea below using the polynomial code approach of [5]. These ideas are presented in a tutorial fashion in [16].

[1]A general formulation need not restrict the assignment to coded submatrices of $\mathbf{A}$ and $\mathbf{B}$. Nevertheless, all known schemes thus far and our proposed schemes work with equal-sized submatrices, so we present the formulation in this way.

TABLE I

COMPARISON WITH EXISTING WORKS [2], [5], [8], [10] AND PARALLEL WORKS [11], [12] IN TERMS OF DIFFERENT PROPERTIES OF THE ALGORITHMS. DECODING COMPLEXITY IS MENTIONED FOR s STRAGGLERS WITH RECOVERY THRESHOLD k WHERE $\mathbf{A} \in \mathbb{R}^{t\times r}$ AND $\mathbf{B} \in \mathbb{R}^{t\times w}$. T AND q ARE DECODING ALGORITHM PARAMETERS FOR THE RANDOM CONVOLUTIONAL CODE, DISCUSSED IN SECTION V, WHERE $T, q \ll r, w$. THE NUMERICAL STABILITY OF THE PRODUCT CODE DEPENDS ON THE UNDERLYING SCHEME THAT IS USED FOR THE COMPONENTS

CODES	MAT-MAT MULT?	OPTIMAL THRESHOLD?	NUMERICAL STABILITY?	DECODING COMPLEXITY FOR MAT-MAT MULT
REPETITION CODES	✓	✗	✓	ZERO
RATELESS CODES [8]	✗	✗	✓	✗
PRODUCT CODES [2]	✓	✗	✓	$O(r^3)$, ASSUMING $r = w$
POLYNOMIAL CODES [5]	✓	✓	✗	$O(rwk)$
ORTHO-POLY CODES [10]	✓	✓	✓	$O(rwk)$
CIRCULANT AND ROTATION MATRIX [12]	✓	✓	✓	$O(rwk)$
RANDOM KHATRI-RAO CODES ([11], REM. 8 - [5])	✓	✓	✓	$O\left(\frac{rw}{k}s^2\right)$
ALL-ONES-CONV CODE (PROPOSED)	✓	✓	✓	$O(rws)$ (ADD/SUBTRACT OPS)
RANDOM-CONV CODE (PROPOSED)	✓	✓	✓	$\min(T, q) \times O\left(\frac{rw}{k}s^2\right)$

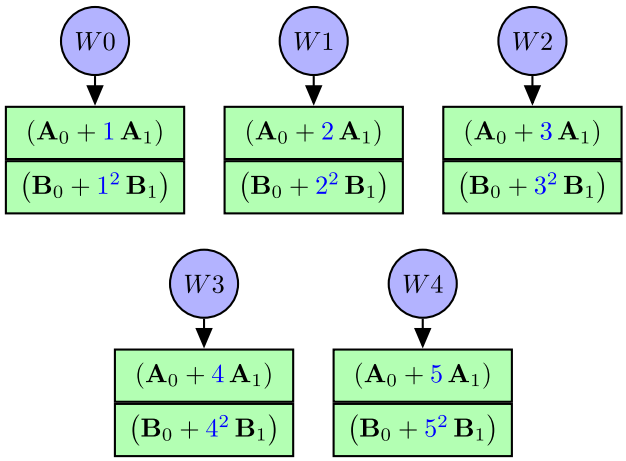

Fig. 3. Matrices $\mathbf{A}$ and $\mathbf{B}$ are divided into two block-columns each. Each worker is assigned one coded submatrix from $\mathbf{A}$ and another coded submatrix from $\mathbf{B}$.

Consider a scenario with $n = 5$ workers where each of these worker nodes can store $\gamma_A = \frac{1}{2}$ fraction of matrix $\mathbf{A}$ and $\gamma_B = \frac{1}{2}$ fraction of matrix $\mathbf{B}$. Consider $u = v = 2$ and $p = 1$, thus we partition both $\mathbf{A}$ and $\mathbf{B}$ into two block-columns $\mathbf{A}_0, \mathbf{A}_1$ and $\mathbf{B}_0, \mathbf{B}_1$ respectively. Next, we define two matrix polynomials as

$$\mathbf{A}(z) = \mathbf{A}_0 + \mathbf{A}_1 z \quad \text{and} \quad \mathbf{B}(z) = \mathbf{B}_0 + \mathbf{B}_1 z^2;$$
$$\text{so } \mathbf{A}^T(z)\mathbf{B}(z) = \mathbf{A}_0^T\mathbf{B}_0 + \mathbf{A}_1^T\mathbf{B}_0 z + \mathbf{A}_0^T\mathbf{B}_1 z^2 + \mathbf{A}_1^T\mathbf{B}_1 z^3.$$

The central node evaluates these polynomial $\mathbf{A}(z)$ and $\mathbf{B}(z)$ at distinct real values $z_0, z_1, \ldots, z_{n-1}$, and sends the corresponding matrices to worker node W_i (see Fig. 3 where $z_i = i + 1$). Each worker node computes the product of its assigned submatrices. It follows that decoding at the central node is equivalent to decoding a degree-3 real-valued polynomial. Thus, the central node can recover $\mathbf{A}^T\mathbf{B}$ as soon as it receives the results from *any* four workers. Thus, in this example, the recovery threshold is, $k = 4$ and the system is resilient to $s = 1$ straggler.

A different solution can be obtained using the approach in [7] for the same example. Let $u = v = 1$ and $p = 2$, so we can write $\mathbf{A}^T\mathbf{B} = \mathbf{A}_0^T\mathbf{B}_0 + \mathbf{A}_1^T\mathbf{B}_1$. Now we define two matrix polynomials as

$$\mathbf{A}(z) = \mathbf{A}_0 z + \mathbf{A}_1 \quad \text{and} \quad \mathbf{B}(z) = \mathbf{B}_0 + \mathbf{B}_1 z;$$
$$\text{so } \mathbf{A}^T(z)\mathbf{B}(z) = \mathbf{A}_1^T\mathbf{B}_0 + \left(\mathbf{A}_0^T\mathbf{B}_0 + \mathbf{A}_1^T\mathbf{B}_1\right) z + \mathbf{A}_0^T\mathbf{B}_1 z^2.$$

As before, the central node will evaluate the polynomial $\mathbf{A}(z)$ and $\mathbf{B}(z)$ at $z_0, z_i, \ldots, z_{n-1}$, and send the corresponding matrices to worker node W_i. It follows that the central node can recover all the unknowns $\left(\text{including}(\mathbf{A}_0^T\mathbf{B}_0 + \mathbf{A}_1^T\mathbf{B}_1)\right)$ as soon as it receives the results from *any* three workers. Thus, in this example, the recovery threshold is, $k = 3$ and the system is resilient to $s = 2$ stragglers.

It should be noted that the latter approach can lead to more straggler resilience, but the computational load per worker has doubled compared to the first approach. Moreover the communication load from the worker nodes to the central node is also higher by a factor of 4 compared to the first approach.

For both schemes above, it can be shown that worker node computation time depends on t, whereas the decoding complexity is independent of it (see for instance [16]). Thus, for scenarios where t is very large, the decoding time can be neglected. Nevertheless, a low decoding complexity is desirable from a practical standpoint.

A. Related Work

As discussed above, [4], [5], [7] convert distributed matrix computation into polynomial evaluation/interpolation, i.e., the coded submatrices correspond to polynomial evaluation maps. We remark here that as far as we are aware, the idea of embedding matrix multiplication using polynomial maps goes back even further to Yagle [17] (the motivation there was fast matrix multiplication).

For fixed storage constraints $\gamma_A = \frac{1}{u}$ and $\gamma_B = \frac{1}{v}$ and for fixed computation overhead per worker with $p = 1$ and arbitrary u and v, the optimal threshold τ is shown to be uv [5] using the polynomial approach. When $p \geq 2$, the work of [4] demonstrates a threshold of $puv + p - 1$. They also present a converse argument which demonstrates that this is within a factor of two of the optimal threshold.

While the computation threshold is somewhat well understood at this point, the issue of numerical stability has received much less attention. When operating over finite fields, proving the invertibility of an appropriate submatrix of the coding matrix suffices to guarantee correct decoding. However, in decoding a real system of equations, errors in the input can get amplified by the condition number (ratio of maximum and minimum singular values) of the associated matrix; hence, a low condition number is critical. For instance, in solving a square system of equations $\mathbf{y} = \mathbf{M}\mathbf{x}$, suppose that $\mathbf{y}$ is perturbed to $\tilde{\mathbf{y}}$ (owing to round-off errors) and that the estimate of $\mathbf{x}$ is $\hat{\mathbf{x}} := \mathbf{M^{-1}}\tilde{\mathbf{y}}$. Then, the normalized error in $\hat{\mathbf{x}}$ is given by

$$\frac{\|\hat{\mathbf{x}}-\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|\mathbf{M^{-1}}(\tilde{\mathbf{y}}-\mathbf{y})\|}{\|\mathbf{M^{-1}}\mathbf{y}\|} \leq \frac{\sigma_{\max}(\mathbf{M^{-1}})}{\sigma_{\min}(\mathbf{M^{-1}})}\frac{\|\tilde{\mathbf{y}}-\mathbf{y}\|}{\|\mathbf{y}\|} = \frac{\sigma_{\max}(\mathbf{M})}{\sigma_{\min}(\mathbf{M})}\frac{\|\tilde{\mathbf{y}}-\mathbf{y}\|}{\|\mathbf{y}\|} = \kappa(\mathbf{M})\frac{\|\tilde{\mathbf{y}}-\mathbf{y}\|}{\|\mathbf{y}\|},$$

where $\sigma_{\max}(\mathbf{M})$ and $\sigma_{\min}(\mathbf{M})$ denote the maximum and minimum singular values of $\mathbf{M}$ and their ratio $\kappa(\mathbf{M})$ is the condition number of the decoding matrix $\mathbf{M}$. Thus, it is clear that a small condition number of the decoding matrix leads to less amplification of the round-off error in $\hat{\mathbf{x}}$.

This issue is especially relevant since it is well recognized that polynomial interpolation over the reals suffers from significant numerical issues since the corresponding Vandermonde matrices have very high condition numbers (that are exponential in their size [18]). In fact, even for clusters with around $n = 30$ nodes, the condition number of the polynomial approach [5] is so large that the decoded result is essentially useless (see Section VI). We note here that Section VII of [4] remarks that the numerical issues can be handled by embedding all operations within a finite field. In Section VI, we demonstrate that the performance of this method is strongly dependent on the entries of matrices $\mathbf{A}$ and $\mathbf{B}$ and the resultant normalized MSE can be quite bad [19].

Some recent works have highlighted and considered the issue of numerical stability in this context. The work of [20], [21] presented strategies for distributed matrix-vector multiplication and demonstrated some schemes that empirically have better numerical performance than polynomial based schemes for some values of n and s. The work in [20] considers a convolutional coding approach, but from a parity check matrix perspective and the work in [21] uses universally decodable matrices which further allows to utilize the partial computations of the stragglers. However, both these approaches work only for the matrix-vector problem and do not provide a computable bound on the condition number of the decoding submatrices. The more recent work of [22] presents an approach that utilizes partial computations within the worker nodes for both matrix-vector and matrix-matrix multiplication; it also shows empirically that the schemes exhibit good numerical stability.

The work of [10] presents an alternate approach that works within the basis of orthogonal polynomials. They demonstrate that the worst case condition number of their schemes is at most $O(n^{2s})$ and their numerical experiments demonstrate improvements with respect to [5]. Our experimental evaluation in Section VI clearly demonstrates that our proposed schemes have condition numbers that are orders of magnitude lower than [10]. Reference [11] presents an approach where the encoded matrices are generated by taking random linear combinations of the block-columns of the respective matrices (this was also suggested in Remark 8 of [5]). We note here that their approach can be considered as a subclass of our methods, as discussed in Section VI. Table I shows a comparison of the features of several well-known approaches for distributed matrix computations. Our results in Section VI show that the underlying structure of our codes consistently results in lower worst case condition numbers than [11]. Finally, the parallel work of [12] presents an approach that leverages the properties of rotation matrices and circulant permutation matrices. They demonstrate that the worst case condition number of their recovery matrices grow at most as $O(n^{s+5.5})$. While their numerical results are better than ours, our work has the advantage of easy encoding and decoding and explores a convolutional approach to this problem which has not been considered before.

B. Summary of Contributions

In this paper we present an efficient and robust scheme for coded matrix computations that is inspired by convolutional codes. Our codes operate over the reals, unlike the majority of convolutional codes that are considered over finite fields [23]. Crucially, they exploit the Vandermonde property of the recovery matrices, where the matrices are defined over a different field (formal Laurent series over $\mathbb{R}$) than the real numbers. This naturally allows for simple encoding and decoding in addition to ensuring the threshold properties.

- Our work is among the first to provide an efficient coded computation approach for *both* matrix-vector and matrix-matrix multiplications that provably works in the (i) essentially noise-free regime where numerical precision issues dominate, and (ii) the noisy regime where noise is significant.
- We present two classes of codes in this work. Our first approach can be decoded using a peeling decoder with only add/subtract operations and has excellent numerical performance when the storage capacity of the nodes is slightly higher than the fundamental lower bound.

When operating very close to the storage capacity lower bound, we propose an alternative random convolutional coding strategy for which we can provide a "computable" upper bound (*cf.* Theorem 2 in Section V-A) on the worst case condition number of the recovery matrices. This naturally leads to a random sampling algorithm to pick a coding matrix with good performance. Our work draws novel connections with this problem and the asymptotic analysis of large Toeplitz matrices [24].

- An exhaustive comparison of our work with other approaches in the literature shows that the condition numbers of our work are orders of magnitude below all the comparable approaches (except [12]) and have fast decoding times. Fig. 4 depicts a comparison of the performance of the different schemes considered in our work.

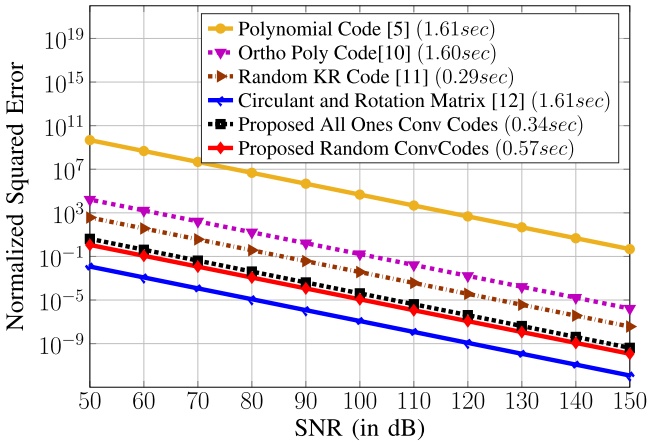

Fig. 4. Normalized MSE vs. SNR for different coded computation schemes for distributed matrix-matrix multiplication over $n = 18$ workers and $s = 3$ stragglers. The decoding time is reported for the different approaches in parentheses in the legend.

• As far as we are aware, most previous work has approached coded computation by exploiting its link with block codes under erasures. Our work is the first to investigate a convolutional coding approach to this problem. This in turn opens up newer problems for investigation in this area.

IV. Convolutional Coding for Distributed Matrix Computation

A. Simple Illustrative Example

We explain our key idea by means of the following example. Consider two row vectors in $\mathbb{R}^q$, $\mathbf{u}_0 = [u_{00}\ u_{01}\ \ldots\ u_{0(q-1)}]$ and $\mathbf{u}_1 = [u_{10}\ u_{11}\ \ldots\ u_{1(q-1)}]$. These vectors can also be represented as polynomials in the *indeterminate* D, $\mathbf{u}_i(D) = \sum_{j=0}^{q-1} u_{ij}D^j$ for $i = 0, 1$. As explained in Appendix A, these polynomials can be treated as elements in the ring of formal Laurent series in D [25]. Moreover, it can be shown that this ring is in fact a field, i.e., each element has a corresponding inverse. Consider the following encoding of $[\mathbf{u}_0(D)\ \ \mathbf{u}_1(D)]$.

$$\begin{aligned}&[\mathbf{c}_0(D)\ \ \mathbf{c}_1(D)\ \ \mathbf{c}_2(D)\ \ \mathbf{c}_3(D)]\\ &= [\mathbf{u}_0(D)\ \mathbf{u}_1(D)]\ \underbrace{\begin{bmatrix}1 & 0 & 1 & 1\\ 0 & 1 & 1 & D\end{bmatrix}}_{\mathbf{G}(D)}.\end{aligned}$$

It is not too hard to see that the polynomials $\mathbf{u}_0(D)$ and $\mathbf{u}_1(D)$ (equivalently the vectors $\mathbf{u}_0, \mathbf{u}_1$) can be recovered (or "decoded") from any two entries of the vector $[\mathbf{c}_0(D)\ \mathbf{c}_1(D)\ \mathbf{c}_2(D)\ \mathbf{c}_3(D)]$. For instance, suppose that we only receive $\mathbf{c}_2(D)$ and $\mathbf{c}_3(D)$. Notice that

$$\begin{aligned}\mathbf{c}_2(D) &= \sum_{j=0}^{q-1}(u_{0j} + u_{1j})D^j \quad \text{and}\\ \mathbf{c}_3(D) &= u_{00} + \sum_{j=0}^{q-2}(u_{0(j+1)} + u_{1j})D^{j+1} + u_{1(q-1)}D^q.\end{aligned}$$

Starting with u_{00} from the constant term of $\mathbf{c}_3(D)$, one can iteratively recover each of the coefficients of $\mathbf{u}_0(D)$ and $\mathbf{u}_1(D)$, with only one new variable to recover in each iteration. A similar argument applies if we consider a different set of two entries from $[\mathbf{c}_0(D)\ \mathbf{c}_1(D)\ \mathbf{c}_2(D)\ \mathbf{c}_3(D)]$. We refer to such a decoding scheme as a "peeling decoder".

Observe that the encoded polynomial $\mathbf{c}_3(D)$ has degree q, while the others have degree $q-1$. Thus, if the coefficients of the polynomials $\mathbf{c}_i$ correspond to encoded data that were sent to node i for processing, then node 3 would need slightly higher storage/processing capacity than nodes 0, 1, 2. Secondly, observe that the above idea can also be equivalently understood by replacing the 2×4 matrix of polynomials $\mathbf{G}(D)$ by a larger matrix of size $2q \times (4q+1)$ and rewriting all the scalar polynomials as row vectors. Let $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$ be row vectors of length q and $\mathbf{c}_3$ be a row vector of length $q+1$. Then,

$$\begin{bmatrix}\mathbf{c}_0 & \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3\end{bmatrix} = \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1\end{bmatrix}\begin{bmatrix}\mathbf{I}_q & \mathbf{0}_{q\times q} & \mathbf{I}_q & [\mathbf{I}_q\ \mathbf{0}]\\ \mathbf{0}_{q\times q} & \mathbf{I}_q & \mathbf{I}_q & [\mathbf{0}\ \mathbf{I}_q]\end{bmatrix}$$

where $\mathbf{0}_{q\times q}$ is a $q \times q$ matrix of zeroes, $\mathbf{I}_q$ is a $q \times q$ identity matrix, and $\mathbf{0}$ is a column of zeroes. In what follows, we consider generalizations of this basic example where the $\mathbf{u}_i$'s will correspond to block-columns of $\mathbf{A}$ and $\mathbf{B}$.

B. Proposed Matrix-Vector Multiplication Scheme

The above idea can naturally be adapted to the distributed matrix-vector multiplication setting. We show an example in Fig. 5 with $n = 4$ workers and $s = 2$ stragglers, so $k = n-s = 2$. Suppose that matrix $\mathbf{A}$ is partitioned into kq block-columns (the choice of q will be discussed shortly). In our work, the presentation follows more naturally if we index the block-columns of $\mathbf{A}$ using two indices instead of one. In particular, they are indexed as $\mathbf{A}_{\langle i,j\rangle}, i \in [k], j \in [q]$ (where $[\ell]$ denotes the set $\{0, \ldots, \ell-1\}$) and each worker node stores at most γr columns of length-t (γ is called the storage fraction).

Let $\mathbf{U}_i(D) = \sum_{j=0}^{q-1} \mathbf{A}^T_{\langle i,j\rangle} D^j$ for $0 \le i \le k-1$. Furthermore, let $\mathbf{Y}_{k,s}$ denote a $k \times s$ matrix whose (i,j)-th submatrix is $(\mathbf{Y}_{k,s})_{i,j} = (D^j)^i$, for $i \in [k], j \in [s]$, i.e., $\mathbf{Y}_{k,s}$ has the Vandermonde structure. We define

$$\mathbf{G}_{mv}(D) = \left[\underbrace{\mathbf{I}_k}_{\text{message part}} \;\middle|\; \underbrace{\mathbf{Y}_{k,s}(D)}_{\text{parity part}}\right]. \tag{1}$$

Consider the encoding

$$\begin{aligned}&[\mathbf{C}_0(D) \quad \mathbf{C}_1(D) \quad \ldots \quad \mathbf{C}_{n-1}(D)]\\ &= [\mathbf{U}_0(D) \quad \mathbf{U}_1(D) \quad \ldots \quad \mathbf{U}_{k-1}(D)]\ \mathbf{G}_{mv}(D).\end{aligned}$$

To arrive at the distributed matrix-vector multiplication scheme, we simply interpret the coefficients of the powers of D in $\mathbf{C}_i(D)$ as the encoded submatrices assigned to worker i (see Fig. 5 for an example). With this assignment, worker i computes the inner product of its assigned matrices and $\mathbf{x}$. We say that a $k \times n$ matrix is maximum-distance-separable (MDS) if any of its $k \times k$ submatrices is nonsingular. This property further implies that $\mathbf{A}^T\mathbf{x}$ can be recovered as long as

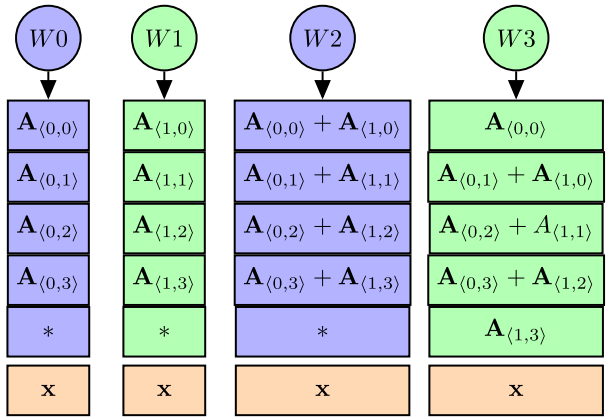

Fig. 5. Matrix-vector case with $n = 4$ workers and $s = 2$ stragglers, with $\gamma = \frac{5}{8}$.

any k workers complete their tasks. The following result shows that $\mathbf{G}_{mv}(D)$ is MDS; the proof appears in the Appendix.

Corollary 1 (Corollary of Upcoming Theorem 1 Given in Section IV-C): Any $k \times k$ submatrix of $\mathbf{G}_{mv}(D)$ has a determinant which is a non-zero polynomial in D, i.e., it is non-singular.

Analogous to convolutional coding, we call the first k workers the message workers and the last s workers the parity workers. Each of the first k message workers receives q submatrices $\mathbf{A}_{\langle i,j\rangle}, j = 0, 1, \ldots, q-1$, each of which is a matrix of size $t \times r/(kq)$. The rest of the s parity workers will receive $\geq q$ such submatrices. The highest exponent of D in the generator matrix $\mathbf{G}_{mv}(D)$ is $(s-1)(k-1)$. Thus, the maximum storage needed by a worker is $q+(s-1)(k-1)$ submatrices which indicates that the parity workers need to store a little more than $\frac{1}{k}$ fraction of matrix $\mathbf{A}$. This can be brought as close to $\frac{1}{k}$ as desired by choosing q large enough, hence this imbalance of the number of assigned submatrices is not significant. If we assume a bound of γ on the storage capacity fraction of any worker, we need

$$\begin{aligned} &\left(q + (s-1)(k-1)\right)\frac{r}{kq} \;\leq\; \gamma r, \\ &\implies q \geq \frac{(s-1)(k-1)}{k(\gamma - \frac{1}{k})}. \end{aligned} \tag{2}$$

For example, in Fig. 5, γ is set to $\frac{5}{8}$ which leads to $q = 4$.

C. Proposed Matrix-Matrix Multiplication Scheme

The matrix-matrix multiplication case requires the generalization of the above ideas. Let $\bar{a} = [a_0\ a_1\ \ldots\ a_{s-1}]$ and $\bar{b} = [b_0\ b_1\ \ldots\ b_{k-1}]$ be vectors of non-negative integers such that $0 \leq a_0 < a_1 < \cdots < a_{s-1}$ and $0 \leq b_0 < b_1 < \cdots < b_{k-1}$. Let $\mathbf{Y}_{\bar{b},\bar{a}}(D)$ denote a $k \times s$ matrix whose (i,j)-th entry is given by

$$[\mathbf{Y}_{\bar{b},\bar{a}}(D)]_{i,j} = (D^{a_j})^{b_i}. \tag{3}$$

Using this matrix, define a generalization of $\mathbf{G}_{mv}(D)$ as follows

$$\mathbf{G}(D) \;=\; \left[\, \mathbf{I}_k \;\;\middle|\;\; \mathbf{Y}_{\bar{b},\bar{a}}(D) \right]. \tag{4}$$

Observe that we obtain $\mathbf{G}_{mv}(D)$ by setting $a_j = j, 0 \leq j \leq s-1$ and $b_i = i, 0 \leq i \leq k-1$, which corresponds to $\mathbf{Y}_{k,s}(D)$. We will design an encoding scheme for matrix-matrix multiplication whose equivalent generator matrix is of the form in (4). Before we explain the design, we show that this matrix also satisfies the MDS property (the proof appears in the Appendix).

Theorem 1: Any $k \times k$ submatrix of the generator matrix $\mathbf{G}(D)$ defined in (4) is non-singular.

While non-singularity by itself does not reveal information about the corresponding condition numbers, Theorem 1 provides a class of schemes with a specific structure that have excellent numerical stability (see Fig. 4 "All-ones" curve) and can be modified and analyzed for condition number using the techniques discussed in Theorem 2 within Section V. The structure of $\mathbf{G}(D)$ in (4) also allows for an efficient peeling decoder.

In the matrix-matrix case, we design generator matrices $\mathbf{G}_A(D)$ of size $k_A \times n$ and $\mathbf{G}_B(D)$ of size $k_B \times n$ such that $s = n - k_A k_B$. Each worker stores fractions γ_A and γ_B of matrices $\mathbf{A}$ and $\mathbf{B}$ respectively. Let z be a large enough positive integer and let

$$\mathbf{U}_i^A(D) = \sum_{j=0}^{q_A-1} \mathbf{A}_{\langle i,j\rangle}^T D^{zj}, i \in [k_A], \text{ and} \tag{5}$$

$$\mathbf{U}_i^B(D) = \sum_{j=0}^{q_B-1} \mathbf{B}_{\langle i,j\rangle} D^{j}, i \in [k_B]. \tag{6}$$

Furthermore, we let $\mathbf{U}^A(D) = [\mathbf{U}_0^A(D)\ \ldots\ \mathbf{U}_{k_A-1}^A(D)]$ and $\mathbf{U}^B(D) = [\mathbf{U}_0^B(D)\ \ldots\ \mathbf{U}_{k_B-1}^B(D)]$. The final goal of the central node is to recover all products of the form $\mathbf{A}_{\langle i_1,j_1\rangle}^T \mathbf{B}_{\langle i_2,j_2\rangle}$ for $i_1 \in [k_A], j_1 \in [q_A], i_2 \in [k_B], j_2 \in [q_B]$. Once again by forming

$$\begin{aligned} &[\mathbf{C}_0^A(D)\ \ \mathbf{C}_1^A(D)\ \ldots\ \mathbf{C}_{n-1}^A(D)] = \mathbf{U}^A(D)\mathbf{G}_A(D), \text{ and} \\ &[\mathbf{C}_0^B(D)\ \ \mathbf{C}_1^B(D)\ \ldots\ \mathbf{C}_{n-1}^B(D)] = \mathbf{U}^B(D)\mathbf{G}_B(D), \end{aligned}$$

we can represent the assignment of coded submatrices of $\mathbf{A}$ and $\mathbf{B}$ to worker node i by the coefficients of $\mathbf{C}_i^A(D)$ and $\mathbf{C}_i^B(D)$ respectively. Following this step, each worker node computes the pairwise product of each coded submatrix of $\mathbf{A}$ and coded submatrix of $\mathbf{B}$ assigned to it.

The matrices $\mathbf{G}_A(D)$ and $\mathbf{G}_B(D)$ will be picked in such a way so that the pairwise product of each coefficient of $\mathbf{C}_i^A(D)$ and each coefficient of $\mathbf{C}_i^B(D)$ appears in $\mathbf{C}_i^A(D) \times \mathbf{C}_i^B(D)$, i.e., each worker node equivalently computes $\mathbf{C}_i^A(D) \times \mathbf{C}_i^B(D)$. Using MATLAB notation and Kronecker product properties, for $i = 1, 2, \ldots, n$, we have

$$\begin{aligned} &\mathbf{C}_i^A(D) \times \mathbf{C}_i^B(D) \\ &= \left[\mathbf{U}^A(D)\mathbf{G}_A(D)(:,i)\right] \times \left[\mathbf{U}^B(D)\mathbf{G}_B(D)(:,i)\right] \\ &= \left[\mathbf{U}^A(D) \otimes \mathbf{U}^B(D)\right] \times \left[\mathbf{G}_A(D)(:,i) \otimes \mathbf{G}_B(D)(:,i)\right], \end{aligned}$$

where $\otimes$ denotes the Kronecker product. Therefore, the computation peformed by the worker nodes can be compactly

represented using the Khatri-Rao product [26] (denoted by $\odot$)[2] Moreover, using the properties of the Khatri-Rao product, we have

$$\begin{aligned}&\left[\mathbf{U}^A(D)\mathbf{G}_A(D)\right] \odot \left[\mathbf{U}^B(D)\mathbf{G}_B(D)\right]\\ &= \left[\mathbf{U}^A(D) \otimes \mathbf{U}^B(D)\right]\ \left[\mathbf{G}_A(D) \odot \mathbf{G}_B(D)\right]. \qquad (7)\end{aligned}$$

The key idea at this point is to ensure that $\mathbf{G}_A(D)\odot\mathbf{G}_B(D)$ has the structure of a matrix as in (4). Towards this end, we choose

$$\mathbf{G}_A(D) = \begin{bmatrix} \overbrace{\mathbf{1}_{k_B} & \mathbf{0} & \dots & \mathbf{0}}^{k_A} \\ \mathbf{0} & \mathbf{1}_{k_B} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{1}_{k_B} \end{bmatrix} \mathbf{Y}_{k_A,s}(D^z),$$

$$\mathbf{G}_B(D) = \left[\overbrace{\mathbf{I}_{k_B} \quad \mathbf{I}_{k_B} \quad \dots \quad \mathbf{I}_{k_B}}^{k_A} \quad \mathbf{Y}_{k_B,s}(D) \right],$$

where $\mathbf{1}_{k_B}$ is an all-ones row vector of length k_B, and the total number of rows in $\mathbf{G}_A(D)$ and $\mathbf{G}_B(D)$ are k_A and k_B respectively. This implies that

$$\mathbf{G}_A(D) \odot \mathbf{G}_B(D) = [\mathbf{I}_k \ \mid \ \mathbf{Y}_{k_A,s}(D^z) \odot \mathbf{Y}_{k_B,s}(D)] \quad (8)$$

where $k = k_A k_B$. The following lemma shows that the RHS of (8) has the structure of the matrix in (4).

Lemma 1: The Khatri-Rao product $\mathbf{Y}_{k_A,s}(D^z)\odot\mathbf{Y}_{k_B,s}(D)$ is a matrix in the form of (3).

Proof: Note that the Kronecker product of ℓ-th column of $\mathbf{Y}_{k_A,s}(D^z)$ and ℓ-th column of $\mathbf{Y}_{k_B,s}(D)$ can be expressed as

$$\begin{bmatrix} 1 \\ D^{zl} \\ D^{2zl} \\ \vdots \\ D^{(k_A-1)zl} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ D^{l} \\ D^{2l} \\ \vdots \\ D^{(k_B-1)l} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ D^{(k_B-1)l} \\ D^{zl} \\ \vdots \\ D^{(k_B-1+z)l} \\ \vdots \\ D^{(k_A-1)zl} \\ \vdots \\ D^{(k_B-1+(k_A-1)z)l} \end{bmatrix}. \quad (9)$$

The vector on the RHS above consists of powers of D^l and can be seen to be in the form of (3). ∎

Lemma 1 explains why Theorem 1 is applicable to the coding scheme used for matrix-matrix multiplication. Thus, this lemma, along with Theorem 1 implies that the proposed convolutional code based matrix-matrix multiplication scheme is MDS.

Now, we need to choose such a value of z which ensures that $\left[\mathbf{U}^A(D) \otimes \mathbf{U}^B(D)\right]$ in (7) contains all the distinct pairwise products that we are interested. We know that worker i will be assigned the jobs according to the column i of the RHS in (8). Now by examining the structure of the RHS in (8), it can be verified that for $i = 0, 1, 2, \dots, k-1$, worker i will be assigned q_A submatrices from $\mathbf{A}$ and q_B submatrices from $\mathbf{B}$. And for $i = k, k+1, k+2, \dots, n-1$, any worker i will be assigned $q_A + (i-k) \times (k_A - 1)$ submatrices from $\mathbf{A}$ and $q_B + (i-k) \times (k_B - 1)$ submatrices from $\mathbf{B}$. Thus the maximum number of submatrices will be assigned to worker $n-1$, which will have $q_A + (s-1) \times (k_A - 1)$ submatrices from $\mathbf{A}$ and $q_B + (s-1) \times (k_B - 1)$ submatrices from $\mathbf{B}$, since $s = n - k$. For the assignment of this worker,

$$\begin{aligned} C^{\mathbf{A}}_{n-1}(D) &= \mathbf{U}^A_0(D) + \mathbf{U}^A_1(D)\ D^{z(s-1)} + \mathbf{U}^A_2(D)\ D^{2\ z(s-1)} \\ &\quad + \dots + \mathbf{U}^A_{k_A-1}(D)\ D^{z(k_A-1)(s-1)}\ ; \text{and} \\ C^{\mathbf{B}}_{n-1}(D) &= \mathbf{U}^B_0(D) + \mathbf{U}^B_1(D)\ D^{s-1} + \mathbf{U}^B_2(D)\ D^{2(s-1)} \\ &\quad + \dots + \mathbf{U}^B_{k_B-1}(D)\ D^{(k_B-1)(s-1)}\ . \end{aligned}$$

It can be verified that $\mathbf{C}^{\mathbf{A}}_{n-1}(D)$ is a polynomial in D where the exponent of D at any term is an integer multiple of z. Since each $\mathbf{U}^B_i(D)$ has a degree $q_B - 1$, the degree of $C^{\mathbf{B}}_{n-1}(D)$ is $q_B - 1 + (s-1)(k_B - 1)$, and thus we conclude that

$$z \ \geq \ q_B + (s-1)(k_B - 1). \quad (10)$$

It should be noted that this value of z is large enough for (9) to hold.

Next, using an approach similar to (2), we can derive

$$q_A \geq \frac{(s-1)(k_A-1)}{k_A(\gamma_A - \frac{1}{k_A})} \quad \text{and} \quad q_B \geq \frac{(s-1)(k_B-1)}{k_B(\gamma_B - \frac{1}{k_B})}.$$

Example 1: Consider the computation of $\mathbf{A}^T\mathbf{B}$ over $n = 6$ workers and $s = 2$ stragglers. Assume that each worker can store/process $\gamma_A = 5/8$ fraction of matrix $\mathbf{A}$ and $\gamma_B = 2/3$ fraction of matrix $\mathbf{B}$. We set $k_A = k_B = 2$, so that $q_A = 4$ and $q_B = 3$. By setting $z = q_B + (s-1)(k_B - 1) = 4$, we obtain

$$\mathbf{U}^A_i(D) = \sum_{j=0}^{3} \mathbf{A}^T_{\langle i,j\rangle} D^{4j}, \text{ for } i = 0, 1;$$

$$\text{and } \ \mathbf{U}^B_i(D) = \sum_{j=0}^{2} \mathbf{B}_{\langle i,j\rangle} D^{j}, \text{ for } i = 0, 1.$$

Furthermore,

$$\mathbf{G}_A(D) = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & D^4 \end{bmatrix} \text{ and}$$

$$\mathbf{G}_B(D) = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & D \end{bmatrix}.$$

The assignment of jobs to all the workers can be obtained from $[\mathbf{U}^A_0(D)\ \mathbf{U}^A_1(D)]\,\mathbf{G}_A(D)$ and $[\mathbf{U}^B_0(D)\ \mathbf{U}^B_1(D)]\,\mathbf{G}_B(D)$. This is shown in Fig. 6.

Remark 1: Our proposed encoding process is very simple and involves only additions at the central node.

D. Decoding Algorithm: Peeling Decoder

In Section IV-A we demonstrated by an example that in the matrix-vector case, our scheme can be decoded by a peeling decoder. The situation is similar in the matrix-matrix case. In particular, the underlying structure of the convolutional code

[2]For two matrices with the same column dimension, the Khatri-Rao product corresponds to the matrix obtained by taking the Kronecker product of the corresponding columns.

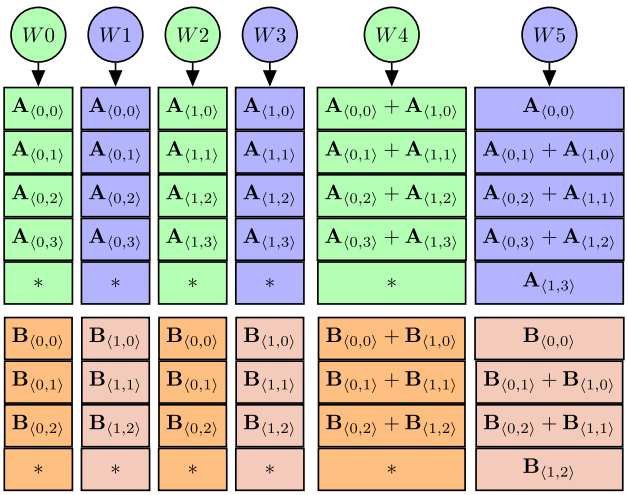

Fig. 6. Matrix-matrix multiplication with $n = 6$ workers and $s = 2$ stragglers with $\gamma_A = \frac{5}{8}$ and $\gamma_B = \frac{2}{3}$.

allows for a very simple peeling decoder whereby, at each step, the algorithm is guaranteed to find an equation with only one unknown.

Example 2: In Fig. 6 (corresponding to Example 1) suppose that workers $W0$ and $W1$ are stragglers. The goal of the central node is to recover all products of the form $\mathbf{A}^T_{\langle i_1,j_1\rangle}\mathbf{B}_{\langle i_2,j_2\rangle}$ for $i_1 \in [2], j_1 \in [4], i_2 \in [2], j_2 \in [3]$, hence we have total $2 \times 4 \times 2 \times 3 = 48$ unknowns that need to be recovered. Note that we can directly obtain $4 \times 6 = 24$ unknowns from workers $W2$ and $W3$ which correspond to the unknowns of the form $\mathbf{A}^T_{\langle 1,j_1\rangle}\mathbf{B}_{\langle i_2,j_2\rangle}$. This is equivalent to having recovered $\mathbf{U}^A_1(D)\mathbf{U}^B_0(D)$ and $\mathbf{U}^A_1(D)\mathbf{U}^B_1(D)$.

So it remains to recover all unknowns of the form $\mathbf{A}^T_{\langle 0,j_1\rangle}\mathbf{B}_{\langle i_2,j_2\rangle}$ for $j_1 \in [4], i_2 \in [2], j_2 \in [3]$ or equivalently $\mathbf{U}^A_0(D)\mathbf{U}^B_0(D)$ and $\mathbf{U}^A_0(D)\mathbf{U}^B_1(D)$ from the results obtained from workers $W4$ and $W5$.

First, we concentrate on the first block product obtained from $W5$, which helps to recover $\mathbf{A}^T_{\langle 0,0\rangle}\mathbf{B}_{\langle 0,0\rangle}$ directly. Following this we examine the first block product from $W4$, which is $\left(\mathbf{A}_{\langle 0,0\rangle} + \mathbf{A}_{\langle 1,0\rangle}\right)^T \left(\mathbf{B}_{\langle 0,0\rangle} + \mathbf{B}_{\langle 1,0\rangle}\right)$; the only unknown here, that is yet to be recovered, is $\mathbf{A}^T_{\langle 0,0\rangle}\mathbf{B}_{\langle 1,0\rangle}$ which can therefore be decoded. We can use this to decode $\mathbf{A}^T_{\langle 0,0\rangle}\mathbf{B}_{\langle 0,1\rangle}$ from the next product in $W5$. Thus, we can keep moving back and forth between $W4$ and $W5$ and it can be verified that we can recover all the block products $\mathbf{A}^T_{\langle 0,j_1\rangle}\mathbf{B}_{\langle i_2,j_2\rangle}$ in a similar fashion. Another way to understand the decoding process is to consider the unknown polynomials $\mathbf{U}^A_0(D)\mathbf{U}^B_0(D)$ and $\mathbf{U}^A_0(D)\mathbf{U}^B_1(D)$ and realize that $W4$ and $W5$ give us (i) $\mathbf{U}^A_0(D)(\mathbf{U}^B_0(D) + \mathbf{U}^B_1(D))$ and (ii) $\mathbf{U}^A_0(D)(\mathbf{U}^B_0(D) + D\mathbf{U}^B_1(D))$. The unknowns can thus be decoded analogous to the illustrative example in the matrix-vector case in Section IV-A.

Crucially, this scheme can be encoded and decoded purely with only add/subtract operations and can thus be highly optimized. This algorithm is very fast and has excellent numerical stability (*cf.* Fig. 4) in experiments.

Decoding Complexity: Suppose that we obtain results from workers in $\mathcal{I} \subset \{0, 1, \ldots, n-1\}$, with $|\mathcal{I}| \geq k$. In the matrix-vector case our unknowns are $\mathbf{u}_{il} = \mathbf{A}^T_{\langle i,l\rangle}\mathbf{x}, i \in [k], l \in [q]$; each of these is a vector of length $r/(kq)$. Let row-vector $\mathbf{z}_j$ denote the collection of the j-th entries of each of these unknowns, where $j \in [r/(kq)]$. Let the output of the worker nodes corresponding to $\mathbf{z}_j$ be denoted by $\mathbf{y}_j$. The length of $\mathbf{y}_j$ depends on $\mathcal{I}$.

We assume that the central node obtains results from a subset of the message workers, $\mathcal{I}_1 \subset \{0, 1, \ldots, k-1\}$, so that $|\mathcal{I}_1| \leq k$. This implies that it can recover $|\mathcal{I}_1|q$ unknowns directly. Moreover, it obtains results from the parity workers indexed by $\mathcal{I}_2 \subset \{k, k+1, \ldots, n-1\}$, where $|\mathcal{I}_2| = k - |\mathcal{I}_1|$. Thus, it needs to recover the remaining $kq - |\mathcal{I}_1|q$ unknowns.

We consider the worst case where $|\mathcal{I}_2| = s$. According to the design of this scheme, each of the kq unknowns appears once in every parity worker, and thus the system of equations has at most kqs non-zero entries. Furthermore, in a peeling decoder one variable can be decoded and substituted in the remaining equations at each iteration. Therefore, the time complexity of solving this sparse system is $O(kqs)$. As we solve a total of $r/(kq)$ such systems of equations, the total time taken is $O(rs)$ which is *independent* of q and thus does not grow with it; similarly it can be shown that for the matrix-matrix case the time is $O(rws)$.

It should be noted that the matrices $\mathbf{A}$ and $\mathbf{B}$ are of sizes $t \times r$ and $t \times w$ respectively, thus the computational complexity of computing $\mathbf{A}^T\mathbf{B}$ is $O(rwt)$. In a distributed system, this job is distributed over n workers with s stragglers, so, on average, the computational complexity of each of the workers is $O\left(\frac{rwt}{k}\right)$, where $k = n-s$. On the other hand, to get the final result, we need to recover rw unknowns, which is the size of $\mathbf{A}^T\mathbf{B}$. Thus the decoding complexity does not depend on the parameter t which indicates that the decoding time can be often considered negligible in comparison to the worker computation time when t is very large [16]. Nevertheless, fast decoding is a desirable feature of any coded computation scheme.

E. Effect of q: Storage Fraction, Imbalance in Task Assignment

Our presented scheme thus far is provably MDS, efficiently decodable and has excellent numerical stability in experiments. Note that our schemes require lower bounds on the value of q which have an inverse dependence on $\gamma - 1/k$. Thus, if one wants to reduce the imbalance between the task assignments to the message nodes and the parity nodes, then q needs to be chosen large enough. It turns out that for large values of q, the worst case condition number of our scheme can be very large. We present a theoretical treatment of this phenomenon in the upcoming Section V and discuss techniques for mitigating this effect.

V. Numerical Stability Analysis

To understand numerical stability, we first introduce a modified encoding scheme and then discuss the matrix representation of the coding ideas described above.

Definition 1 (Randomly Scaled Generator Matrix): Let $\mathbf{R}$ be a $k \times s$ matrix of real numbers. Consider the generator

matrix $\mathbf{G}(D)$ defined in (4). Replace $\mathbf{Y}_{\bar{b},\bar{a}}(D)$ by $\mathbf{R} \circ \mathbf{Y}_{\bar{b},\bar{a}}(D)$. Here, $\circ$ denotes Hadamard product (.* operation in MATLAB).

Note that if we set $r_{ij} = 1$ for all entries of the matrix $\mathbf{R}$, we recover the old generator matrix $\mathbf{G}(D)$ (the "All-ones" case).

1) Understanding the Matrix Representation: It is not hard to see that the matrix representation of the transformation induced by the $k \times n$ generator polynomial matrix $\mathbf{G}(D)$ from Definition 1 can be understood as right multiplying a kq-length row vector of input data by the following matrix. An example of this was given in Section IV-A.

Definition 2 ($\tilde{\mathbf{G}}$: Matrix Representation of $\mathbf{G}(D)$): We first define a $q \times (q+h)$ shift matrix that takes a q-length row vector and returns a $q+h$-length row vector, where the original vector is shifted to the right by j components. This is the matrix $\tilde{\mathbf{D}}^{h;j} \triangleq \begin{bmatrix} \mathbf{0}_{q\times j} & \mathbf{I}_q & \mathbf{0}_{q\times(h-j)} \end{bmatrix}$. The (i,ℓ)-th block matrix of $\tilde{\mathbf{G}}$ for $\ell = 0, 1, \ldots, k-1$ and $i = 0, 1, \ldots, k-1$ is

$$(\tilde{\mathbf{G}})_{i,\ell} = \begin{cases} \mathbf{I}_q & \text{if } \ i = \ell \\ \mathbf{0}_{\mathbf{q}\times\mathbf{q}} & \text{if } \ i \neq \ell \end{cases}$$

and for $\ell = k + j$, $j = 0, 1, \ldots (s-1)$,

$$(\tilde{\mathbf{G}})_{i,\ell} = r_{ij} \tilde{\mathbf{D}}^{a_j b_{k-1}; a_j b_i}.$$

Thus, $\tilde{\mathbf{G}}$ is a $kq \times (nq + \delta)$ matrix where

$$\delta \ = \ b_{k-1} \sum_{j=0}^{s-1} a_j. \tag{11}$$

With the above definition, decoding can be understood as inverting the specific $k \times k$ block submatrix of $\tilde{\mathbf{G}}$, denoted $\tilde{\mathbf{G}}_{\mathcal{I}}$ where $\mathcal{I}$ is the set of indices of the k workers that have returned their jobs.

2) Quantifying Round-Off Error Amplification: When assuming perfectly noise-free computations, invertibility of the decoding matrix, $\tilde{\mathbf{G}}_{\mathcal{I}}$, is sufficient to guarantee perfect recovery/decoding of the desired matrix-matrix product. However, since all computing devices are finite precision, matrix multiplications will frequently result in bit overflow/underflow and hence round-off errors. As explained earlier (*cf.* Section III-A), the decoding process amplifies the round-off error by a factor that can at most be as large as the condition number of the decoding matrix. Thus, the numerical stability of our scheme is quantified by the largest condition number over all block submatrices $\tilde{\mathbf{G}}_{\mathcal{I}}$, i.e., by

$$\kappa_{worst} \triangleq \max_{\mathcal{I}\subset[n], |\mathcal{I}|=k} \kappa(\tilde{\mathbf{G}}_{\mathcal{I}}).$$

A. Upper Bounding κ_{worst}

Observe that the matrix $\tilde{\mathbf{G}}$, and consequently the decoding submatrix $\tilde{\mathbf{G}}_{\mathcal{I}}$ with $|\mathcal{I}| = k$, has a very specific structure. Because of this, it is possible to show that the matrix $\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^T$ is a $k \times k$ block matrix with Toeplitz blocks of size $q \times q$ (see Appendix B). This fact is useful since the asymptotics of $\lambda_{\max}(\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^T)$ and $\lambda_{\min}(\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^T)$ when q is large have been studied in [27]. In particular, Theorem 3 of [27] shows that using Fourier transform ideas, one can bound the eigenvalues of such matrices by computing the minimum (and maximum) of the smallest (and largest) eigenvalues of a much smaller $k \times k$ matrix that is a function of a scalar parameter ω which lies in $[-\pi, \pi]$.

With some abuse of notation, let $\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})$ represent the matrix obtained by extracting $\mathbf{G}_{\mathcal{I}}(D)$ (from $\mathbf{G}(D)$ in (4)) and then substituting $D = e^{\mathrm{i}\omega}$ (where $\mathrm{i} = \sqrt{-1}$). By adapting the results of [27] (see Appendix B for a detailed description), we have the following theorem.

Theorem 2: For $\mathcal{I} \subset \{0, \ldots, n-1\}$ such that $|\mathcal{I}| = k$, we have

$$\lim_{q\to\infty} \lambda_{\min}(\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^*) = \min_{\omega\in[-\pi,\pi]} \lambda_{\min}[(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))^*];$$
$$\lim_{q\to\infty} \lambda_{\max}(\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^*) = \max_{\omega\in[-\pi,\pi]} \lambda_{\max}[(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))^*].$$

Moreover, for any q

$$\lambda_{\max}(\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^*) \leq \max_{\omega\in[-\pi,\pi]} \lambda_{\max}[(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))^*];$$
$$\text{and} \quad \lambda_{\min}(\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^*) \geq \min_{\omega\in[-\pi,\pi]} \lambda_{\min}[(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))^*].$$

Theorem 2 shows that we can find an upper bound on the condition number of $\tilde{\mathbf{G}}_{\mathcal{I}}$ based on a scalar optimization over $\omega \in [-\pi, \pi]$. When $\mathbf{R}$ is chosen to be the all-ones matrix, the characterization of Theorem 2 allows us to conclude that when $s > 1$, there exist choices of $\mathcal{I} \subseteq \{0, 1, \ldots, n-1\}, |\mathcal{I}| = k$ such that $\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^*$ has a minimum eigenvalue that will go to zero as $q \to \infty$. In particular, the corresponding $\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})$ has repeated columns for $\omega = 0$.

Example 3: Consider the $(n, k) = (4, 2)$ example with $G(D) = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & D \end{bmatrix}$. Suppose that $\mathcal{I} = \{2, 3\}$. This implies that

$$\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^T = \tilde{\mathbf{G}}_2\tilde{\mathbf{G}}_2^T + \tilde{\mathbf{G}}_3\tilde{\mathbf{G}}_3^T = \begin{bmatrix} 2\mathbf{I}_q & \mathbf{I}_q + \mathbf{L} \\ \mathbf{I}_q + \mathbf{U} & 2\mathbf{I}_q \end{bmatrix},$$

where $\mathbf{U}$ and $\mathbf{L}$ are $q \times q$ upper shift and lower shift matrices respectively (see, e.g., (17) in the Appendix).

The corresponding $\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})^*$ can be obtained as

$$\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})^* = \begin{bmatrix} 2 & 1 + e^{-\mathrm{i}\omega} \\ 1 + e^{\mathrm{i}\omega} & 2 \end{bmatrix}.$$

Using Theorem 2, we can conclude therefore that $\lim_{q\to\infty} \lambda_{max}[\mathcal{T}] = 2$ (achieved at $\omega = \pi$) and $\lim_{q\to\infty} \lambda_{min}[\mathcal{T}] = 0$ (achieved at $\omega = 0$). This implies therefore that as q becomes larger and larger, the matrix $\tilde{\mathbf{G}}_{\mathcal{I}}$ becomes more and more ill-conditioned, though it is nonsingular for any fixed q.

Therefore, considering a nontrivial scaling of the parity part with a matrix $\mathbf{R}$ is essential for well-conditioned behavior when q is very large.

B. Randomly-Weighted Convolutional Coding

We now show that choosing the matrix $\mathbf{R}$ randomly in Definition 1 results in better numerical stability than the All-ones scheme in the regime of large q but requires marginally higher decoding complexity.

The following result shows that the MDS property continues to holds with probability 1 when the entries are chosen

i.i.d. from a continuous distribution. The proof is an easy consequence of Theorem 1 and appears in the Appendix A.

Corollary 2: If the entries of the matrix $\mathbf{R}$ are chosen i.i.d. from any continuous-valued probability distribution, then, any $k \times k$ submatrix of the generator matrix mentioned in Definition 1 is non-singular with probability one.

We now demonstrate that choosing the matrix $\mathbf{R}$ randomly allows us to upper bound the worst case condition number (over the recovery matrices) even when $q \to \infty$. In the matrix-vector scenario, Theorem 2 suggests the following algorithm for choosing $\mathbf{R}$. We proceed by randomly choosing $\mathbf{R}$. Let $\mathcal{I} \subset \{0, \ldots, n-1\}, |\mathcal{I}| = k$ and let $\Omega = \{0, \pm\frac{\pi}{N}, \pm\frac{2\pi}{N}, \ldots, \pm\frac{(N-1)\pi}{N}, \pm\pi\}$ for a large positive integer N denote a fine enough grid of the interval $[-\pi, \pi]$. Let $\kappa_{\mathbf{R}}$ be defined as

$$\kappa_{\mathbf{R}} = \max_{\substack{\mathcal{I} \subset \{0,\ldots,n-1\}, \\ |\mathcal{I}|=k}} \sqrt{\left(\frac{\max\limits_{\omega \in \Omega} \lambda_{\max}[(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))^*]}{\min\limits_{\omega \in \Omega} \lambda_{\min}[(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))(\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega}))^*]} \right)}.$$

Thus, $\kappa_{\mathbf{R}}$ indicates the maximum condition number of $\mathbf{G}_{\mathcal{I}}(e^{\mathrm{i}\omega})$ overall $\binom{n}{k}$ choices of $\mathcal{I}$; this is an upper bound on the maximum condition number of $\tilde{\mathbf{G}}_{\mathcal{I}}$. The algorithm repeatedly generates choices of $\mathbf{R}$ and retains the choice that has the lowest value of $\kappa_{\mathbf{R}}$; this denoted by $\mathbf{R}^\star$. The matrix-matrix case is similar, except that we generate two random matrices denoted $\mathbf{R}_A$ and $\mathbf{R}_B$ and consider the worst case condition number of the appropriate submatrices of (8) to obtain $\mathbf{R}_A^\star$ and $\mathbf{R}_B^\star$. We emphasize that even though the search requires optimizing over $\binom{n}{k} = \binom{n}{s}$ choices of $\mathcal{I}$, this is a one-time cost for designing the coding scheme for a system with n worker nodes which is resilient to $s = n - k$ stragglers. Furthermore, (i) the search does not have any dependence on q, and (ii) the value of s is typically a small constant, that either does not grow or grows very slowly with n. Thus the complexity of the above design, n^s, grows as polynomial in n. Appendix C presents some numerical results on the amount of time taken to find a good $\mathbf{R}$ matrix.

For systems with $n = 12, s = 3$ and $n = 13, s = 3$, we conducted 50 random trials each to find the corresponding $\mathbf{R}^\star$ for the matrix vector multiplication case; the entries were sampled i.i.d. from the uniform distribution on $[-1, 1]$. Our algorithm also returns the asymptotic upper bound on $\kappa(\mathbf{R}^\star)$. By sweeping over values of q, we can also compute the actual worst-case condition number for each particular chosen value of q. Fig. 7 depicts the upper bound and the actual worst case condition numbers for different n and s.

C. Random Convolutional Coding: Decoding Algorithm

In principle, it is possible to use a fast peeling decoder for decoding as done earlier in the all-ones case. Note however that the peeling decoder solves a system of kq equations in kq variables. Thus, it only uses kq columns of the $\tilde{\mathbf{G}}_{\mathcal{I}}$ even though $\tilde{\mathbf{G}}_{\mathcal{I}}$ is a matrix of size $kq \times (kq + \delta')$ where δ' is an integer between zero and δ (*cf.* (11)), depending on which set of k worker nodes finished their computations (in matrix-vector multiplication).

In particular, the stability of the peeling decoder depends on the condition number of the relevant full rank square submatrix of $\tilde{\mathbf{G}}_{\mathcal{I}}$. In general, this condition number is higher than that of $\tilde{\mathbf{G}}_{\mathcal{I}}$. In our numerical experiments we have found that for the all-ones case, the worst case condition numbers of both matrices ($\tilde{\mathbf{G}}_{\mathcal{I}}$ and full rank square submatrix of $\tilde{\mathbf{G}}_{\mathcal{I}}$) are almost the same (see more experimental details in Section VI). This explains the numerically stable behavior of the peeling decoder in the all-ones case.

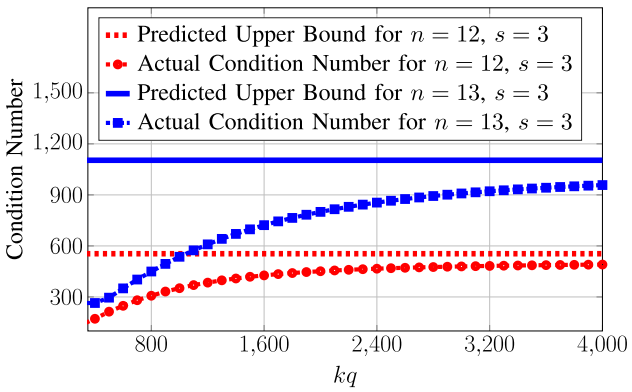

Fig. 7. Worst case condition number for the random convolutional code for different n and s.

The situation changes quite a bit when we consider random scaling of the generator matrix. e.g., when the entries of $\mathbf{R}$ are i.i.d. random Gaussian, the difference is very large. In this case, the condition number of the full rank square submatrix of $\tilde{\mathbf{G}}_{\mathcal{I}}$ can be very high for certain sets of workers $\mathcal{I}$ (see Section VI). But in all cases, κ_{worst} overall $\tilde{\mathbf{G}}_{\mathcal{I}}$ is significantly smaller than that of the all-ones case. Thus, it is clear that one should use all the columns of $\tilde{\mathbf{G}}_{\mathcal{I}}$ for decoding, rather than using only kq equations.

Decoding Complexity: Similar to the discussion in Section IV-D, we assume that the fastest k workers include the message worker set $\mathcal{I}_1$ and the parity worker set $\mathcal{I}_2$, so that $|\mathcal{I}_1| + |\mathcal{I}_2| = k$. We can decode some unknowns directly from the workers in $\mathcal{I}_1$, and in the worst case, we need to recover the other sq unknowns from the parity workers in $\mathcal{I}_2$. In this case, one can solve a least square (LS) problem to recover the sq unknowns. This LS problem can be solved in different ways. The most straightforward way would be matrix inversion ($O\left((sq)^3\right)$ time) followed by solving $\frac{rw}{kq}$ systems of equations ($O\left(\frac{rw}{kq}(sq)^2\right)$ time). If $sq \ll r, w$; we can write it as $O\left(\frac{rw}{k}s^2\, q\right)$. On the other hand if the value of q is large, then we can use techniques such as conjugate gradient descent to solve the LS problem. This is especially useful when q is large since the underlying system of equations is sparse. Thus, each iteration of conjugate gradient descent can be solved in a fast manner. In particular, if we run it for T iterations to recover these sq unknown blocks, the decoding complexity is $O\left(\frac{rw}{kq} \times sq \times s \times T\right) = O\left(\frac{rw}{k}s^2\, T\right)$. To reach within ϵ fraction of the solution, the number of iterations scales a $O(\kappa \log(1/\epsilon))$ where κ is the condition number of the linear system of equations.

Overall the decoding complexity of the random convolutional code setting is marginally higher than the all-ones case, depending on which algorithm is used for the LS solution.

TABLE II

COMPARISON OF WORST CASE CONDITION NUMBERS (κ_{worst}) AND AVERAGE CONDITION NUMBER κ_{avg} FOR MATRIX-MATRIX MULTIPLICATION FOR $n = 18$ AND $s = 3$

METHODS	κ_{worst}	κ_{avg}
POLYNOMIAL CODE [5]	4.031×10^7	5.67×10^6
ORTHO-POLY CODE [10]	2.506×10^4	245.25
RAND. KR PRODUCT CODE[11]	5329.3	116.27
CIRC. AND ROT. MATRIX [12]	102	14.72
PROP. ALL-ONES CONV. CODE	**4417.8**	**589.41**
PROP. RANDOM CONV. CODE	**1829.4**	**113.08**

VI. COMPARISONS AND NUMERICAL EXPERIMENTS

In this section, we discuss the results of the numerical experiments for our proposed approaches and compare our methods with other available methods.

The polynomial code approach [5] suffers from the problem that real Vandermonde matrices have condition numbers that are exponential in their size. This in turn implies that for large number of workers (for example, 30 workers) the condition number of the decoding matrix is so high that the recovered result by the central node is actually useless.

To avoid this numerical issue, Section VII of [4] remarks that the real computation can be embedded within a large enough finite field of prime order p. It turns out that the performance of this scheme is strongly dependent on the entries of $\mathbf{A}$ and $\mathbf{B}$ and the resultant normalized MSE can be quite bad. These arguments have appeared in [19]; we present an outline below.

We note that computations in this method are error-free only when each entry of the product matrix $\mathbf{A}^T\mathbf{B}$ is an integer in $\{0, 1, \ldots, p-1\}$. If this requirement is violated, the proposed mod-p computations can return catastrophically wrong answers [19]. This means that the matrices A and B need to be multiplied by a scalar and quantized so that each entry of the resulting matrix is an integer that is within the appropriate range. Suppose that the absolute values of the entries of $\mathbf{A}$ and $\mathbf{B}$ are upper bounded by α; then we need $\alpha^2\ t < p$. This is referred to as the dynamic range constraint in [19]. For instance, with 64-bit integers (the standard on present day computers), the largest integer is $\approx 10^{19}$. Thus, even if $t < 10^5$, the method can only support $\alpha \leq 10^7$. Thus, the range is rather limited.

The work of [19] constructs adversarial $\mathbf{A}$ and $\mathbf{B}$ integer matrices for this method as follows. Let $p = 2147483647$ (note that this is much larger than the publicly available code of [5] which uses $p = 65537$) so that their method can support higher dynamic range. Next let $r = w = t = 2000$. This implies that α needs to be ≤ 1000 by the dynamic range constraint. The matrices have the following block decomposition.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} \end{bmatrix}, \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} \\ \mathbf{B}_{1,0} & \mathbf{B}_{1,1} \end{bmatrix}.$$

Each $\mathbf{A}_{i,j}$ and $\mathbf{B}_{i,j}$ is a matrix of size 1000×1000, with entries chosen from the following distributions. $\mathbf{A}_{0,0}$, $\mathbf{A}_{0,1}$ distributed Unif$(0, \ldots, 9999)$ and $\mathbf{A}_{1,0}$, $\mathbf{A}_{1,1}$ distributed Unif$(0, \ldots, 9)$. Next, $\mathbf{B}_{0,0}$, $\mathbf{B}_{0,1}$ distributed Unif$(0, \ldots, 9)$ and $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ distributed Unif$(0, \ldots, 9999)$. In this scenario, the dynamic range constraint requires us to multiply each matrix by 0.1 and quantize each entry between 0 and 999. Note that this implies that $\mathbf{A}_{1,0}, \mathbf{A}_{1,1}, \mathbf{B}_{0,0}, \mathbf{B}_{0,1}$ are all quantized into zero submatrices since the entry in these four submatrices is less than 10. We emphasize that the finite field embedding technique *only* recovers the product of these quantized matrices. However, this product is the all-zeros matrix, i.e., the decoded matrix will also be the all-zeros matrix. Therefore, the normalized MSE in this case will be 100%. There are also significant computational issues as discussed in [19]. We note here that such adversarial matrices can be found even for larger choices of p. It is worth noting that the normalized MSE of the other methods do not depend on the actual values of $\mathbf{A}$ and $\mathbf{B}$.

The work of [10] uses orthogonal polynomials and Chebyshev-Vandermonde matrices for the encoding part, which significantly improves the condition number of the decoding matrices compared to [5] and [6]. The work in [11] uses random Khatri-Rao product where random coefficients are used for the encoding, which further improves the numerical stability. The recent preprint [12] uses circulant and permutation matrices to improve the numerical stability of the polynomial approach. We compare our approaches with these methods with exhaustive numerical experiments which are performed over a cluster in `AWS` (Amazon Web Services). A `t2.2xlarge` machine is used as the central node and `t2.small` machines are used as the worker nodes. Software code for recreating these experiments can be found at [28].

A. Comparing κ_{worst} and MSE for Matrix-Matrix Case

For a system with $n = 18$ workers and $s = 3$ stragglers for matrix-matrix multiplication, we set $\gamma_A = \frac{1}{4}$ and $\gamma_B = \frac{2}{5}$ with $k_A = 5$ and $k_B = 3$, so $k = k_A k_B = n - s = 15$. Table II reports a comparison of the worst-case condition numbers for different approaches in the literature. It can be observed that the work of [5] and [10] have much higher condition numbers than our proposed schemes (All-ones and Random). Both our approaches are also better than the work of [11] in terms of worst case condition number (κ_{worst}) values. It should be noted that the average condition numbers (κ_{avg}) (over all choices of k workers) are also quite small for both of our proposed approaches. We point out that the methods in [20] and [8] are developed for matrix-vector multiplication, so those are not applicable for this comparison.

In our next experiment we compare the mean-squared error (MSE) of the different matrix-matrix multiplication methods for their respective worst case scenarios when $n = 18$ and $s = 3$. For matrix-matrix case, we define MSE as

$$\text{MSE} = \frac{||\mathbf{A}^T\mathbf{B} - \widehat{\mathbf{A}^T\mathbf{B}}||^2}{||\mathbf{A}^T\mathbf{B}||^2} \times 100\%$$

where $\widehat{\mathbf{A}^T\mathbf{B}}$ is the recovered result and $\mathbf{A}^T\mathbf{B}$ is the actual result. Here, the matrices $\mathbf{A}$ and $\mathbf{B}$ are of size $15,000 \times 10080$ and $15,000 \times 12000$ respectively. We simulate errors in the worker node computations by adding white Gaussian noise to the calculated submatrix products obtained from the worker

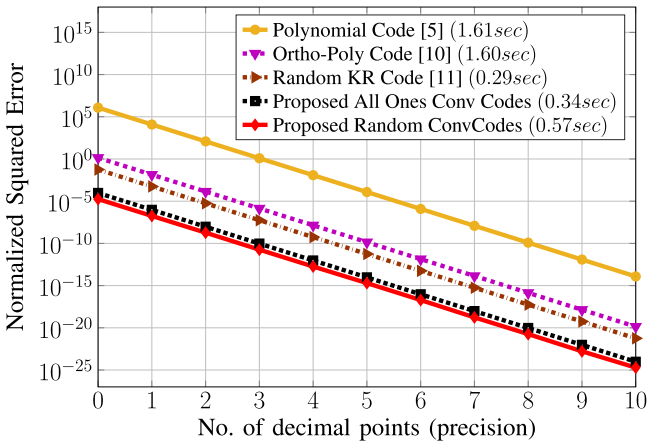

Fig. 8. Normalized MSE vs. number of decimal points of precision for different coded computation schemes for distributed matrix-matrix multiplication over $n = 18$ workers and $s = 3$ stragglers. The decoding time is reported for the different approaches in parentheses in the legend.

TABLE III

COMPARISON OF κ_{worst} AND κ_{avg} FOR MATRIX-VECTOR MULTIPLICATION FOR $n = 30$ AND $s = 2$ WITH $\gamma = \frac{1}{25}$

METHODS	κ_{worst}	κ_{avg}
POLYNOMIAL CODE [5]	2.293×10^{13}	7.51×10^{12}
CONVOLUTIONAL CODE [20]	5.124×10^{4}	1450.0
ORTHO-POLY CODE [10]	7902.6	98.92
RAND. KR PROD. CODE [11]	3642.7	160.82
CIRC. AND ROT. MATRIX [12]	52	12.04
PROP. ALL-ONES CONV. CODE	**2868.3**	**817.25**
PROP. RANDOM CONV. CODE	**1374.6**	**114.44**

nodes and sweeping the range of SNRs. The results appear in Fig. 4 (for additive Gaussian noise) and Fig. 8 (for round-off errors). In Fig. 4 we observe that even at $SNR = 70\ dB$, our approach is around 9, 4 and 2 orders of magnitude better than [5], [10] and [11]. The corresponding decoding time is also reported in the legend which shows that the decoding time for our approaches compare quite well with other approaches. The behavior of the curves in Fig. 8 is similar in nature.

B. Comparing κ_{worst} and MSE for Matrix-Vector Case

We carry out an experiment to compare the worst case condition number of the decoding matrix for different approaches for matrix-vector multiplication. Table III shows the worst case condition number for a scenario with $n = 30$ workers, with $s = 2$ stragglers where each worker node can store $\gamma_A = \frac{1}{25}$ fraction of matrix $\mathbf{A}$. From the table, it is clear that the approaches in [5] and [20] provide much larger condition numbers in comparison to the others. We can also see that our proposed approaches provide lower worst case condition numbers (κ_{worst}) than the approaches [10] and [11]. It should be noted that the average condition numbers (κ_{avg}) are also quite small for both of our proposed approaches.

In our next experiment we compare the normalized MSE of the different methods for their respective worst case scenarios.

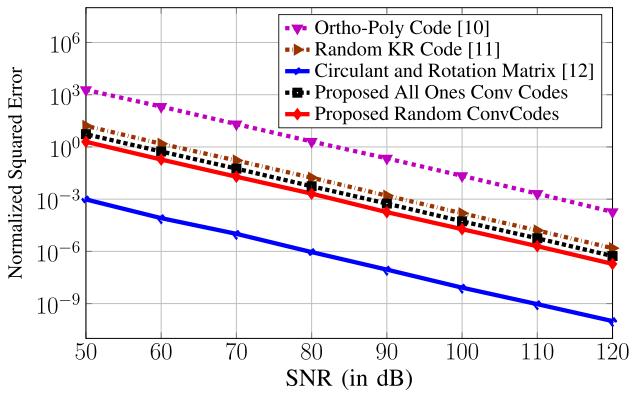

Fig. 9. Normalized MSE vs SNR plot for matrix-vector multiplication for $n = 30$ and $s = 2$.

TABLE IV

COMPARISON OF OUR PROPOSED METHODS. $n = 11, k_A = k_B = 3$ AND $\mathbf{A}$ AND $\mathbf{B}$ HAVE SIZE 10000×12600

METRICS	METHODS	$\gamma = \frac{2}{5}$	$\gamma = \frac{5}{14}$	$\gamma = \frac{7}{20}$
DECODING TIME	ALL ONES	$0.35s$	$0.36s$	$0.39s$
	RANDOM	$0.39s$	$1.16s$	$2.89s$
κ_{worst} FOR $\tilde{\mathbf{G}}_{\mathcal{I}}$	ALL ONES	95.2	275.9	395.6
	RANDOM	76.9	112.2	117.5
κ_{worst} FOR SQR. SUBMAT. OF $\tilde{\mathbf{G}}_{\mathcal{I}}$	ALL ONES	96.5	277.9	397.8
	RANDOM	7.46×10^{6}	9.64×10^{17}	1.11×10^{28}

For matrix-vector case, we define MSE as

$$\text{MSE} = \frac{||\mathbf{A}^T\mathbf{x} - \widehat{\mathbf{A}^T\mathbf{x}}||^2}{||\mathbf{A}^T\mathbf{x}||^2} \times 100\%$$

where $\widehat{\mathbf{A}^T\mathbf{x}}$ is the recovered result and $\mathbf{A}^T\mathbf{x}$ is the actual result. We consider the same scenario with $n = 30$ and $s = 2$ where we have matrix $\mathbf{A}$ of size $30,000 \times 31,500$ and a vector $\mathbf{x}$ of length $30,000$. We want to compute the product $\mathbf{A}^T\mathbf{x}$. Fig. 9 shows the normalized MSE of the different approaches for different SNR. From the figure we can see that our proposed approaches perform significantly better than all other schemes except the scheme of [12]. This supports our condition number results in Table III. For example, at $SNR = 60dB$, the approach in [11] provides around 1.6% error whereas our All-ones and Random convolutional code approaches provide only 0.5% and 0.2% error, respectively, for the worst case.

C. Comparing [12] and Our Approach

It can be observed that the recent preprint of [12] has the best κ_{worst} and MSE numbers for both the matrix-matrix and matrix-vector scenarios. However, our work has much simpler encoding (additions/subtractions in the all-ones case) and decoding (peeling decoder) than their method. Our work is also the first to propose a convolutional coding strategy for this problem.

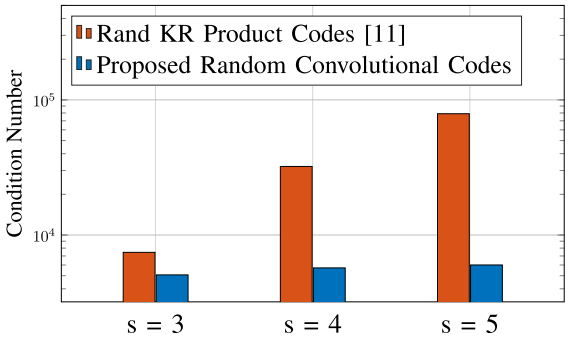

Fig. 10. Comparison of κ_{worst} for matrix-vector multiplication between the method in [11] and our proposed random convolutional code approach for $n = 20$ with $s = 3, 4$ and 5. To find κ_{worst}, the proposed method used $\gamma = \frac{1}{15}, \frac{1}{14}, \frac{1}{13}$ for $k = 17, 16, 15$, respectively.

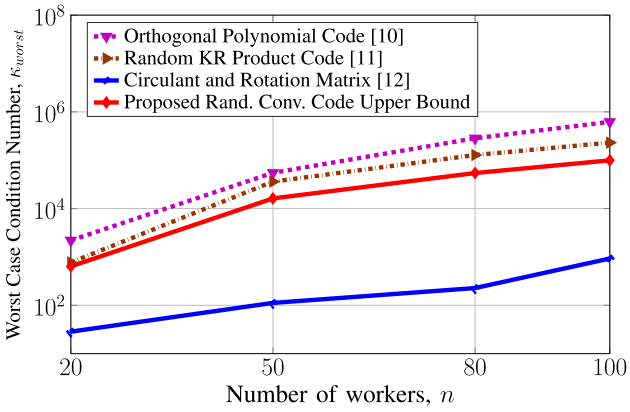

Fig. 11. Comparison for worst case condition number for matrix-matrix multiplication for $s = 2$ for different n.

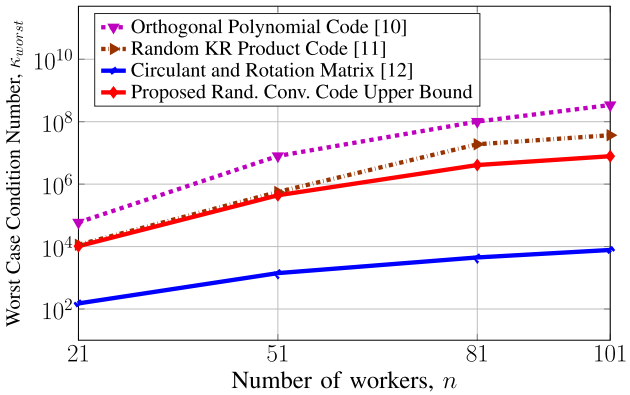

Fig. 12. Comparison for worst case condition number for matrix-matrix multiplication for $s = 3$ for different n.

D. Comparing [11] and Our Approach

The Random KR Product code-based approach can be considered as specific instance of our random scaling method where the scaling is applied to a trivial all-ones parity matrix, instead of a carefully designed $\mathbf{Y}_{\bar{b},\bar{a}}(D)$. As both approaches are random and pick the best choices, we conducted an experiment where we ran 100 trials for both methods (with $n = 20$ and $s = 3, 4, 5$) and picked the respective best choices (see Fig. 10 for the corresponding worst case condition numbers). It is clear that the structure imposed in our construction definitely improves the condition number as compared to the work of [11].

E. Comparing Our All-ones and Random Approaches

Recall that for our methods q_A and q_B increase when $\gamma_A - 1/k_A$ and $\gamma_B - 1/k_B$ become smaller (*cf.* Sections IV-B and IV-C). Table IV, shows a comparison of our proposed approaches in terms of decoding time and worst case condition number for three different values of $\gamma = \gamma_A = \gamma_B$. The following inferences can be drawn.

- The decoding time remains more or less constant for the all-ones case, whereas it can increase with decreasing γ because of solving LS problem for the random case.
- The worst case condition number for the all-ones case continues to increase with decreasing γ, whereas it saturates for the random case.
- For the all-ones case, the worst case condition numbers of both matrices ($\tilde{\mathbf{G}}_{\mathcal{I}}$ and full rank square submatrix of $\tilde{\mathbf{G}}_{\mathcal{I}}$) are almost the same for different γ. However, if the entries of $\mathbf{R}$ are random Gaussian, then the difference between these two condition numbers is very large.

F. Comparing the Approaches for Different n

We now compare the worst case condition numbers for the different approaches for matrix-matrix multiplication by sweeping across a range of number of workers. We choose recovery threshold, $k = 18, 48, 78$ and 98 with $s = 2$ and 3, and show the comparison of κ_{worst} in Figs. 11 and 12. It can be seen that the theoretical upper bounds of the condition number for our proposed random convolutional code approach are significantly smaller than those of [10] in all the cases. These upper bounds are comparable with the condition numbers of the approach [11], however, in Fig. 10, we have shown that the actual worst case condition numbers of our approaches can be significantly smaller than [11]. We have omitted the plot of [5] in the figures as the corresponding numbers (for larger n) are way too high $\geq 10^{21}$. It should be noted that we do not have order results on the growth of the worst case condition number (see [10], [12]). However, order results are often quite loose and computable upper bounds can be more useful.

As discussed previously, the All-ones approach does not have an upper bound on the worst case condition number with increasing q. Nevertheless, its performance in practice is very good. To illustrate this further, we chose matrices $\mathbf{A}$ and $\mathbf{B}$ of sizes 10000×10080 and 10000×12000, respectively and choose a setting of $k = 48$ with $s = 2$ and 3. In both cases, we pick randomly chosen 250 sets of k worker nodes and report the corresponding worst case normalized error percentage in Fig. 13. The smaller normalized error for both "All-ones" and "Random" cases when compared to [10] and [11] confirm that these approaches are robust and efficient for different n and s.

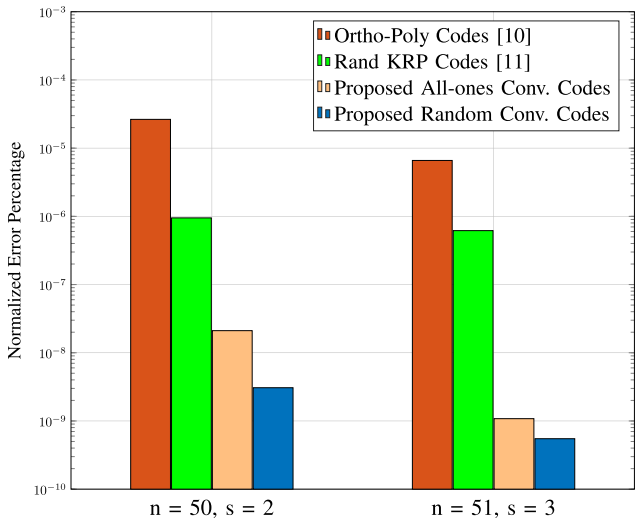

Fig. 13. Comparison of Normalized error percentage matrix-matrix multiplication among the methods in [10], [11] and our proposed convolutional code approaches for $n = 50$ and 51 with $s = 2$ and 3, respectively.

VII. Conclusion and Future Work

Most current approaches for coded computation work within the framework of block codes. In this work we presented a convolutional approach to coded matrix computation. Our codes possess simple encoding and decoding algorithms. We demonstrated novel connections between the analysis of numerical stability of our codes and the properties of large block Toeplitz matrices. The normalized MSE performance of our codes is better than most of the existing known approaches (except [12]). In particular as shown in Fig. 4 at 100 dB, our approach is better than [5], [10] and [11] by 9, 4 and 2 orders of magnitude respectively; Fig. 8 and 13 also demonstrate similar trends. Furthermore, our schemes have much lower decoding times than [5], [10] and [12].

There are several opportunities for future work. It would be interesting to consider other classes of convolutional codes for coded computation and attempt to characterize their properties. In particular, our work only considers column-wise decompositions of matrices $\mathbf{A}$ and $\mathbf{B}$. Considering both row and column-wise decompositions within the convolutional coding context are of interest. In this work, we developed schemes to mitigate the stragglers, which are assumed to be slow or failed nodes and considered as erasures; however, considering whether convolutional codes can be applied for efficient error-decoding (rather than erasure decoding) can be an interesting avenue for future work.

Appendix

A. Proof of Theorem 1 and Corollary 2 (MDS Property of Our Codes)

We begin by a formal description of the field in which the polynomials in the indeterminate D lie. Consider the set of real infinite sequences $\{u_r, u_{r+1}, \dots\}$ for $r \in \mathbb{Z}$ that start at some finite integer index r, and continue thereafter. These sequences can be treated as elements of the formal Laurent series [29] in indeterminate D with coefficients from $\mathbb{R}$, i.e., $\mathbf{u}(D) = \sum_{i=r}^{\infty} u_i D^i$. Let us denote the ring of formal Laurent series over $\mathbb{R}$ as $\mathbb{R}((D))$ under the normal addition and multiplication of formal power series. It can be shown [25] that $\mathbb{R}((D))$ forms a field, i.e., each non-zero element in it has a corresponding inverse. Thus, the polynomials $\mathbf{u}(D) = \sum_{i=0}^{\ell} u_i D^i$ that we consider in this work are members of $\mathbb{R}((D))$ and can be added, multiplied and divided to obtain other members of $\mathbb{R}((D))$. The zero element and identity element are precisely the real number 0 and the real number 1 within this field.

The proof of Theorem 1 is an immediate consequence of Lemma 2 below since any $k \times k$ submatrix of $G(D)$ is of the form $\mathbf{X}(D)$ given in the lemma.

Lemma 2: Consider a square matrix $\mathbf{X}(D)$ such that

$$\mathbf{X}(D) = \begin{bmatrix} (D^{a_0})^{b_0} & (D^{a_1})^{b_0} & \dots & (D^{a_{v-1}})^{b_0} \\ (D^{a_0})^{b_1} & (D^{a_1})^{b_1} & \dots & (D^{a_{v-1}})^{b_1} \\ \vdots & \vdots & \vdots & \vdots \\ (D^{a_0})^{b_{v-1}} & (D^{a_1})^{b_{v-1}} & \dots & (D^{a_{v-1}})^{b_{v-1}} \end{bmatrix}$$

where a_i and b_j are positive integers for $0 \le i, j \le v-1$ such that $0 \le a_0 < a_1 < \cdots < a_{v-1}$ and $0 \le b_0 < b_1 < \cdots < b_{v-1}$. Then $\mathbf{X}(D)$ is nonsingular, i.e., its determinant is a non-zero polynomial in D. Furthermore, if $\mathbf{R}$ is a $v \times v$ matrix with entries chosen i.i.d. from a continuous distribution, then $\mathbf{R} \circ \mathbf{X}(D)$ (where $\circ$ denotes the Hadamard product) is nonsingular with probability 1.

The proof of Lemma 2 involves Schur polynomials that are defined next.

Definition 3: Let $\lambda_0 \ge \lambda_1 \ge \dots \lambda_{v-1}$ be non-negative integers and let $\lambda = (\lambda_0, \dots, \lambda_{v-1})$. Then,

$$\mathcal{S}_\lambda(x_0, \dots, x_{v-1}) = \sum_T x_0^{t_0} x_1^{t_1} \dots x_{v-1}^{t_{v-1}} \tag{12}$$

where the summation is over all semistandard Young tableaux T of shape λ [30].

A Young diagram of shape λ consists of a collection of boxes arranged in left-justified rows. The i-th row has λ_i boxes. A semistandard Young tableau T is obtained by filling the boxes with the integers $0, \dots, v-1$ such that entries are in ascending order from left to right in the rows and in strictly increasing order from top to bottom in the columns. The t_i values in (12) are obtained by counting the occurrences of the number i in tableau T.

Proof: Matrix $\mathbf{X}(D)$ can be written upon permuting some rows as $\hat{\mathbf{X}}(D)$, which is given by

$$\hat{\mathbf{X}}(D) = \begin{bmatrix} (D^{a_0})^{\lambda_0+v-1} & (D^{a_1})^{\lambda_0+v-1} & \dots & (D^{a_{v-1}})^{\lambda_0+v-1} \\ (D^{a_0})^{\lambda_1+v-2} & (D^{a_1})^{\lambda_1+v-2} & \dots & (D^{a_{v-1}})^{\lambda_1+v-2} \\ \vdots & \vdots & \vdots & \vdots \\ (D^{a_0})^{\lambda_{v-1}} & (D^{a_1})^{\lambda_{v-1}} & \dots & (D^{a_{v-1}})^{\lambda_{v-1}} \end{bmatrix}$$

where we can assume that $\lambda_0 \ge \lambda_1 \ge \cdots \ge \lambda_{v-1}$. We need to prove that the determinant of $\hat{\mathbf{X}}(D)$ is non-zero. According to [30] (Chapter 1),

$$\begin{aligned} \det(\hat{\mathbf{X}}(D)) &= \det\left(\mathbf{Z}(D^{a_0}, D^{a_1}, \dots, D^{a_{v-1}})\right) \\ &\quad \times \mathcal{S}_\lambda\left(D^{a_0}, D^{a_1}, \dots, D^{a_{v-1}}\right), \end{aligned}$$

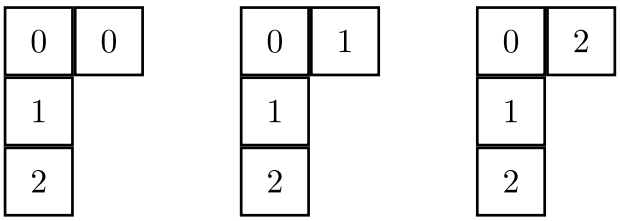

Fig. 14. Young tableaux of shape $\lambda = (2,1,1)$ leads to three different distribution for $T = \{(2,1,1),(1,2,1),(1,1,2)\}$ which helps to obtain $\mathcal{S}_\lambda\left(D, D^2, D^4\right)$.

where

$$\begin{aligned}&\mathbf{Z}(D^{a_0},\ldots,D^{a_{v-1}})\\ &=\begin{bmatrix}(D^{a_0})^{v-1} & (D^{a_1})^{v-1} & \ldots & (D^{a_{v-1}})^{v-1}\\ (D^{a_0})^{v-2} & (D^{a_1})^{v-2} & \ldots & (D^{a_{v-1}})^{v-2}\\ \vdots & \vdots & \vdots & \vdots\\ D^{a_0} & D^{a_1} & \ldots & D^{a_{v-1}}\\ 1 & 1 & \ldots & 1\end{bmatrix}. \end{aligned} \tag{13}$$

Note that $\det\left(\mathbf{Z}(D^{a_0}, D^{a_1},\ldots,D^{a_{v-1}})\right)$ is a non-zero polynomial in D as it is a Vandermonde matrix.

Furthermore, based on Definition 3, $\mathcal{S}_\lambda\left(D^{a_0}, D^{a_1},\ldots,D^{a_{v-1}}\right)$ consists of the sum of terms of the form $(D^{a_0})^{t_0}\ (D^{a_1})^{t_1}\ \ldots\ (D^{a_{v-1}})^{t_{v-1}}$ all of which have positive coefficients. Thus, it follows that $\mathcal{S}_\lambda\left(D^{a_0}, D^{a_1},\ldots,D^{a_{v-1}}\right)$ is not the zero-polynomial. ∎

Proof of Corollary 2: To see the extension, we note that $\det(\mathbf{R}\circ\mathbf{X}(D))$ is a polynomial in D whose coefficients in turn are multivariate polynomials in the elements of $\mathbf{R}$, i.e., $\{r_{i,j}\}, 0 \le i,j \le v-1$. Based on the proof above, it is clear that setting $\mathbf{R}$ to be a matrix of all-ones results in a nonsingular matrix. This implies that $\det(\mathbf{R}\circ\mathbf{X}(D))$ is not identically zero. Next, the elements of $\mathbf{R}$ are chosen i.i.d. from a continuous distribution. Therefore the probability that all the coefficients evaluate to zero over the random choice is also zero. ∎

Example 4 (Illustration of Lemma 2): Suppose that $v = 3$ and consider the square submatrix,

$$\mathbf{E} = \begin{bmatrix} D^4 & D^8 & D^{16}\\ D^2 & D^4 & D^8\\ D & D^2 & D^4\end{bmatrix}$$

where $\lambda_0 = 2, \lambda_1 = 1$ and $\lambda_2 = 1$, so $\lambda = (2,1,1)$. The determinant of $\mathbf{E}$ is given by

$$\begin{aligned}\det(\mathbf{E}) \ &= \ \mathcal{S}_\lambda\left(D, D^2, D^4\right) \ \times \det\left(\begin{bmatrix} D^2 & D^4 & D^8\\ D & D^2 & D^4\\ 1 & 1 & 1\end{bmatrix}\right)\\ &= \ \mathcal{S}_\lambda\left(D, D^2, D^4\right) \ \times \left[\left(D-D^2\right)\left(D^2-D^4\right)\left(D-D^4\right)\right].\end{aligned}$$

The Schur polynomial can be obtained from Fig. 14 as

$$\begin{aligned}\mathcal{S}_\lambda\left(D, D^2, D^4\right) = \left(D^4\right)^2\left(D^2\right)^1(D)^1 + \left(D^4\right)^1\left(D^2\right)^2(D)^1\\ + \left(D^4\right)^1\left(D^2\right)^1(D)^2 = \ D^{11} + D^9 + D^8.\end{aligned}$$

B. Proof of Theorem 2

Let $\bar{b}$ be a vector of length $2q-1$, whose entries are indexed as $\bar{b}_\ell, -(q-1) \le \ell \le (q-1)$. A Toeplitz matrix of size $q\times q$, denoted by $\mathrm{Toeplitz}(\bar{\mathrm{b}})$ is such that its (i,j)-th entry is given by $\bar{b}_{i-j}$ for $i \in [q], j \in [q]$. Thus, it is such that each diagonal is a constant from top-left to bottom-right.

Our proof of Theorem 2 relies on a result from [27]. Consider a $kq \times kq$ matrix $\tilde{\mathbf{B}}$ that has Toeplitz blocks of size $q\times q$ with the (i,j)-th block specified by the $(2q-1)$-length vector $\bar{b}^{i,j}$. To be precise, for $i = 0,1,\ldots,(k-1),\ j = 0,1,\ldots,(k-1)$,

$$(\tilde{\mathbf{B}})_{i,j} = \mathrm{Toeplitz}(\bar{b}^{i,j}).$$

The result in [27] shows that the minimum and maximum eigenvalues of such a matrix can be bounded by computing the minimum and maximum of the eigenvalues of the following (*much smaller*) $k\times k$ Fourier transform (FT) matrix $\mathbf{B}(\omega)$ over the frequency parameter ω. The (i,j)-the entry of $\mathbf{B}(\omega)$ is defined by simply computing the Fourier transform of the corresponding vector $\bar{b}^{i,j}$, i.e.,

$$\left(\mathbf{B}(\omega)\right)_{i,j} = \sum_{\ell=-(q-1)}^{(q-1)} \bar{b}_\ell^{i,j} e^{-\mathrm{i}\omega\ell}.$$

We can now state the result.

Lemma 3 (Theorem 3 of [27]):

(i) For all q, the eigenvalues of $\tilde{\mathbf{B}}$ lie in

$$\left[\min_{\omega\in[-\pi,\pi]}\lambda_{\min}\mathbf{B}(\omega)\ ,\ \max_{\omega\in[-\pi,\pi]}\lambda_{\max}\mathbf{B}(\omega)\right].$$

(ii) Furthermore,

$$\lim_{q\to\infty}\lambda_{\min}\left(\tilde{\mathbf{B}}\right) = \min_{\omega\in[-\pi,\pi]}\lambda_{\min}\left(\mathbf{B}(\omega)\right); \tag{14}$$

$$\lim_{q\to\infty}\lambda_{\max}\left(\tilde{\mathbf{B}}\right) = \max_{\omega\in[-\pi,\pi]}\lambda_{\max}\left(\mathbf{B}(\omega)\right). \tag{15}$$

In other words, the behavior of the eigenvalues of $\tilde{\mathbf{B}}$ which is a $kq\times kq$ matrix can be studied instead by computing the eigenvalues of the $k\times k$ matrix $\mathbf{B}(\omega)$ and finding its minimum and maximum eigenvalues over the range $\omega \in [-\pi,\pi]$.

The next two lemmas below help prove that $\tilde{\mathbf{G}}_{\mathcal{I}}\tilde{\mathbf{G}}_{\mathcal{I}}^T$ has Toeplitz blocks.

Let $\mathbf{U}$ and $\mathbf{L} = \mathbf{U}^T$ denote square upper and lower shift matrices respectively, i.e., $\mathbf{U}$ is a $q\times q$ matrix such that

$$\mathbf{U}_{ij} = \begin{cases}1 & \text{if } j = i+1\\ 0 & \text{otherwise.}\end{cases}$$

Thus, for instance if $q = 5$, then

$$\mathbf{U} = \begin{bmatrix}0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}. \tag{17}$$

Lemma 4: Let $h \ge \max(i,j)$. Then

$$(\tilde{\mathbf{D}}^{h;i})(\tilde{\mathbf{D}}^{h;j})^T = \begin{cases}\mathbf{U}^{i-j} & \text{if } i > j,\\ \mathbf{L}^{j-i} & \text{if } i \le j.\end{cases}$$

Note that the matrices on the RHS above are Toeplitz.

Proof: We only prove the case when $i > j$ as the other part is very similar. The product $(\tilde{\mathbf{D}}^{h;i})(\tilde{\mathbf{D}}^{h;j})^T$ can be expressed as

$$\begin{bmatrix}\mathbf{0}_{q\times i} & \mathbf{I}_q & \mathbf{0}_{q\times(h-i)}\end{bmatrix} \times \begin{bmatrix}\mathbf{0}_{j\times q}\\ \mathbf{I}_q \\ \mathbf{0}_{(h-j)\times q}\end{bmatrix} = \begin{bmatrix}\mathbf{0}_{(q-(i-j))\times(i-j)} & \mathbf{I}_{q-(i-j)} \\ \mathbf{0}_{(i-j)\times(i-j)} & \mathbf{0}_{(i-j)\times(q-(i-j))}\end{bmatrix} = \mathbf{U}^{i-j}.$$

■

Lemma 5: Let $\tilde{\mathbf{G}}_\ell$ denote the ℓ-th block-column of $\tilde{\mathbf{G}}$. For $\ell = 0, 1, \ldots, k-1$,

$$[\tilde{\mathbf{G}}_\ell \tilde{\mathbf{G}}_\ell^T]_{i,j} = \begin{cases}\mathbf{I}_q & \text{if } i = j = \ell, \\ \mathbf{0} & \text{otherwise.}\end{cases}$$

For $\ell = k + \tilde{\ell}$, $\tilde{\ell} = 0, 1, \ldots, s-1$, and for $i \geq j$

$$[\tilde{\mathbf{G}}_\ell \tilde{\mathbf{G}}_\ell^T]_{i,j} = \begin{cases} r_{i\tilde{\ell}}^2 \mathbf{I}_q & \text{if } i = j, \\ r_{i\tilde{\ell}} r_{j\tilde{\ell}} \mathbf{U}^{a_{\tilde{\ell}}(b_i - b_j)} & \text{if } i > j.\end{cases}$$

Since the matrix is symmetric, specifying its entries for $i \geq j$ is sufficient.

Proof: This follows directly by using Lemma 4 and the definition of $\tilde{\mathbf{G}}_\ell$. ■

Furthermore, using the property that the sum of Toeplitz matrices is Toeplitz, we can conclude that for any subset $\mathcal{I} \subset \{0, \ldots, n-1\}$ such that $|\mathcal{I}| = k$, we have that the matrix $\tilde{\mathbf{G}}_\mathcal{I} \tilde{\mathbf{G}}_\mathcal{I}^T$ is a matrix with Toeplitz blocks.

For ease of presentation let $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$ where $\mathcal{I}_1 \subseteq \{0, \ldots, k-1\}$, $\mathcal{I}_2 \subseteq \{k, \ldots, n-1\}$ and $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$ and $\tilde{\ell} = \ell - k$. Then, for $0 \leq i, j \leq k-1$ and $i \geq j$ we can express the (i,j)-th block of $(\tilde{\mathbf{G}}_\mathcal{I})(\tilde{\mathbf{G}}_\mathcal{I})^T$ as follows.

$$[(\tilde{\mathbf{G}}_\mathcal{I})(\tilde{\mathbf{G}}_\mathcal{I})^T]_{i,j} = \begin{cases} (\sum_{\ell\in\mathcal{I}_2} r_{i\tilde{\ell}}^2)\mathbf{I}_q + \mathbb{1}_{i\in\mathcal{I}_1}\mathbf{I}_q, & \text{if } i = j, \\ \sum_{\ell\in\mathcal{I}_2} r_{i\tilde{\ell}} r_{j\tilde{\ell}} \mathbf{U}^{a_{\tilde{\ell}}(b_i - b_j)} & \text{if } i > j.\end{cases} \tag{18}$$

where $\mathbb{1}$ denotes the indicator function. By symmetry it suffices to specify $[(\tilde{\mathbf{G}}_\mathcal{I})(\tilde{\mathbf{G}}_\mathcal{I})^T]_{i,j}$ for $i \geq j$. Each of the blocks is of dimension $q \times q$.

Proof of Theorem 2: We emphasize that our matrix $[(\tilde{\mathbf{G}}_\mathcal{I})(\tilde{\mathbf{G}}_\mathcal{I})^T]$ (see (18)) has Toeplitz blocks. Let $\tilde{\mathbf{B}} = (\tilde{\mathbf{G}}_\mathcal{I})(\tilde{\mathbf{G}}_\mathcal{I})^T$. Then we have

$$\tilde{\mathbf{B}}_{i,j} = [(\tilde{\mathbf{G}}_\mathcal{I})(\tilde{\mathbf{G}}_\mathcal{I})^T]_{i,j} = \begin{cases} (\sum_{\ell\in\mathcal{I}_2} r_{i\tilde{\ell}}^2)\mathbf{I}_q + \mathbb{1}_{i\in\mathcal{I}_1}\mathbf{I}_q, & \text{if } i = j, \\ \sum_{\ell\in\mathcal{I}_2} r_{i\tilde{\ell}} r_{j\tilde{\ell}} \mathbf{U}^{a_{\tilde{\ell}}(b_i - b_j)} & \text{if } i > j,\end{cases} \tag{19}$$

where $\tilde{\ell} = \ell - k$. Observe $\mathbf{U}^a$ is a matrix with 1's on the $(a+1)$-th diagonal and zeros everywhere else. Thus, $\tilde{\mathbf{B}}_{i,j}$ is a Toeplitz matrix with the $(a_{\tilde{\ell}}(b_i - b_j))$-th diagonal equal to $r_{i\tilde{\ell}} r_{j\tilde{\ell}}$. Therefore, the corresponding sequence $\bar{b}^{i,j}$ for $i > j$ is given by

$$\bar{b}_m^{i,j} = \begin{cases} r_{i\tilde{\ell}} r_{j\tilde{\ell}} & \text{if } m = -a_{\tilde{\ell}}(b_i - b_j), \\ 0 & \text{otherwise.}\end{cases}$$

Thus, following the discussion above, we obtain

$$(\mathbf{B}(\omega))_{i,j} = \sum_{\ell\in\mathcal{I}_2} r_{i\tilde{\ell}} r_{j\tilde{\ell}} \exp\left(\mathrm{i}\omega a_{\tilde{\ell}}(b_i - b_j)\right).$$

The expressions above can equivalently be expressed as replacing D with $e^{\mathrm{i}\omega}$ and then computing the inner product of $\mathbf{G}_\mathcal{I}(e^{j\omega})(i,:)$ with $(\mathbf{G}_\mathcal{I}(e^{\mathrm{i}\omega})(j,:))^*$. Therefore, we can compactly represent

$$\mathbf{B}(\omega) = \mathbf{G}_\mathcal{I}(e^{\mathrm{i}\omega})\mathbf{G}_\mathcal{I}(e^{\mathrm{i}\omega})^*.$$

This concludes the proof. ■

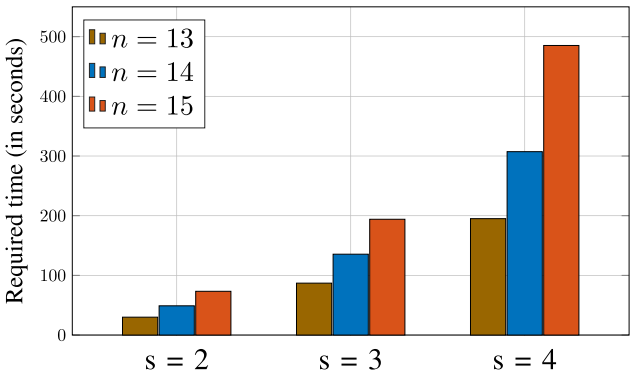

Fig. 15. Comparison of required time to find a good choice of $\mathbf{R}$ for different n and s.

C. Search Time for Random Convolutional Coding

We run an experiment to tabulate the time needed to find a good random matrix $\mathbf{R}$. We run 50 trials to find the best $\mathbf{R}$ for $n = 13, 14, 15$ with $s = 2, 3, 4$. It should be noted that the choice of $\mathbf{R}$ depends on all $\binom{n}{s}$ choices of stragglers. Fig. 15 shows the corresponding time for different pairs of

$$\tilde{\mathbf{G}} = \begin{bmatrix} \overbrace{\mathbf{I}_q & 0 & \ldots & 0}^{k \text{ block-columns}} & \overbrace{r_{00}\tilde{\mathbf{D}}^{a_0\ b_{k-1};a_0\ b_0} & \ldots & r_{(s-1)0}\tilde{\mathbf{D}}^{a_{s-1}b_{k-1};a_{s-1}b_0}}^{n-k \text{ block-columns}} \\ 0 & \mathbf{I}_q & \ldots & 0 & r_{10}\tilde{\mathbf{D}}^{a_0\ b_{k-1};a_0\ b_1} & \ldots & r_{1(s-1)}\tilde{\mathbf{D}}^{a_{s-1}b_{k-1};a_{s-1}b_1} \\ \vdots & \vdots & \ddots & \vdots & r_{20}\tilde{\mathbf{D}}^{a_0\ b_{k-1};a_0\ b_2} & \ldots & r_{2(s-1)}\tilde{\mathbf{D}}^{a_{s-1}b_{k-1};a_{s-1}b_2} \\ 0 & 0 & 0 & \mathbf{I}_q & r_{(k-1)0}\tilde{\mathbf{D}}^{a_0\ b_{k-1};a_0\ b_{k-1}} & \ldots & r_{(k-1)(s-1)}\tilde{\mathbf{D}}^{a_{s-1}b_{k-1};a_{s-1}b_{k-1}} \end{bmatrix}. \tag{16}$$

n and s. From the figure, it can be seen that our system (a processor with CPU speed $3.5\ GHz$ and $16\ GB$ RAM) needs only around 8 minutes to find a good choice of $\mathbf{R}$ for even $n = 15$ and $s = 4$. In other cases, the required amount of time is even lesser. This indicates that for a reasonable system size, we do not need to wait too long to obtain a good choice of $\mathbf{R}$ that ensures that the worst case condition number is bounded. And it should be noted that this is a one-time cost for designing the coding scheme for a system with n worker nodes which is resilient to $s = n - k$ stragglers.

REFERENCES

[1] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Operating System Design and Implementation*. San Diego, CA, USA: USENIX Association, 2008, pp. 29–42. [Online]. Available: https://dl.acm.org/doi/10.5555/1855741.1855744

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

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

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

[5] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 4403–4413.

[6] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 2100–2108.

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

[8] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–40, 2019.

[9] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 5152–5160.

[10] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," *IEEE Trans. Inf. Theory*, vol. 67, no. 5, pp. 2758–2785, Jan. 2021.

[11] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Random Khatri-Rao-product codes for numerically-stable distributed matrix multiplication," in *Proc. 57th Annu. Conf. Commun., Control, Comput. (Allerton)*, Sep. 2019, pp. 253–259.

[12] A. Ramamoorthy and L. Tang, "Numerically stable coded matrix computations via circulant and rotation matrix embeddings," 2019, *arXiv:1910.06515*. [Online]. Available: https://arxiv.org/abs/1910.06515

[13] A. B. Das, L. Tang, and A. Ramamoorthy, "C^{3}LES: Codes for coded computation that leverage stragglers," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.

[14] L. Tang, K. Konstantinidis, and A. Ramamoorthy, "Erasure coding for distributed matrix multiplication for matrices with bounded entries," *IEEE Commun. Lett.*, vol. 23, no. 1, pp. 8–11, Jan. 2019.

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

[16] A. Ramamoorthy, A. B. Das, and L. Tang, "Straggler-resistant distributed matrix computation via coding theory: Removing a bottleneck in large-scale data processing," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 136–145, May 2020.

[17] A. E. Yagle, "Fast algorithms for matrix multiplication using pseudo-number-theoretic transforms," *IEEE Trans. Signal Process.*, vol. 43, no. 1, pp. 71–76, Jan. 1995.

[18] V. Y. Pan, "How bad are vandermonde matrices?" *SIAM J. Matrix Anal. Appl.*, vol. 37, no. 2, pp. 676–694, Jan. 2016.

[19] L. Tang, "Algebraic approaches for coded caching and distributed computing," Ph.D. dissertation, Dept. Elect. Comput. Eng., Iowa State Univ., Ames, IA, USA, 2020.

[20] A. B. Das and A. Ramamoorthy, "Distributed matrix-vector multiplication: A convolutional coding approach," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2019, pp. 3022–3026.

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

[22] A. B. Das and A. Ramamoorthy, "Coded sparse matrix computation schemes that leverage partial stragglers," 2020, *arXiv:2012.06065*. [Online]. Available: https://arxiv.org/abs/2012.06065

[23] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[24] R. M. Gray, "Toeplitz and circulant matrices: A review," *Found. Trends Commun. Inf. Theory*, vol. 2, no. 3, pp. 155–239, 2006.

[25] I. Niven, "Formal power series," *Amer. Math. Monthly*, vol. 76, no. 8, pp. 871–889, 1969.

[26] X.-D. Zhang, *Matrix Analysis and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2017.

[27] H. Gazzah, P. A. Regalia, and J. P. Delmas, "Asymptotic eigenvalue distribution of block Toeplitz matrices and application to blind SIMO channel identification," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 1243–1251, Mar. 2001.

[28] *Straggler Mitigation Codes*. Accessed: Jul. 13, 2021. [Online]. Available: https://github.com/anindyabijoydas/StragglerMitigateConvCodes

[29] T. Fuja, C. Heegard, and M. Blaum, "Cross parity check convolutional codes," *IEEE Trans. Inf. Theory*, vol. 35, no. 6, pp. 1264–1276, Nov. 1989.

[30] I. G. Macdonald, *Symmetric Functions and Hall Polynomials*, 2nd ed. Oxford, U.K.: Oxford Univ. Press, 2015.

Anindya Bijoy Das (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, in 2014, and the M.Eng. degree in electrical engineering from Iowa State University, Ames, in 2018, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. His research interests include coding theory and machine learning.

Aditya Ramamoorthy (Senior Member, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Delhi, and the M.S. and Ph.D. degrees from the University of California, Los Angeles (UCLA).

He is currently a Professor of electrical and computer engineering and (by courtesy) of mathematics with Iowa State University. His research interests are in the areas of classical/quantum information theory and coding techniques with applications to distributed computation, content distribution networks, and machine learning.

Dr. Ramamoorthy was a recipient of the 2020 Mid-Career Achievement in Research Award, the 2019 Boast-Nilsson Educational Impact Award and the 2012 Early Career Engineering Faculty Research Award from Iowa State University, the 2012 NSF CAREER Award, and the Harpole-Pentair Professorship in 2009 and 2010. He served as an Editor for IEEE TRANSACTIONS ON INFORMATION THEORY from 2016 to 2019 and IEEE TRANSACTIONS ON COMMUNICATIONS from 2011 to 2015.

Namrata Vaswani (Fellow, IEEE) received the B.Tech. degree from IIT-Delhi, India, in 1999, and the Ph.D. degree from the University of Maryland, College Park, in 2004. She is currently the Anderlik Professor of electrical and computer engineering with Iowa State University. Her research interests include intersection of statistical machine learning/data science, computer vision, and signal processing. She is an IEEE Fellow (class of 2019) for her contributions to dynamic high-dimensional structured data recovery. She was a recipient of the IEEE Signal Processing Society (SPS) Best Paper Award (2014) for her T-SP paper on Modified-CS coauthored with her graduate student Lu; the University of Maryland ECE Distinguished Alumni Award (2019); and the Iowa State Mid-Career Achievement in Research Award (2019).