Bivariate Polynomial Coding for Efficient Distributed Matrix Multiplication

Burak Hasırcıoğlu, *Graduate Student Member, IEEE*, Jesús Gómez-Vilardebó, *Senior Member, IEEE*, and Deniz Gündüz, *Senior Member, IEEE*

***Abstract*—Coded computing is an effective technique to mitigate "stragglers" in large-scale and distributed matrix multiplication. In particular, univariate polynomial codes have been shown to be effective in straggler mitigation by making the computation time depend only on the fastest workers. However, these schemes completely ignore the work done by the straggling workers resulting in a waste of computational resources. To reduce the amount of work left unfinished at workers, one can further decompose the matrix multiplication task into smaller sub-tasks, and assign multiple sub-tasks to each worker, possibly heterogeneously, to better fit their particular storage and computation capacities. In this work, we propose a novel family of *bivariate polynomial codes* to efficiently exploit the work carried out by straggling workers. We show that bivariate polynomial codes bring significant advantages in terms of upload communication costs and storage efficiency, measured in terms of the number of sub-tasks that can be computed per worker. We propose two bivariate polynomial coding schemes. The first one exploits the fact that bivariate interpolation is always possible on a rectangular grid of evaluation points. We obtain such points at the cost of adding some redundant computations. For the second scheme, we relax the decoding constraints and require decodability for almost all choices of the evaluation points. We present interpolation sets satisfying such decodability conditions for certain storage configurations of workers. Our numerical results show that bivariate polynomial coding considerably reduces the average computation time of distributed matrix multiplication. We believe this work opens up a new class of previously unexplored coding schemes for efficient coded distributed computation.**

***Index Terms*—Bivariate polynomial interpolation, coded computation, distributed computation, distributed matrix multiplication, polynomial codes.**

I. Introduction

THE AVAILABILITY of massive datasets and model sizes makes computation tasks for machine learning applications so demanding that they cannot be carried out on a single machine within a reasonable time frame. To speed up learning, most demanding computation tasks, e.g., matrix multiplication, are distributed to multiple dedicated servers, called *workers*. However, due to unpredictable delays in their service time, some workers, called *stragglers*, may become a bottleneck for the overall computation task. One can mitigate the effects of stragglers by assigning redundant computations. In particular, one can treat stragglers as random erasures, and improve the computation time by creating redundant computations similarly to channel coding for erasure channels. Assuming all the workers start computing simultaneously, we define the *computation time* as the time from the start until sufficiently many computations that allow decoding AB at the master are received. It excludes the communication time as well as the encoding and decoding times. For the matrix multiplication task, the authors in [3] propose to partition one of the matrices, encode its partitions by using an MDS code, and send coded partitions to the workers together with the other matrix (which is not partitioned or coded). It is then shown that the full matrix multiplication can be decoded by using only a subset of the multiplications between the coded partitions of the first matrix and the second matrix.

In [4], polynomial codes are proposed to speed up the multiplication of matrices A and B. In this scheme, a *master* partitions A row-wise and B column-wise. Then, two separate encoding polynomials, whose coefficients are the partitions of A and B, respectively, are generated. The master evaluates both polynomials at the same point and sends the evaluations to the workers, which multiply them and return the result to the master. Using a subset of the responses from the fastest workers, the full multiplication can be recovered. This scheme is optimal in terms of the *download rate*, which is defined as the ratio between the total number of bits needed to be downloaded from the workers and the number of bits needed to represent the result of the multiplication. In [5], MatDot codes are proposed, which use an alternative partitioning scheme for matrices; that is, A is partitioned column-wise and B is partitioned row-wise. The authors show that, compared to [4], their approach improves the *recovery threshold*, which is defined as the minimum number of responses the master must receive from the workers to guarantee decoding the product AB. However, both the amount of computation each worker should carry out, referred to as the *computation cost*, and the download rate are higher than in [4]. Also in [5], PolyDot codes are proposed as an interpolation

Manuscript received March 2, 2021; revised July 8, 2021 and August 9, 2021; accepted August 10, 2021. Date of publication August 20, 2021; date of current version September 20, 2021. This work was supported in part by the European Research Council (ERC) through Starting Grant BEACON under Grant 677854, and in part by the U.K. EPSRC through the CHIST-ERA Program under Grant EP/T023600/1. The work of Jesús Gómez-Vilardebó was supported in part by the Catalan Government under Grant SGR2017-1479, and in part by the Spanish Government under Grant RTI2018-099722-B-100 (ARISTIDES). Parts of this paper were presented in 2020 IEEE International Symposium on Information Theory (ISIT) [1] and 2020 IEEE Global Communication Conference (Globecom) [2]. *(Corresponding author: Burak Hasırcıoğlu.)*

Burak Hasırcıoğlu and Deniz Gündüz are with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, U.K. (e-mail: b.hasircioglu18@imperial.ac.uk; d.gunduz@imperial.ac.uk).

Jesús Gómez-Vilardebó is with Centre Tecnológic de Telecomunicacions de Catalunya (CTTC/CERCA), 08860 Barcelona, Spain (e-mail: jesus.gomez@cttc.es).

This article has supplementary downloadable material available at https://doi.org/10.1109/JSAIT.2021.3105365, provided by the authors.

Digital Object Identifier 10.1109/JSAIT.2021.3105365

between polynomial codes in [4], and MatDot codes by trading off between the recovery threshold and the computation and download costs. In [6], the same problem is studied, and entangled polynomial codes are proposed, which improve the recovery threshold in [5] under a fixed computation cost and a fixed download rate. Generalized PolyDot codes are proposed in [7] achieving the same recovery threshold in [6]. In [8], batch multiplication of matrices, i.e., A_iB_i, $i \in [L]$ where $L > 1$, is studied and cross subspace alignment (CSA) codes are proposed. It is shown that, in the batch multiplication setting, CSA codes improve the upload-download cost trade-off compared to applying entangled polynomial codes separately for each multiplication task in the batch. Since the decoding process in the polynomial coding approaches is based on polynomial interpolation, numerical stability becomes an important research problem for practical implementations. In [9]–[12], numerically stable coding schemes are proposed for distributed coded matrix multiplication problem.

In all of these approaches, the result of all the work assigned to a worker is communicated to the master only if it is finished completely. Workers that fail to complete all their assignments by the time the recovery threshold is reached are treated as erasures, which implies ignoring completely the work done by them. Such an approach is sub-optimal, especially if the workers' speeds are close to each other, in which case, the ignored workers have probably completed a significant portion of the work assigned to them [13], [14]. To exploit the partially completed work done by stragglers, a multi-message approach is considered in [14]–[16], where workers' tasks are divided into smaller sub-tasks, and the result of each sub-task is communicated to the master as soon as it is completed. The approaches in [14], [16] are based on uncoded computation and a hybrid of uncoded and coded computation, respectively, and it is shown that uncoded computation may be more beneficial if the workers' computation speeds are similar. In our work, we allow the workers to be heterogeneous, as encountered in serverless computing, peer-to-peer applications, or edge computing, and consider coded computation with multi-message communication. A similar setting is considered in [15], and product codes are employed.

In [13], a hierarchical coding framework for the straggler exploitation problem is proposed, also taking into account the decoding complexity. This work is extended to matrix-vector and matrix-matrix multiplications in [17]. It is shown that while gaining in terms of the decoding complexity, the computation time of hierarchical coding is only slightly larger than [15] with univariate polynomial coding. Thus, the benefits of hierarchical coding are significant mainly if the decoding time is comparable to the computation time.

In all of the aforementioned polynomial-type coding approaches [3]–[8], univariate polynomials are used. As we will show in this paper, under fixed storage capacities at the workers, in univariate polynomial coding, dividing a task into sub-tasks by a given factor reduces the fraction of work that can be done by the workers by the same factor, resulting in inefficient use of workers' storage capacity and upload costs. Product codes proposed in [15], which are basically a combination of two MDS codes, partially address this issue. However, in product codes, computations at workers are not one-to-any replaceable, i.e., some might be redundant, and hence, not useful, which results in poor performance in various scenarios. Moreover, univariate polynomial codes, as well as product codes, impose certain constraints preventing fully heterogeneous workloads across workers.

In this work, we propose bivariate polynomial codes to improve the computation time of distributed matrix-matrix multiplication under limited storage at the workers. The main contributions of this work can be summarized as follows:

- We first show the limitation of univariate polynomial codes in terms of both computational and storage efficiency when extended to the multi-message setting.
- We introduce bivariate polynomial coding schemes to address these limitations. Interpolation of bivariate polynomials cannot be guaranteed by simply requiring all evaluation points to be distinct. Here, we introduce the concepts of regular (always invertible), and almost regular (almost always invertible) interpolation matrices.
- We first extend the product coding scheme of [15] to bivariate polynomial coding, which leads to a regular interpolation matrix by imposing a particular rectangular grid structure on the interpolation points. This strategy attains maximum storage efficiency, but the computation efficiency can be limited due to redundant computations.
- Next, we propose two novel bivariate coding schemes. We demonstrate that unlike univariate schemes, for bivariate coding, the order by which the computations are done at the workers has a non-trivial impact on decodability; and hence, we impose a special computation order for the tasks assigned to each worker. These schemes achieve maximum computation efficiency by completely avoiding redundant computations. Their storage efficiency is limited, yet higher than that of univariate schemes. We further propose two alternative bivariate polynomial codes with higher storage efficiency at the cost of a slight decrease in computation efficiency.
- We numerically validate our findings assuming a shifted exponential model for computation speeds, and show the superiority of the proposed bivariate schemes compared to univariate alternatives and product codes.
- While polynomial codes have been extensively studied with numerous applications in practice, to the best of our knowledge, our work provides the first examples of bivariate polynomial code constructions with superior performance compared to their univariate counterparts.

II. System Model and Problem Formulation

In our system, illustrated in Fig. 1, a master server wants to multiply two matrices $A \in \mathbb{R}^{r\times s}$ and $B \in \mathbb{R}^{s\times c}$, r, s, $c \in \mathbb{Z}^+$, by offloading partial computations to N workers with heterogeneous storage capacities and computation speeds. The master divides A horizontally and B vertically into K and L partitions, respectively, such that $A = \begin{bmatrix} A_1^T & A_2^T & \cdots & A_K^T \end{bmatrix}^T$ and $B = \begin{bmatrix} B_1 & B_2 & \cdots & B_L \end{bmatrix}$, where $A_i \in \mathbb{R}^{\frac{r}{K}\times s}$, $\forall i \in [1:K]$[1] and $B_j \in \mathbb{R}^{s\times \frac{c}{L}}$, $\forall j \in [1:L]$. The master generates and sends to worker $i \in [1:N]$, $m_{A,i}$ and $m_{B,i}$ coded matrix partitions $\tilde{A}_{i,k}$ and $\tilde{B}_{i,l}$ based on A and B, respectively, for $k \in [1:m_{A,i}]$ and $l \in [1:m_{B,i}]$, where $m_{A,i}$ and $m_{B,i} \in \mathbb{Z}^+$, and $\tilde{A}_{i,k} \in \mathbb{R}^{\frac{r}{K}\times s}$, $\tilde{B}_{i,l} \in \mathbb{R}^{s\times \frac{c}{L}}$. Thus, worker $i \in [1:N]$ is assumed to store a fraction $M_{A,i} = \frac{m_{A,i}}{K}$ of A and $M_{B,i} = \frac{m_{B,i}}{L}$ of B. How these coded matrix partitions are generated depends on the specific coding scheme employed. In this work, they will be obtained as linear combinations of the original matrix partitions.

Depending on the coding scheme employed, worker i can compute all, or a subset of the products of coded matrix partitions assigned to it, i.e., $\tilde{A}_{i,k}\tilde{B}_{i,l}$, $k \in [1:m_{A,i}]$, $l \in [1:m_{B,i}]$

[1] Given $a < b$, we define $[a:b] \triangleq \{a, a+1, a+2, \ldots, b-1, b\}$

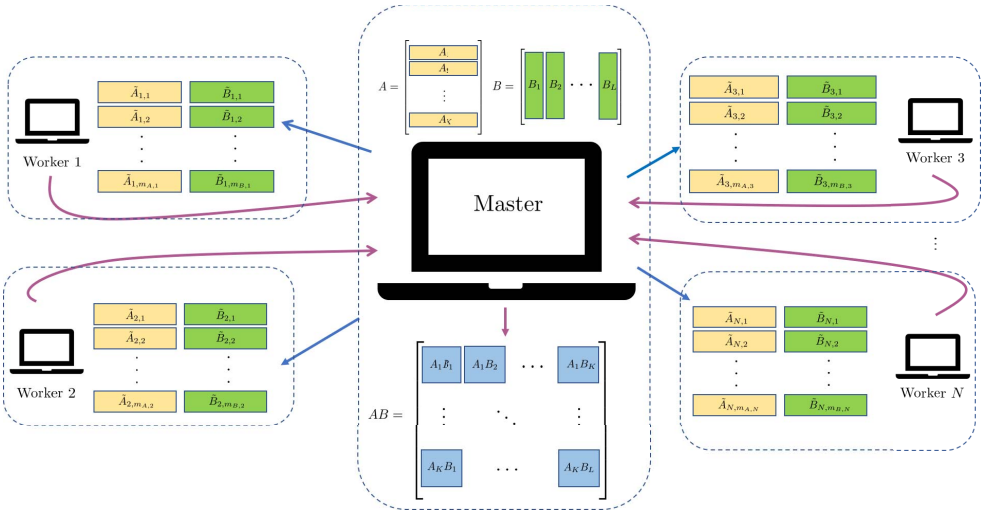

Fig. 1. The master computes AB by offloading partial computations to N workers.

TABLE I
COMPARISON OF THE KEY PARAMETERS AND SYSTEM CONSTRAINTS

Scheme	$C_{\max,i}$	C_{wasted}	System constraints
UPC	$M_A M_B$	$NM_AM_B - 1$	$m_{A,i} = m_{B,i} = 1$
UPC-PC	$\frac{M_{A,i}M_{B,i}}{m_i}$	$\sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_i^2}$	$m_{A,i} = m_{B,i} = m_i \in [1 : \min(K, L)]$
B-PROC	$M_A M_B$	$\sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}$ $+(n_A M_B - 1)(1 - \frac{M_A}{m_A})$ $+(n_B M_A - 1)(1 - \frac{M_B}{m_B})$	$N = n_A n_B$ $m_{A,i} = m_A$, $m_{B,i} = m_B$ $K \leq n_A m_B$, $L \leq n_B m_A$
BPC-VO	$M_{A,i}M_{B,i}$	$\sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}$	$m_{A,i} = 1$ and $m_{B,i} \leq L$ **or** $m_{A,i} \geq 1$ and $m_{B,i} = L$
BPC-HO	$M_{A,i}M_{B,i}$	$\sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}$	$m_{B,i} = 1$ and $m_{A,i} \leq K$ **or** $m_{B,i} \geq 1$ and $m_{A,i} = K$
BPC-NZO	$M_{A,i}M_{B,i}$	$\sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}$ $+(\mu_B - 2)(\frac{L}{\mu_B} - 1)\frac{1}{KL}$	$\mu_B \mid L$, $\mu_B \mid m_{B,i}$, $m_{A,i} = K$ and $m_{B,i} \leq L$ **or** $m_{B,i} = \mu_B$,$\mu_B \mid L$, $m_{B,i} \leq L$ and $m_{A,i} \leq K$ **or** $m_{A,i} = 1$, $m_{B,i} < \mu_B$ and $\mu_B \mid L$
BPC-ZZO	$M_{A,i}M_{B,i}$	$\sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}$ $+(\mu_A - 2)(\frac{K}{\mu_A} - 1)\frac{1}{KL}$	$\mu_A \mid K$, $\mu_A \mid m_{A,i}$, $m_{A,i} \leq K$ and $m_{B,i} = L$ **or** $m_{A,i} = \mu_A$,$\mu_A \mid K$, $m_{A,i} \leq K$ and $m_{B,i} \leq L$ **or** $\mu_A \mid K$, $m_{A,i} < \mu_A$ and $m_{B,i} = 1$

in a prescribed order, which is also specific to the coding scheme. We denote by η_i the maximum number of computations worker i can provide, which can be possibly used by the master for decoding AB. Thus, $\eta_i \leq m_{A,i}m_{B,i}$, and the specific value of η_i depends on the coding scheme. In order to exploit the partial work done by straggling workers, the results of these individual products are sent to the master as soon as they are finished. The master collects the responses from the workers until the received set of computations allow the master to uniquely recover AB. Then, the master instructs all the workers to stop computing and decodes AB. Note that the recovery threshold, which is defined as the minimum number of computations that guarantee the decodability of AB, does not have to be a fixed quantity in our setting. Depending on the coding scheme, R_{th} can be a function of the collected computations by the master.

As is common in the related literature, we specify the storage capacity at workers separately for each of the two matrices, i.e., $M_{A,i}$ and $M_{B,i}$. However, in practice, it is more appropriate to assume a total storage capacity at each worker, which can be freely allocated between the partitions of the two matrices. Assume that the rows of A and the columns of B require the same amount of storage. We define the storage capacity of worker i, denoted by $s_i \in \mathbb{N}^+$, as the sum of the total number of rows of A and the total number of columns of B that the i^{th} worker can store. Accordingly, for a given K, L, and s_i, we allocate $m_{A,i}$ and $m_{B,i}$ to maximize η_i subject to $M_{A,i}r + M_{B,i}c = s_i$. Defining $C_{\text{part}} \triangleq \frac{1}{KL}$ as the fraction of work corresponding to a single partial product $\tilde{A}_{i,j}\tilde{B}_{i,l}$, the maximum fraction of work that can be done by worker i is given by

$$C_{\max,i} \triangleq \eta_i C_{\text{part}} = \frac{\eta_i}{m_{A,i}m_{B,i}} M_{A,i}M_{B,i}. \tag{1}$$

Under the same storage constraints, a code that can provide more fraction of work uses its storage more efficiently; hence, $C_{\max,i}$ will be used to measure the *storage efficiency*.

We define C_{wasted} as the worst-case fraction of wasted computations with respect to the full product, AB. There are two sources of wasted computations. Firstly, depending on the coding scheme, some of the computations completed by the workers may not be used in decoding AB. Secondly, when R_{th} is reached, the master instructs all the workers to stop their computations and the ongoing computations of the workers are wasted. We assume that the communication time for the stop signal to reach from the master to the workers is short enough

that the workers receive this instruction before finishing their ongoing computations. In the following sections, we compute the fraction of the wasted computations of the second type based on this assumption. If this assumption does not hold, the wasted computations of the second type may increase. While it is out of the scope of this work, designing coding schemes that minimize wasted computations of the second type when the relative speed of communication is comparable to the speed of a unit computation can be an interesting challenge for a follow-up study.

For a fixed N and a total storage capacity at worker i, s_i, our objective is to minimize the average computation time of AB. This depends on the statistics of the computation speeds of the workers and is difficult to obtain in closed form. Instead, we use $C_{max,i}$ and C_{wasted} as proxies for the performance of a code. These metrics do not depend on the worker's speeds and provide general indicators on the code performance. Note that, especially in heterogeneous settings, in which some workers may be much faster than the others, the higher fraction of work provided by faster workers helps to finish the task earlier. Therefore, storage efficiency, or $C_{\max,i}$, is a factor to be optimized to improve the average computation time. Moreover, low C_{wasted} implies that more of the available computation capacity across the workers is exploited towards completing the desired computation. Therefore, to minimize the average computation time, we are interested in maximizing $C_{\max,i}$ and minimizing C_{wasted}. Table I summarizes the key code parameters $C_{\max,\text{i}}$, C_{wasted} and system constraints for the schemes considered in this work. A detailed discussion on these parameters is postponed to the later sections.

III. Univariate Schemes

We first review the codes based on univariate polynomial interpolation.

Univariate Polynomial Codes (UPC): With the univariate polynomial codes presented in [4], the master encodes the matrix partitions using the polynomials

$$A(x) = A_1 + A_2 x + \cdots + A_K x^{K-1}, \tag{2}$$

$$B(x) = B_1 + B_2 x^K + \cdots + B_i x^{(i-1)K} + \cdots + B_L x^{(L-1)K}. \tag{3}$$

The master sends $A(x_i)$ and $B(x_i)$ to worker i, $i \in [1:N]$, for some distinct $x_i \in \mathbb{R}$. Thus, every worker receives one coded partition of A and one partition of B, i.e., $m_{A,i} = m_{B,i} = 1$ and $M_{A,i} = M_A = 1/K$, $M_{B,i} = M_B = 1/L$, $\forall i \in [1:N]$. Worker i computes $A(x_i)B(x_i)$ and returns the result. No other computations are done at the workers, and thus $\eta_i = m_{A,i}m_{B,i} = 1$. Once the master receives any $R_{th} = KL$ results, it can interpolate the polynomial

$$A(x)B(x) = \sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{i-1+K(j-1)} \tag{4}$$

of degree $KL-1$. Observe that the coefficients of the interpolated polynomial correspond to the KL sub-products A_iB_j, $\forall i \in [1:K]$, $\forall j \in [1:L]$ of AB. Finally, notice that

$$C_{\max,i} = C_{\text{part}} = \frac{1}{KL} = M_A M_B. \tag{5}$$

Observe that with $N > R_{th}$, this scheme can tolerate up to $N - R_{th}$ stragglers. It helps to reduce the average computation time thanks to the parallelization afforded by redundant workers. However, all the work done by the $N - R_{\text{th}}$ slowest workers are ignored. In the worst case, where the $N - R_{\text{th}} + 1$ slowest workers finish simultaneously, we have

$$C_{\text{wasted}} = (N - R_{\text{th}})C_{\text{part}} = (N - R_{\text{th}})\frac{1}{KL} = NM_AM_B - 1. \tag{6}$$

We observe from (6) that without changing the number of workers N or the storage capacities of workers M_A, M_B, it is not possible to improve C_{wasted}, and thus reduce the amount of work lost at workers. Next, we provide an extension of UPC such that C_{wasted} can be improved by increasing K and L, exploiting the partial work done at the workers.

Univariate Polynomial Codes With Partial Computations (UPC-PC): To exploit the partial work done at slower workers, we present an extension of UPC, which is based on the multi-message approach and also allow heterogeneous storage capacities at workers. The main idea is to divide the task assigned to a worker into smaller sub-tasks, i.e., larger K and L, and allowing the workers to store multiple partitions. Specifically, we allow worker i to store $m_i = m_{A,i} = m_{B,i}$ coded partitions of A and B, i.e., $M_{A,i} = m_i/K$ and $M_{B,i} = m_i/L$. For worker i, the master evaluates $A(x)$ and $B(x)$ at m_i different points $\{x_{i,1}, \ldots, x_{i,m_i}\}$ such that $x_{i,k} \neq x_{j,l}$ if $(i,k) \neq (j,l)$, $\forall i,j \in [1:N]$ and $\forall k,l \in [1:m_i]$. Worker i computes $A(x_{i,j})B(x_{i,j})$ consecutively for $j \in [1:m_i]$ and sends each result as soon as completed. Observe that multiplications are only allowed between $A(x)$ and $B(x)$ evaluated at the same $x_{i,k}$ values, and thus $\eta_i = m_i$, $\forall i \in [1:N]$. The master is able to interpolate $A(x)B(x)$ as soon as it receives $R_{th} = KL$ responses from the workers. Thus

$$C_{\text{part}} = \frac{1}{KL} = \frac{M_{A,i}M_{B,i}}{m_i^2}, \tag{7}$$

$$C_{\max,i} = m_i C_{\text{part}} = \frac{M_{A,i}M_{B,i}}{m_i}. \tag{8}$$

The total fraction of wasted work in the worst case, in which all the workers were up to finish its ongoing partial multiplication once the R_{th}-th result is received by the master, is given by

$$C_{\text{wasted}} = (N-1)C_{\text{part}} = (N-1)\frac{1}{KL} = \sum_{i=1}^{N-1}\frac{M_{A,i}M_{B,i}}{m_i^2}. \tag{9}$$

As opposed to UPC, according to (9), UPC-PC can improve C_{wasted} by increasing K and L. Stragglers might be unable to complete the full task assigned to them, but they might complete a part of it. Clearly, the smaller are the sub-tasks executed at workers, the smaller is the work that can be lost at a straggler. On the other hand, UPC-PC makes quite an inefficient use of the storage capacity at workers. Observe that even if a worker has enough storage to fully store A and B, i.e., $M_{A,i} \geq 1$ and $M_{B,i} \geq 1$, it, alone, can only provide $\min\{K,L\}$ partial computations. Indeed, for a fixed storage capacity at the workers, i.e., $M_{A,i}$ and $M_{B,i}$ are kept constant, the maximum fraction of work done at a worker, $C_{\max,i}$, decreases while K and L increases to improve C_{wasted}, which results in less efficient use of the storage capacities of the workers. The bivariate schemes presented in the next section address this problem.

IV. Bivariate Polynomial Coding

For bivariate polynomial coding schemes, we encode the matrix partitions of A and B by using the following encoding

polynomials

$$A(x) = A_1 + A_2 x + \cdots + A_K x^{K-1}, \tag{10}$$
$$B(y) = B_1 + B_2 y + \cdots + B_L y^{L-1}. \tag{11}$$

Depending on the coding scheme, coded matrix partitions $\tilde{A}_{i,k}$ and $\tilde{B}_{i,l}$ are either direct evaluations of the encoding polynomials $A(x)$ and $B(y)$, respectively, or the evaluations of their derivatives. Hence, the workers obtain evaluations of the bivariate polynomial

$$A(x)B(y) = \sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{i-1} y^{j-1} \tag{12}$$

or of its derivatives, by multiplying the coded matrix partitions $\tilde{A}_{i,k}$ and $\tilde{B}_{i,l}$'s. Finally, the master interpolates the bivariate polynomial $A(x)B(y)$ by making use of these products.

In addition to allowing heterogeneous storage capacities across workers, bivariate coding schemes allow different numbers of stored coded partitions of A and B for each worker, i.e., $m_{A,i} \neq m_{B,i}$ in general. The maximum number of computations a worker can generate is $\eta_i = m_{A,i} m_{B,i}$, resulting in $C_{\max,i} = m_{A,i} m_{B,i} C_{\text{part}} = M_{A,i} M_{B,i}$.

Observe that, unlike univariate polynomial coding schemes, for a given storage capacity $M_{A,i}$ and $M_{B,i}$, the maximum amount of work done at worker i, $C_{\max,i}$, does not decrease with $m_{A,i}$ and $m_{B,i}$. In univariate schemes, the reason behind storage inefficiency is that the workers can use each evaluation of $A(x)$ and $B(x)$ only for one partial computation. For example, $A(x_{i,k})B(x_{i,l})$, for $k \neq l$, cannot be used to interpolate $A(x)B(x)$ in a univariate scheme. Bivariate polynomial coding eliminates this limitation and allows the workers to provide additional useful computations at no additional storage cost. Moreover, like UPC-PC, bivariate polynomial codes can exploit the computational power of the stragglers.

Since $A(x)B(y)$ has KL coefficients, we need KL partial computations to interpolate it. However, in some cases, the set of first KL computations may not be enough to guarantee decodability and more computations may be required. Thus, the number of computations needed to guarantee decodability satisfies $R_{th} \geq KL$. Moreover, at the instant when the R_{th}-th computation is completed by a worker, all the ongoing computations become unnecessary. Therefore, in this setting, we have two sources of wasted computations: $R_{th} - KL$ redundant computations that have been received by the master but not used for the actual interpolation, and the ongoing computations at all the workers except the worker providing R_{th}-th computation. We consider the worst-case scenario and assume all these ongoing computations at the remaining $N-1$ workers are close to end. Thus, we count them as wasted computations.

As a result, the maximum wasted fraction of computations is given by

$$\begin{aligned} C_{\text{wasted}} &= (N-1)C_{\text{part}} + (R_{th} - KL)C_{\text{part}} \\ &= \sum_{i=1}^{N-1} \frac{M_{A,i} M_{B,i}}{m_{A,i} m_{B,i}} + \frac{R_{th}}{KL} - 1. \end{aligned} \tag{13}$$

Before presenting the bivariate schemes, we introduce some basic concepts and definitions from polynomial interpolation theory.

Definition 1: The interpolation of a bivariate polynomial of the form $A(x)B(y)$ can be formulated as a system of linear equations. The unknowns of these equations are the coefficients of $A(x)B(y)$. We define the **interpolation matrix** as the coefficient matrix of this linear system, denoted by M.

Recall that the interpolation matrices we consider result from evaluations of $A(x)B(y)$ or their derivatives at different points. The rules the coding schemes impose on the computations, e.g., computation orders, types of computations assigned to the workers, etc., may result in $\det(M)$ to become an identically zero polynomial no matter which points are chosen. This is an undesirable situation, and we should show that this does not happen for a proposed scheme. Next, we define two notions in which such undesirable structures are not imposed on the interpolation matrix.

Definition 2 [18, Definition 3.1.3]: An interpolation scheme is called **regular** if $\det(M) \neq 0$ for every set of distinct and non-zero evaluation points. On the other hand, if $\det(M) \neq 0$ for almost all choices of the evaluation points, then the interpolation scheme is called **almost regular**. Almost regularity implies that $\det(M)$ is not the zero polynomial in general and the Lebesgue measure of the evaluation points making $\{\det(M) = 0\}$ is zero in $\mathbb{R}^2$.

To understand the practical meaning of almost regularity, let us assume that we sample our evaluation points uniform randomly from the interval $[l, u]$, where $l, u \in \mathbb{R}$ and $l < u$. Since the Lebesgue measure of the evaluation points making $\det(M) = 0$ is zero, the probability of sampling such evaluation points is exactly zero. Note that this is due to using an uncountable set to sample our evaluation points and there are infinitely many possible choices of evaluation points. Even if we have countably many bad choices of evaluation points, the invertibility of M is guaranteed almost surely.

Univariate polynomial interpolation is regular if the evaluation points are distinct and non-zero since the corresponding interpolation matrix is a Vandermonde matrix, which is known to be invertible. However, for bivariate interpolation, there are very few cases for which sufficient conditions for regularity are known. Next, we consider one such case.

A. Bivariate Polynomial Interpolation on Rectangular Grids

It is well known that interpolation of $A(x)B(y)$ such that $A(x)$ and $B(y)$ have degrees $K-1$ and $L-1$, respectively, is regular for any rectangular grid of points $\{x_1, x_2, \ldots, x_K\} \times \{y_1, y_2, \ldots, y_L\}$ provided all x_i's and y_i's are distinct. The interpolation scheme we propose next exploits this fact. It was originally proposed in [15] using product codes. Here, we present it using bivariate polynomial codes, which is equivalent to the product-code form in terms of all the performance metrics considered in this paper. We further generalize it to allow $m_A \neq m_B$ and $n_A \neq n_B$, where $N = n_A n_B$.

Bivariate Product Coding (B-PROC): Assume that all the workers can store m_A partitions of A and m_B partitions of B, and N can be factored as $N = n_A n_B$ such that $K \leq m_A n_A$ and $L \leq m_B n_B$. The master generates n_A disjoint sets $\mathcal{X}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,m_A}\}$, $i \in [1 : n_A]$ and n_B disjoint sets $\mathcal{Y}_j = \{y_{j,1}, y_{j,2}, \ldots, y_{j,m_B}\}$, $j \in [1 : n_B]$, with distinct elements. Then, it enumerates the workers as (i, j), $i \in [1 : n_A]$, $j \in [1 : n_B]$ and sends $A(x_{i,k})$, $k \in [1 : m_A]$ and $B(y_{j,l})$, $l \in [1 : m_B]$, to worker (i, j). Worker (i, j) can compute any product $A(x_{i,k})B(y_{j,l})$. Altogether, the set of evaluation points at workers form a rectangular grid of size $m_A n_A \times m_B n_B$. Observe that, n_A workers have the evaluation $B(\hat{y})$ for any $\hat{y} \in \mathcal{Y}_j$, and each of them can compute m_A distinct evaluations of the univariate polynomial $A(x)B(\hat{y})$, of degree $K-1$ with respect to x. Once the first K of these evaluations are received at the master, $A(x)B(\hat{y})$ can be interpolated. Similarly, for a given $\hat{x} \in \mathcal{X}_i$, $A(\hat{x})B(y)$ can be interpolated from any L evaluations as it is a univariate polynomial in y with degree $L-1$. As a result, once we have the evaluations of $A(x)B(y)$

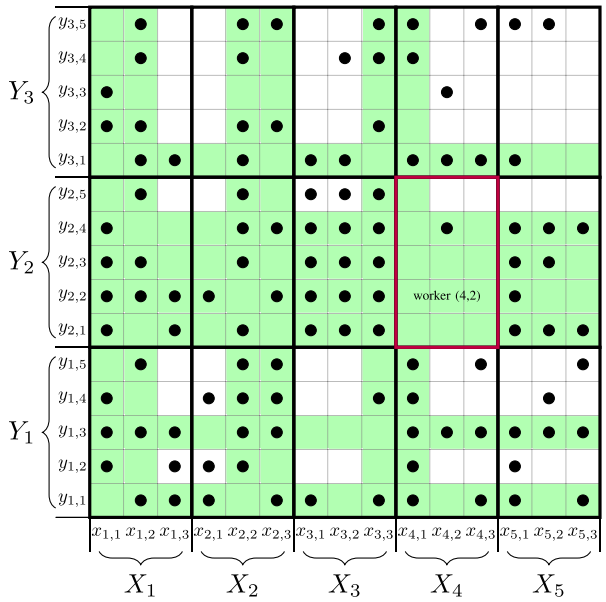

Fig. 2. The case in Example 1.

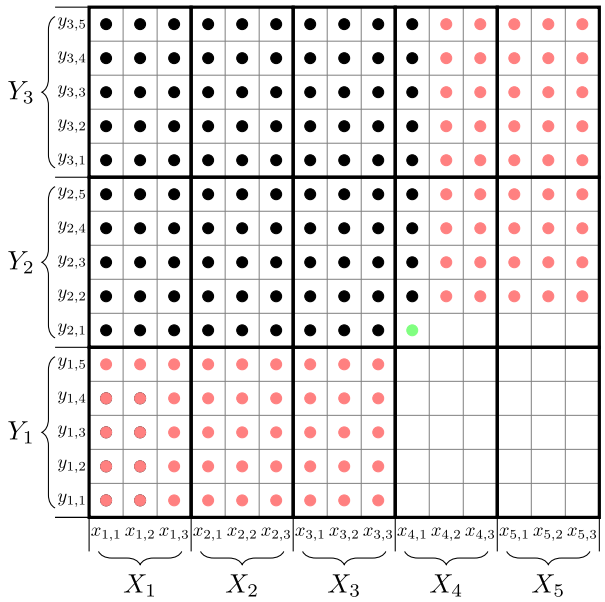

Fig. 3. Worst case scenario.

on any rectangular grid of size $K \times L$, either directly received from the workers or via univariate interpolation, the bivariate interpolation problem can be solved.

Observe, however, that the computations that were already interpolated from previous results are redundant. To minimize such computations, in [15], for the particular case of $m_A = m_B$, $n_A = n_B$, different heuristics to schedule computations at the workers were discussed.

Example 1: Let us assume that both matrices A and B are divided into $K = L = 10$ partitions, and there are $N = 15$ workers, while every worker can store $M_A = 3/10$ of A and $M_B = 5/10$ of B. We take $n_A = 5$ and $n_B = 3$. Worker (i, j) stores $\{A(x_{i,1}), A(x_{i,2}), A(x_{i,3})\}$ and $\{B(y_{j,1}), B(y_{j,2}), B(y_{j,3}), B(y_{j,4}), B(y_{j,5})\}$. Let us assume that the order of computations is random within a worker. Fig. 2 shows an instance of the received responses from the workers. Each worker is represented by a 3×5 rectangle and each filled circle represents a received computation by the master. To clarify how a worker's finished computations look like, worker $(4, 2)$ is emphasized in the figure. All the elements in the columns and the rows coloured by green can be interpolated, i.e., decoded, by using the received responses on the same column or the row. Observe that there are columns/rows coloured by green even if they have less than 10 computations, e.g., the column of $x_{4,1}$. Such rows and columns can be decoded after decoding rows and columns with at least 10 computations, by utilizing all the elements in these columns and rows after decoding. Since there must be at least 10 green columns and 10 green rows in order to decode $A(x)B(y)$, in our example, the received responses are not sufficient, although there are $110 > KL = 100$ responses received by the master.

The total fraction of the work wasted in the worst case depends on the heuristics employed. If we consider uniform random computation order at the workers, which is reported to perform well in [15], then the computations can be received at any order by the master. In the worst case, there may be $K - 1$ fully computed columns, that is, in every column there are exactly $n_B m_B$ computations, and one column with exactly L computations. Thus, in this case, $n_B m_B - L$ computations in each of the $K - 1$ fully computed columns are wasted. On the other hand, there may be $L - 1$ fully computed rows and one row with exactly K computations. In this case, $n_A m_A - K$ computations in each of the $L - 1$ fully computed rows are wasted. Thus, in total, we have $(n_B m_B - L)(K - 1) + (n_A m_A - K)(L - 1)$ wasted computations. Therefore, the worst-case R_{th} of B-PROC is

$$R_{th}^{B-PROC} = KL + (n_B m_B - L)(K - 1) + (n_A m_A - K)(L - 1). \quad (14)$$

Note that this expression is a worst-case value and depending on the received responses, the actual number of computations that guarantee decodability may be much lower. If we plug (14) into (13), the fraction of wasted computations for B-PROC in the worst case becomes

$$\begin{aligned} C_{\text{wasted}} &= \sum_{i=1}^{N-1} \frac{M_A M_B}{m_A m_B} + [(n_B m_B - L)(K - 1) \\ &\qquad + (n_A m_A - K)(L - 1)] \frac{1}{KL} \\ &= \sum_{i=1}^{N-1} \frac{M_A M_B}{m_A m_B} + (n_B M_B - 1)\left(1 - \frac{M_A}{m_A}\right) \\ &\qquad + (n_A M_A - 1)\left(1 - \frac{M_B}{m_B}\right). \end{aligned} \quad (15)$$

The expression is highly dependent on how n_A and n_B are allocated. Increasing the memory, i.e., M_A and M_B, while K and L remain constant, or increasing K and L while the memory remains constant both increase C_{wasted}. However, it is worth noting that C_{wasted} we calculated here is for the worst-case scenario, and the situation may not be that bad most of the time. For the setting in Example 1, we visualize the worst-case situation in Fig. 3.

B-PROC requires additional constraints on the system, i.e., $N = n_A n_B$, $K \leq n_A m_A$, $L \leq n_B m_B$ and homogeneous storage capacities at workers, and yet it is not possible to ensure that the first KL results arriving at the master form a regular interpolation problem. To address these issues, in the next subsection, we propose novel bivariate polynomial codes. However, showing the regularity of these schemes is a hard problem, if not impossible. Therefore, instead, we use the

notion of almost regularity, which is a relaxation of regularity and propose almost regular bivariate interpolation schemes.

B. Almost Regular Bivariate Interpolation Schemes

For the almost regular bivariate interpolation schemes we propose, the polynomial $A(x)B(y)$ is interpolated from the evaluations of it and its derivatives. Such an interpolation is known as Hermite interpolation in the literature [19, Ch. 3.6].

1) Encoding: The almost regular interpolation schemes we describe in the sequel have a common encoding procedure. Consider the polynomials in (10) and (11). To each worker $i \in [1:N]$ the master assigns a distinct evaluation point $(x_i, y_i) \in \mathbb{R}^2$, and sends the evaluations of $A(x)$ and $B(y)$, and their derivatives up to order $m_{A,i}-1$ and $m_{B,i}-1$, respectively, at (x_i, y_i). Thus, the values generated and sent to the worker i by the master are $\mathcal{A}_i \triangleq \{A(x_i), \frac{dA(x_i)}{dx}, \ldots, \frac{d^{(m_{A,i}-1)}A(x_i)}{dx^{(m_{A,i}-1)}}\}$ and $\mathcal{B}_i \triangleq \{B(y_i), \frac{dB(y_i)}{dy}, \ldots, \frac{d^{(m_{B,i}-1)}B(y_i)}{dy^{(m_{B,i}-1)}}\}$. For brevity, in the remaining of the paper, we use $\partial_k A(x_i)$ and $\partial_l B(y_i)$ to denote $\frac{d^k}{dx^k}A(x_i)$and $\frac{d^l}{dy^l}B(y_i)$, respectively.

2) Computations at Workers: For all the coding schemes, after receiving $\mathcal{A}_i$ and $\mathcal{B}_i$ from the master, each worker i starts computing, one by one, all the cross products between elements in $\mathcal{A}_i$ and those in $\mathcal{B}_i$, and sends the result of each computation to the master as soon as it is completed. We require each worker to follow a specific computation order for these multiplications according to the **priority score** of each computation, which we will define shortly for each scheme. We note that a lower priority score gives more priority to a computation. Specifically, for any worker i, each computation $\partial_k A(x_i)\partial_l B(y_i)$ for $k \in [0{:}K-1]$ and $l \in [0:L-1]$ is assigned a priority score denoted as $S(k,l)$. Worker i computes $\partial_k A(x_i)\partial_l B(y_i)$ once all the computations $\partial_{\tilde{k}} A(x_i)\partial_{\tilde{l}} B(y_i)$, $\tilde{k} \in [0:K-1]$ and $\tilde{l} \in [0:L-1]$ such that $S(\tilde{k},\tilde{l}) < S(k,l)$ are already computed. Notice that priority scores $S(k,l)$ are defined for computations that might not be available at worker i, i.e., $K > k \geq m_{A,i}$ or $L > l \geq m_{B,i}$. Whenever such a computation has the lowest priority score among all the remaining computations at worker i, the worker must discard all the remaining computations and stop.

Definition 3 (Derivative Order Space): The derivative order space of a bivariate polynomial $A(x)B(y)$ is defined as the 2-dimensional space of all its possible derivative orders. When $A(x)$ and $B(y)$ have degrees $K-1$ and $L-1$, respectively, the derivative order space becomes $\{(k,l): 0 \leq k < K, 0 \leq l < L\}$, where the tuple (k,l) represents the derivative $\partial_k A(x)\partial_l B(y)$.

Bivariate Polynomial Coding With Vertical Computation Order (BPC-VO): In this scheme, workers follow the *vertical computation order*, illustrated in Fig. 4a for $K = L = 6$. In the vertical computation order, a worker first completes a column k in the derivative order space, i.e., all the computations in $\{\partial_k A(x_i)B(y_i), \partial_k A(x_i)\partial_1 B(y_i), \ldots, \partial_k A(x_i)\partial_{L-1} B(y_i)\}$ before moving on to the computations from column $k+1$. Specifically, for any worker i, computation $\partial_k A(x_i)\partial_l B(y_i)$ for $k \in [0:K-1]$ and $l \in [0:L-1]$, has a priority score of $S_{\mathcal{V}}(k,l) \triangleq (K-1)L(\lceil \frac{l}{L} \rceil - 1) + L(k-1) + l$. Because only the computations $\partial_k A(x_i)\partial_l B(y_i)$ $k \in [0:m_{A,i}-1]$, and $l \in [0:m_{B,i}-1]$ can be computed by worker i, in order to satisfy the vertical computation order without discarding any computations, worker i can store either:

1) a single coded partition of A, and any number of coded partitions of B not more than L, i.e., $m_{A,i} = 1$ and $1 \leq m_{B,i} \leq L$, or
2) coded partitions of B equivalent to the full matrix B in size, and not more than K coded partitions of A, i.e., $1 \leq m_{A,i} \leq K$ and $m_{B,i} = L$.

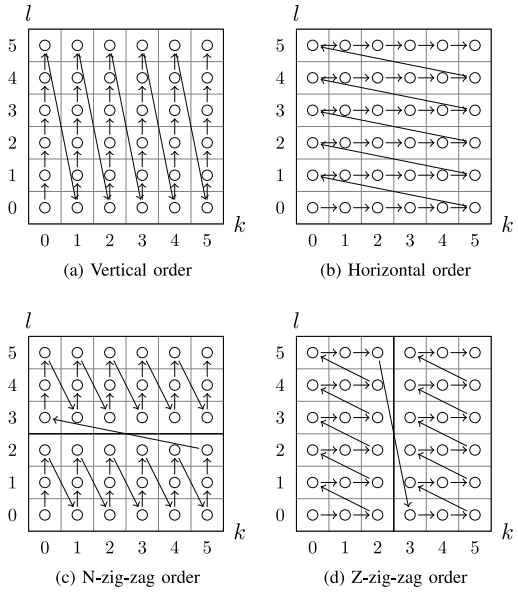

Fig. 4. Computation orders at the workers proposed in this work.

Bivariate Polynomial Coding With Horizontal Computation Order (BPC-HO): In this scheme, workers follow the *horizontal computation order*, illustrated in Fig. 4b for $K = L = 6$. In the horizontal computation order, a worker first completes a row l in the derivative order space, i.e., all the computations in $\{A(x_i)\partial_l B(y_i), \partial_1 A(x_i)\partial_l B(y_i), \ldots, \partial_{K-1} A(x_i)\partial_l B(y_i)\}$ before moving on to the computations from row $l+1$. Specifically, for any worker i, computation $\partial_k A(x_i)\partial_l B(y_i)$ has a priority score of $S_{\mathcal{H}}(k,l) \triangleq K(L-1)(\lceil \frac{k}{K} \rceil - 1) + K(l-1) + k$. As for the vertical computation order, because only computations $\partial_k A(x_i)\partial_l B(y_i)$ $k \in [0:m_{A,i}-1]$, and $l \in [0:m_{B,i}-1]$ can be computed by worker i, in order to satisfy the horizontal computation order without discarding any computations, worker i can store either:

1) a single coded partition of B, and any number of coded partitions of A not more than K, i.e., $1 \leq m_{A,i} \leq K$ and $m_{B,i} = 1$, or
2) coded partitions of A equivalent to the full matrix A, and any number of coded partitions of B not more than L, i.e., $m_{A,i} = K$ and $1 \leq m_{B,i} \leq L$.

Bivariate Polynomial Coding With N-zig-zag Computation Order (BPC-NZO): For this scheme, we relax the vertical computation order by dividing the derivative order space into L/μ_B equal horizontal *blocks*, where μ_B is a design parameter such that $\mu_B \mid L$. For computation $\partial_k A(x_i)\partial_l B(y_i)$, we assign an *N-zig-zag order* priority score of $S_{\mathcal{N}}(k,l) = (K-1)\mu_B(\lceil \frac{l}{\mu_B} \rceil - 1) + \mu_B(k-1) + l$. In Fig. 4c, we illustrate the N-zig-zag order for $K = L = 6$ and $\mu_B = 3$. For this computation order, we simply apply vertical computation order inside each horizontal block in the derivative order space starting from the lowermost block. Only when all the computations in a block are completed, the computations from the next block can start. Although they are more relaxed than the vertical computation order, in order to satisfy the N-zig-zag order without discarding any computations at worker i, one of the following conditions must be imposed on $m_{A,i}$ and $m_{B,i}$:

1) $m_{B,i}$ is a positive integer multiple of μ_B, and $m_{A,i} = K$, or

2) $m_{B,i} = \mu_B$ and $1 \leq m_{A,i} \leq K$, or,
3) $m_{A,i} = 1$ and $1 \leq m_{B,i} \leq \mu_B$

Observe that by setting $\mu_B = L$, the N-zig-zag order reduces to the vertical computation order.

Bivariate Polynomial Coding With Z-zig-zag Computation Order (BPC-ZZO): For this scheme, we relax the horizontal computation order by dividing the derivative order space into K/μ_A equal vertical blocks, where μ_A is a design parameter such that $\mu_A \mid L$. For the computation $\partial_k A(x_i)\partial_l B(y_i)$, we define the *Z-zig-zag order* priority score as $S_{\mathcal{Z}}(k,l) = (L-1)\mu_A(\lceil \frac{k}{\mu_A} \rceil - 1) + \mu_A(l-1) + k$. In Fig. 4d, we visualize the Z-zig-zag computation order when $K = L = 6$ and $\mu_A = 3$. For this computation order, we apply horizontal computation order inside each vertical block starting from the leftmost block. Again, only when all the computations in a block are completed, the computations from the next block can start. In order to satisfy the Z-zig-zag computation order without discarding any computations at worker i, we need to impose one of the following constraints on $m_{A,i}$ and $m_{B,i}$:

1) $m_{A,i}$ is a positive integer multiple of μ_A, and $m_{B,i} = L$, or
2) $m_{A,i} = \mu_A$ and $1 \leq m_{B,i} \leq L$, or
3) $m_{B,i} = 1$ and $1 \leq m_{A,i} \leq \mu_A$

Observe that setting $\mu_A = K$, we recover the horizontal computation order conditions.

3) Decoding Procedure of Almost Regular Interpolation Schemes: For all of the computation orders defined in this section, the master receives responses from the workers and decodes AB by solving a bivariate polynomial interpolation problem. That is, $A(x)B(y)$ is interpolated from the evaluations of $A(x)B(y)$ and its derivatives. Since the degree of $A(x)B(y)$ is KL, to solve the interpolation problem, the master needs at least KL computations returned from the workers. In this case, assuming, without loss of generality, the responses are received from all N workers, we have an interpolation matrix as in (16), at the bottom of the page.

In this example, the master received 3 responses from worker 1, and 2 responses from worker N. Since we see the derivatives are taken with respect to x, we can conclude that this interpolation matrix belongs to BPC-HO or BPC-ZZO.

The next theorem and the corollary characterize the number of computations needed to decode $A(x)B(y)$ in the worst-case scenario by considering the invertibility of the interpolation matrix.

Theorem 1: a) For BPC-NZO, the worst-case recovery threshold is $R_{th}^{NZO} \triangleq KL + \max\{0, (\mu_B - 2)(\frac{L}{\mu_B} - 1)\}$.

b) For BPC-ZZO, the worst-case recovery threshold is $R_{th}^{ZZO} \triangleq KL + \max\{0, (\mu_A - 2)(\frac{K}{\mu_A} - 1)\}$.

Thus, if the number of computations received by the master is at least R_{th}^{NZO} and R_{th}^{ZZO} for BPC-NZO and BPC-ZZO, respectively, then $\det(M) \neq 0$ for almost all choices of the evaluation points.

The proof of Theorem 1 is given in Section VI.

Corollary 1: BPC-VO and BPC-HO can be obtained by setting $\mu_B = L$ and $\mu_A = K$ in BPC-NZO and BPC-ZZO, respectively. Therefore, the recovery thresholds of BPC-VO and BPC-HO are $R_{th}^{VO} = R_{th}^{HO} \triangleq KL$, meaning any KL computations received by the master results in $\det(M) \neq 0$ for almost all choices of the evaluation points.

According to Corollary 1, for BPC-VO and BPC-HO, every partial computation sent by the workers is useful at the master, i.e., $R_{th} = KL$. Therefore, for these schemes, the computations are one-to-any replaceable. Thus, according to (13), we have

$$C_{\text{wasted, BPC-VO}} = C_{\text{wasted, BPC-HO}} = \sum_{i=1}^{N-1} \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}. \quad (17)$$

This is the main advantage of these schemes over B-PROC. On the other hand, while in the BPC-HO and BPC-VO, η_i is limited by the constraints imposed on $m_{A,i}$ and $m_{B,i}$, in B-PROC, all the available storage can be fully exploited. Therefore, B-PROC has a better storage efficiency $C_{\max,i}$ compared to BPC-VO and BPC-HO. The main motivation of introducing BPC-NZO and BPC-ZZO is to relax these constraints. According to Theorem 1, this can be done at the cost of potentially introducing redundant computations; however, the number of redundant computations needed is much less than those needed for B-PROC. Specifically, from (13), we have

$$C_{\text{wasted, BPC-NZO}} = \sum_{i=1}^{N-1} \frac{M_A M_B}{m_A m_B} + (\mu_B - 2)\left(\frac{L}{\mu_B} - 1\right)\frac{1}{KL},$$
$$C_{\text{wasted, BPC-ZZO}} = \sum_{i=1}^{N-1} \frac{M_A M_B}{m_A m_B} + (\mu_A - 2)\left(\frac{K}{\mu_A} - 1\right)\frac{1}{KL}. \quad (18)$$

The following example illustrates the storage efficiency of bivariate polynomial codes.

Example 2: Assume $K = L = 8$, i.e., the size of partitions of A and B are equal, and each worker can store 8 coded matrix partitions in total, i.e., $m_{A,i} + m_{B,i} = 8$. Univariate schemes can only carry out $\eta_i = m_{A,i} = m_{B,i} = 4$ computations. Instead, B-PROC can set $m_{A,i} = m_{B,i} = 4$, resulting in $\eta_i = 16$ computations. On the other hand, in BPC-VO and BPC-HO, the same worker can generate at most $\eta_i = 7$ computations by setting $m_{A,i} = 1, m_{B,i} = 7$ for BPC-VO, or $m_{A,i} = 7, m_{B,i} = 1$ for BPC-HO. It is not possible to satisfy condition 2 of BPC-VO and BPC-HO under this storage capacity. Finally, for BPC-NZO or BPC-ZZO, by setting $\mu_A = \mu_B = 4$ and $m_{A,i} = m_{B,i} = 4$, we can also reach $\eta_i = 16$. Note that BPC-NZO and BPC-ZZO may not always obtain the B-PROC storage efficiency, but they can usually perform very close.

Remark 1: Note that R_{th}^{NZO} and R_{th}^{ZZO} provided in Theorem 1 are worst-case values. Depending on the number of computations each worker sends to the master, smaller values, even

$$M = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{K-1} & \cdots & x_1^{K-1}y_1^{L-1} \\ 0 & 1 & 2x_1 & 3x_1^2 & \cdots & (K-1)x_1^{K-2} & \cdots & (K-1)x_1^{K-2}y_1^{L-1} \\ 0 & 0 & 2 & 6x_1 & \cdots & (K-1)(K-2)x_1^{K-3} & \cdots & (K-1)(K-2)x_1^{K-3}y_1^{L-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 & \cdots & x_N^{K-1} & \cdots & x_N^{K-1}y_N^{L-1} \\ 0 & 1 & 2x_N & 3x_N^2 & \cdots & (K-1)x_N^{K-2} & \cdots & (K-1)x_N^{K-2}y_N^{L-1} \end{bmatrix} \quad (16)$$

KL computations may be enough. In Section VI, Lemma 3 presents certain conditions under which the computations received from the workers are useful. If the number of computations provided by all workers satisfies these conditions, then all computations are useful and *KL* computations are enough. Otherwise, we need to discard some computations and since we need to compensate for these discarded computations, the recovery threshold may increase up to the values presented in Theorem 1. In Section VI, the discussion after Lemma 3 explains what kind of computations we discard to guarantee almost regular decodability.

Remark 2: When the conditions of Theorem 1 are satisfied, the bivariate polynomial interpolation problem has a unique solution. The interpolation problem can be solved simply via inverting the interpolation matrix and multiplying it with the vector of responses collected from the workers. This has a complexity of $\mathcal{O}(rs(KL)^2)$. However, such an interpolation strategy may result in large numerical errors; and hence, more sophisticated methods, such as Newton interpolation, may be needed in practice [20], [21]. This aspect is worth investigation, but lies beyond the scope of this work. We leave it as a future research direction.

Selecting Between Computation Orders: When the partitions of B are smaller than those of A, i.e., $c/L < r/K$, under a fixed storage capacity, reducing $m_{A,i}$ by 1 will increase $m_{B,i}$ at least by 1. Since, in this case, the constraints of vertical-type computation orders BPC-VO and BPC-NZO can be satisfied more easily than those of BPC-HO and BPC-ZZO, the schemes having a vertical-type computation order should be chosen. Similarly, when $r/K < c/L$, we should prefer horizontal ordering schemes BPC-HO or BPC-ZZO. Choosing between BPC-HO and BPC-ZZO when $r/K < c/L$, or between BPC-VO and BPC-NNO when $c/L < r/K$, depends on the storage capacity per worker and is discussed further in Section V.

Alternative Formulation of Almost Regular Interpolation Schemes: The reason why we formulate almost regular interpolation schemes in terms of Hermite interpolation throughout the paper is to shorten the proof of Theorem 1. Alternatively, instead of interpolating $A(x)B(y)$ from the evaluations of its derivatives, i.e., Hermite interpolation, almost regular interpolation schemes can also be formulated as the interpolation of $A(x)B(y)$ from its evaluations, as done in B-PROC. Such an approach is equivalent to the Hermite interpolation-based formulation, under the almost regularity condition. We include a more technical discussion about this in the supplementary material. Before reading it, we advise the reader to go through Section VI, since the content in supplementary material is based on the definitions and techniques therein.

V. Numerical Results

In this section, we compare the schemes presented throughout the paper in terms of the average computation time. We only focus on the computation time since the bivariate polynomials to be interpolated in B-PROC, BPC-VO, BPC-HO, BPC-NZO and BPC-ZZO schemes have the same number of coefficients, and thus, the variations in their encoding and decoding times are considered negligible. We also assume that the communication time is negligible. We model the computation speed of the workers by the shifted exponential model [3], [22], which is commonly used in the literature to analyze coded computation schemes. In this model, the probability that a worker finishes at least p computations by time t is $F(p,t) = 1 - e^{-\lambda(\frac{t}{p} - \nu)}$, if $t \geq p\nu$, and 0, otherwise. Thus, the probability of completing exactly p computations by time t is given by $P(p,t) = F(p,t) - F(p+1,t)$ assuming $F(0,t) = 1$, and $F(p_{max}+1,t) = 0$, where p_{max} is the maximum number of computations a worker can complete. In $F(p,t)$, ν is the minimum duration of one computation. The scale parameter λ controls the variance of computation times. The smaller is λ the more variance, and thus more heterogeneous computation speeds among the workers. To cover more heterogeneous cases, we choose $\nu = 0.01$ and $\lambda = 0.1$.

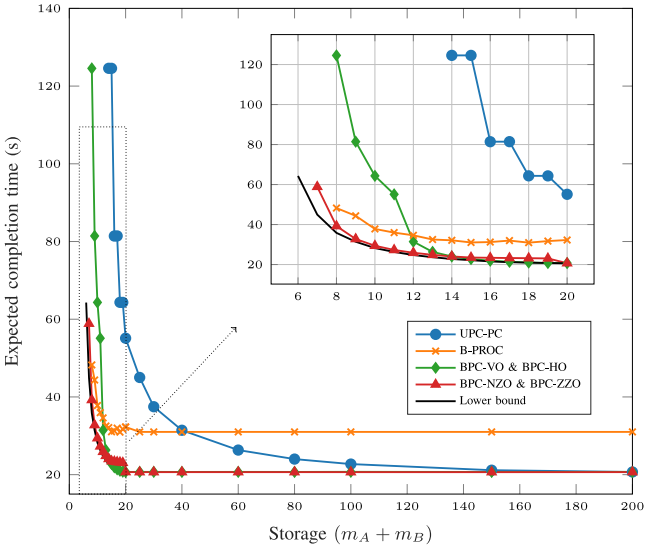

Fig. 5. Average computation times of univariate and bivariate polynomial codes as a function of available storage when partitions of A and B have equal size.

We run Monte Carlo simulations to compute the expected computation time for each scheme under different memory availability. We consider two scenarios in which the sizes of the partitions of A and B are equal, i.e., $\frac{c}{L} = \frac{r}{K}$, and the size of the partitions of B is twice larger than those of A, i.e., $\frac{c}{L} = \frac{2r}{K}$. In both cases, we assume that the workers have the same storage capacity, as required by B-PROC. Thus, $M_{A,i} = M_A$ and $M_{B,i} = M_B$, $\forall i \in [1:N]$. We set $K = L = 10$ and assume $N = 15$. In both scenarios, we set $\mu_B = 5$ and $\mu_A = 5$ for BPC-NZO and BPC-ZZO, respectively. For each memory value, we run 10^4 experiments. The results of the first scenario and the second scenario are given in Fig. 5 and Fig. 6, respectively. For each scheme, the minimum memory required to complete $KL = 100$ computations with $N = 15$ workers is different. Thus, we plot each scheme starting from a different minimum memory value.

Let us first consider the scenario in which the partitions of A and B have equal size. In this case, since we also have $K = L$ and $\mu_A = \mu_B = 5$, there is no difference between BPC-HO and BPC-VO, and also no difference between BPC-NZO and BPC-ZZO. In Fig. 5, we observe that BPC-NZO and BPC-ZZO result in a much lower expected computation time than the other schemes for low storage capacities. Even though we allow partial computations, the univariate polynomial coding, which is UPC-PC, performs far worse than all the others due to inefficient use of the memory resulting in a much less fraction of work done per worker compared to other schemes. In B-PROC, despite the optimality in the memory allocation between $m_{A,i}$ and $m_{B,i}$, we see that the higher number of useless computations aggravates the average computation time. For the same reason, increasing the storage capacity does not improve the average computation time beyond a certain point. While simulating B-PROC, we use a random computation order at the workers, which is reported

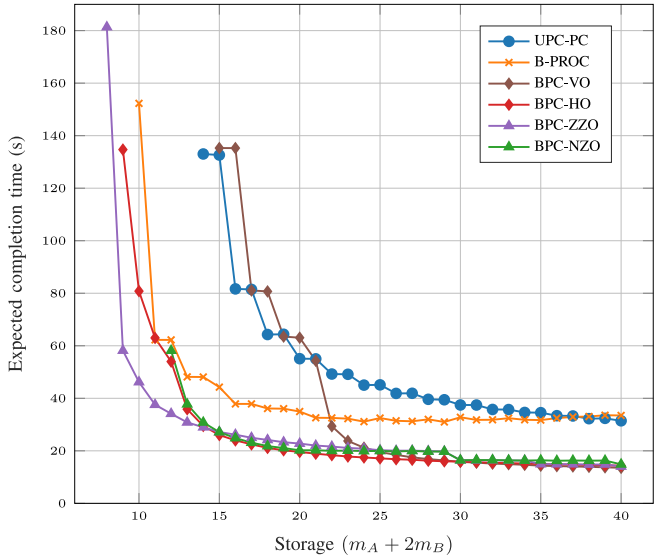

Fig. 6. Average computation times of univariate and bivariate polynomial codes as a function of storage capacity, when partitions of B are twice larger than partitions of A.

to perform well in [15] and stop the computation as soon as the master is able to decode. On the other hand, for BPC-NZO and BPC-ZZO, we consider the worst-case scenario, in which the master starts decoding only after $(\mu_B - 2)(\frac{L}{\mu_B} - 1)$ computations for BPC-NZO or $(\mu_A - 2)(\frac{K}{\mu_A} - 1)$ computations for BPC-ZZO are collected. Thus, we can expect the performance of BPC-NZO and BPC-ZZO to be even better than what we observe in Fig. 5. We also observe that BPC-VO and BPC-HO performs significantly better than B-PROC and UPC-PC for the intermediate and large memory values. For this storage regime, we also observe that BPC-HO and BPC-VO perform slightly better than BPC-ZZO and BPC-NZO due to the first constraint of these schemes. For instance, in BPC-NZO, when $m_{A,i} = K$, we need $m_{B,i}$ to be a multiple of μ_B. Therefore, increasing storage capacity while keeping $m_{A,i} = K$ improves the expected computation time at some specific memory values. This is the reason for the improvement we observe in the expected computation time of the BPC-NZO in Fig. 5 when the storage is 20. On the other hand, in the low storage regime, there is a significant performance degradation of BPC-VO and BPC-HO due to the restrictive constraints of these schemes at small storage values. Since in BPC-NZO and BPC-ZZO, the constraints of BPC-VO and BPC-HO are relaxed especially for small storage values, we observe that BPC-NZO and BPC-ZZO are superior in low storage regimes. To evaluate the performance of our schemes, we also plot a lower bound on the average computation time of any bivariate polynomial-based coding scheme, assuming there are no redundant computations and the memory allocation between $m_{A,i}$ and $m_{B,i}$ is optimal. We observe that the average computation time of the proposed bivariate schemes is quite close to this lower bound especially for the intermediate and high storage regimes. In a low storage regime, we observe that BPC-NZO and BPC-ZZO perform close to the lower bound although for very low values of storage, the gap between BPC-NZO/ZZO and the lower bound increases. This is due to the third constraint of these schemes which forbids optimal memory allocation between $m_{A,i}$ and $m_{B,i}$. This suggests that there might be still room for improvement in the trade-off between the expected computation time and the storage capacities of the workers.

On the other hand, when we consider the scenario in which the partitions of B is twice larger than those of A, in Fig. 6, we observe that neither BPC-NZO and BPC-ZZO nor BPC-VO and BPC-HO are equivalent. Recall the discussion at the end of Section IV-B discussing how we select between computation orders. Since the partitions of B is larger in our case, decreasing $m_{B,i}$ by one may increase $m_{A,i}$ more than one. Therefore, satisfying the constraints of horizontal-type schemes, i.e., BPC-ZZO and BPC-HO, is easier in this case. Therefore, we expect they perform better than the schemes with vertical-type order. We verify this in Fig. 6, in which the performances of BPC-ZZO and BPC-HO are superior especially in the low storage regime. We also observe that for low and intermediate values of the storage, the performance of BPC-VO degrades close to that of UPC-PC. That is because the computations are done column-by-column in BPC-VO, see Fig. 4a, and to assign one more computation twice more storage availability is needed compared to BPC-HO and BPC-ZZO. BPC-NZO suffers from the same problem, but since in the zig-zag order, the constraints are relaxed, i.e., the computation grid is divided into blocks, its performance stays reasonable. We observe that it performs similarly to the BPC-HO in the intermediate and large storage values. Finally, similar to Fig. 5, we observe that for the large storage regime, almost regular schemes perform close to each other and much better than B-PROC and UPC-PC.

VI. Proof of Theorem 1

Without loss of generality, we assume $[1{:}n]$, $1 \leq n \leq N$ is the set of workers which provide at least one computation by the time the master collects sufficient responses to decode AB. Consider the interpolation matrix M as defined in Definition 1. To prove the invertibility of an interpolation matrix M, we use Taylor series expansion of $\det(M)$. Note that $\det(M)$ is a polynomial in the evaluation points $z_i \triangleq (x_i, y_i)$, $i \in [1:n]$. We can write the Taylor series expansion of $\det(M)$ around (x_i, y_i) by taking the evaluation point (x_j, y_j) as the variable, as:

$$\det(M) = \sum_{(\alpha_1,\alpha_2)\in\mathbb{N}^2} \frac{1}{\alpha_1!\alpha_2!}(x_j - x_i)^{\alpha_1}(y_j - y_i)^{\alpha_2} D_{\alpha_1,\alpha_2}(\tilde{Z}), \tag{19}$$

where $\tilde{Z} \triangleq \{(x_k, y_k), k \in [1:n]\} \setminus \{(x_j, y_j)\}$, and

$$D_{\alpha_1,\alpha_2}(\tilde{Z}) \triangleq \frac{\partial^{\alpha_1+\alpha_2}}{\partial x_j^{\alpha_1} \partial y_j^{\alpha_2}} \det(M)\Big|_{x_j=x_i, y_j=y_i}. \tag{20}$$

We call (x_i, y_i) the **pivot** node and (x_j, y_j) the **variable** node in this expansion. If the monomials in the set $\{x^{\alpha_1}y^{\alpha_2} \mid (\alpha_1, \alpha_2) \in \mathbb{N}^2\}$ cannot be written as a linear combination of the other monomials in the set, then, they are said to be *linearly independent*. In this sense, the monomials $(x_j - x_i)^{\alpha_1}(y_j - y_i)^{\alpha_2}$ in (19) are linearly independent for different (α_1, α_2) pairs, as long as, there is no dependence between x_i, x_j and y_i, y_j. Consequently, $\det(M) = 0$ for all values of $(x_j, y_j) \in \mathbb{R}^2$, if and only if $D_{\alpha_1,\alpha_2}(\tilde{Z}) = 0, \forall(\alpha_1, \alpha_2) \in \mathbb{N}^2$. That is, to show that M is non-singular, it suffices to show that there exists an (α_1, α_2) pair such that $D_{\alpha_1,\alpha_2}(\tilde{Z})$ is nonzero.

Let us choose some (α_1, α_2) pair, and analyse $D_{\alpha_1,\alpha_2}(\tilde{Z})$. Notice that, $D_{\alpha_1,\alpha_2}(\tilde{Z})$ is a polynomial in the evaluation points, now, in $\tilde{Z}$. Specifically, it does not depend on x_j and y_j since the derivatives were taken with respect to these variables, and then evaluated at $x_j = x_i, y_j = y_i$. We call this procedure the

coalescence of the evaluation points (x_i, y_i) and (x_j, y_j) into (x_i, y_i). Next, to show $D_{\alpha_1,\alpha_2}(\tilde{Z}) \neq 0$, we do a new coalescence, i.e., we write the Taylor series expansion of $D_{\alpha_1,\alpha_2}(\tilde{Z})$ on a new variable point, choose a new (α_1, α_2) pair, and coalescence them into (x_i, y_i). Our proof technique is based on such recursive Taylor series expansions until all evaluation points are coalesced into one. We will present a technique to choose (α_1, α_2) pairs at each step, which guarantees to obtain a non-zero polynomial at the final coalescence step.

In the following, we first present some preliminaries which we will need while presenting our technique for choosing (α_1, α_2) at each step.

A. Preliminaries

In order to choose an (α_1, α_2) pair at each step, we will need to analyze $D_{\alpha_1,\alpha_2}(\tilde{Z})$. Since $D_{\alpha_1,\alpha_2}(\tilde{Z})$ is derived from the Taylor series expansion of a determinant, in some cases, we can write it, again, in terms of the determinants of other matrices, which turns out to be more insightful than using its polynomial form. Before showing this, we introduce the notions of **derivative set** and **shift**, which will be useful in the rest of the proof.

Definition 4: Associated to every evaluation point $z_i \triangleq (x_i, y_i), i \in [1 : n]$, there may be one or more rows in M each corresponding to a different derivative order of $A(x)B(y)$ evaluated at z_i. We define the **derivative set**, $\mathcal{U}_{z_i,M}$, of node z_i as the multiset[2] of derivative orders associated to z_i in M, i.e., we say $(d_x, d_y) \in \mathcal{U}_{z_i,M}$, if M has a row corresponding the evaluation $\partial_{d_x}A(x_i)\partial_{d_y}B(y_i)$ or equivalently the master received the evaluation $\partial_{d_x}A(x_i)\partial_{d_y}B(y_i)$ from worker i.

Definition 5: Let $M \in \mathbb{R}^{KL\times KL}$ be an interpolation matrix such that at least one of its rows depends on (x_j, y_j), and let r_i denote the i^{th} row in M. We define a **simple shift**[3] as

$$\partial_{i,x_j} M \triangleq \left[r_1^T, \ldots, \frac{\partial}{\partial x_j} r_i^T, \ldots, r_{KL}^T\right]^T$$

and

$$\partial_{i,y_j} M \triangleq \left[r_1^T, \ldots, \frac{\partial}{\partial y_j} r_i^T, \ldots, r_{KL}^T\right]^T.$$

Assume that the i^{th} row of M corresponds to $\partial_{d_{i,x}}A(x_j)\partial_{d_{i,y}}B(y_j)$. Then, the derivative sets of node z_i associated to matrices $\partial_{i,x_j}M$ and $\partial_{i,y_j}M$ are shifted versions of the ones associated to M, in the sense that, $\mathcal{U}_{z_j,\partial_{i,x_j}M} = \{(d_{i,x}+1, d_{i,y})\} \cup \mathcal{U}_{z_j,M} \setminus \{(d_{i,x}, d_{i,y})\}$ and $\mathcal{U}_{z_j,\partial_{i,y_j}M} = \{(d_{i,x}, d_{i,y}+1)\} \cup \mathcal{U}_{z_j,M} \setminus \{(d_{i,x}, d_{i,y})\}$. Note that if the multiplicity of any element in a derivative set is greater than one, then the corresponding interpolation matrix has at least two identical rows making the matrix singular. ∂_{i,x_j} is called a **regular** simple shift, if all elements in $\mathcal{U}_{z_j,\partial_{i,x_j}M}$ have a multiplicity of one. Similarly, ∂_{i,y_j} is a regular simple shift if all elements in $\mathcal{U}_{z_j,\partial_{i,y_j}M}$ have a multiplicity of one. Finally, for the **composition of simple shifts**, we introduce the notation $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M$, where the i^{th} entries in $\boldsymbol{k}$ and $\boldsymbol{l}$ are the total order of the derivatives taken on the i^{th} row of M with respect to x_j and y_j, respectively. Thanks to the commutative property of the derivative, given a pair $(\boldsymbol{k}, \boldsymbol{l})$ one can compute $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M$ by taking derivatives from any row, at any order, until completing the derivative orders specified in $(\boldsymbol{k}, \boldsymbol{l})$. We refer to each of these possible choices as a derivative path, and define those paths that only involve regular simple shifts, i.e., after each derivative there are not two equal rows, as **regular derivative paths**. The number of regular derivative paths is denoted by $C_{\boldsymbol{k},\boldsymbol{l}}(M)$.

[2]Here, we use multiset instead of set as we allow multiple instances for each of its elements. The number of instances of a given element is called the multiplicity of that element in the multiset.

[3]The term **shift** to refer to derivatives of interpolation matrices highlight the fact that derivatives applied to interpolation matrices correspond to shifts in their derivative sets when depicted in the derivative order space, as shown in Fig. 7.

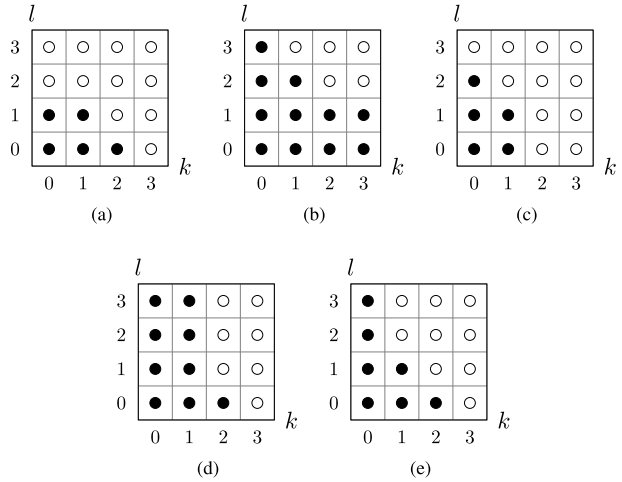

Fig. 7. Example sets of N-zig-zag ordered (a,b), Z-zig-zag ordered (c,d) and neither N-zig-zag ordered nor Z-zig-zag ordered (e).

The following lemma provides an expression for the derivatives of the determinant of an interpolation matrix in terms of a weighted sum of determinants of other interpolation matrices.

Lemma 1: Let $\boldsymbol{k} \in [0 : K-1]^{KL}$, $\boldsymbol{l} \in [0{:}L-1]^{KL}$ and $\alpha_1 = \sum_{i=1}^{KL} \boldsymbol{k}(i)$ and $\alpha_2 = \sum_{i=1}^{KL} \boldsymbol{l}(i)$. Then, we have

$$\frac{\partial^{\alpha_1+\alpha_2}}{\partial x_j^{\alpha_1} \partial y_j^{\alpha_2}} \det(M)\bigg|_{x_j=x_i, y_j=y_i} = \sum_{(\boldsymbol{k},\boldsymbol{l})\in\mathcal{R}_M(\alpha_1,\alpha_2)} C_{\boldsymbol{k},\boldsymbol{l}}(M) \det\left(\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M\right)\bigg|_{x_j=x_i, y_j=y_i} \quad (21)$$

where $\mathcal{R}_M(\alpha_1, \alpha_2)$ is the set of $(\boldsymbol{k}, \boldsymbol{l})$ pairs satisfying $C_{\boldsymbol{k},\boldsymbol{l}}(M) \neq 0$, i.e., there is at least one derivative path for which $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j}$ can be applied by using only regular simple shifts.

We defer the Proof of Lemma 1 to Appendix A.

B. Choosing an (α_1, α_2) pair in a coalescence

Recall that our objective is to find an (α_1, α_2) pair for each step in the successive coalescence procedure. Lemma 1 is an important step in this direction as it allows us to express $D_{\alpha_1,\alpha_2}(\tilde{Z})$ in terms of a sum of determinants of interpolation matrices. However, it still does not provide us a clear clue on how to choose (α_1, α_2), so that $D_{\alpha_1,\alpha_2}(\tilde{Z}) \neq 0$. Next, we define a structure over the derivative sets of the interpolation matrices, similar to the ones defined for the computation orders in Section IV-B, which will eventually help us to define the **quasi-unique shift** pairs (α_1, α_2), which satisfy the conditions needed for completing a coalescence procedure successfully.

Definition 6: A derivative set $\mathcal{U}_{z,M}$ is said to be **N-zig-zag ordered** with parameter μ_B if $(i, j) \in \mathcal{U}_{z,M}$ implies that all the derivatives with order (k, l) such that $S_{\mathcal{N}}(1, 1) \geq S_{\mathcal{N}}(k+1, l+1) \geq S_{\mathcal{N}}(i+1, j+1)$ are also in $\mathcal{U}_{z,M}$, where $S_{\mathcal{N}}(k, l) = (K-1)\mu_B(\lceil \frac{l}{\mu_B} \rceil - 1) + \mu_B(k-1) + l$ as in Section IV-B.

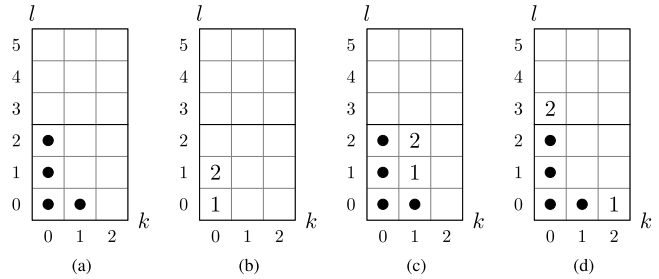

Fig. 8. Depictions of derivative sets in Example 4.

Similarly, $\mathcal{U}_{z,M}$ is **Z-zig-zag ordered** with parameter μ_A if $(i,j) \in \mathcal{U}_{z,M}$ implies that all (k,l) such that $S_{\mathcal{Z}}(1,1) \geq S_{\mathcal{Z}}(k+1, l+1) \geq S_{\mathcal{Z}}(i+1, j+1)$ are in $\mathcal{U}_{z,M}$, where $S_{\mathcal{Z}}(k,l) = (L-1)\mu_A(\lceil \frac{k}{\mu_A} \rceil - 1) + \mu_A(l-1) + k$ as in Section IV-B.

Example 3: Consider the derivative sets with $K = L = 4$. The derivative sets illustrated in Fig. 7a and Fig. 7b are N-zig-zag ordered for $\mu_B = 2$, and the sets in Fig. 7c and Fig. 7d are Z-zig-zag ordered for $\mu_A = 2$. The set in Fig. 7e is neither N-zig-zag nor Z-zig-zag ordered.

Hereafter, for brevity, we stick to the N-zig-zag order. This will allow us to prove the part *a* of Theorem 1. The proof of part *b* follows similarly using the Z-zig-zag order instead, and thus we omit it here.

Definition 7: Consider (x_j, y_j) is the variable node and (x_i, y_i) is the pivot node. Suppose that $\mathcal{U}_{z_j,M}$ obeys the N-zig-zag order, and define $M^* \triangleq \nabla^{x_j,y_j}_{\boldsymbol{k}^*,\boldsymbol{l}^*} M|_{x_j=x_i,y_j=y_i}$. If there is only one $(\boldsymbol{k}^*, \boldsymbol{l}^*) \in \mathcal{R}_M(\alpha_1,\alpha_2)$ such that $\mathcal{U}_{z_j,M^*}$ obeys the N-zig-zag order, then (α_1,α_2) is called **quasi-unique**.

Example 4: Let $(\alpha_1,\alpha_2) = (2,2)$ and $K = 3, L = 6$, $\mu_B = 3$. We assume the derivative sets of the pivot and variable nodes are as depicted in the derivative order space in Fig. 8a and Fig. 8b, respectively. We observe that the interpolation matrix has two rows that depend on the variable node and four rows that depend on the pivot node. Without loss of generality, we assume rows 1 and 2 depend on the variable node. In this example, we stick to the definition in (20), i.e., we take derivatives of the interpolation matrix first with respect to the y component of the variable node and then the x component. Therefore, $\mathcal{R}_M(2,2) = \{([1,1,\mathbf{0}_{KL-2}],[1,1,\mathbf{0}_{KL-2}]),([2,0,\mathbf{0}_{KL-2}],[0,2,\mathbf{0}_{KL-2}])\}$, where $\mathbf{0}_{KL-2}$ is the all-zero vector with dimension $KL-2$. Note that there is no other $(\boldsymbol{k},\boldsymbol{l})$ pair such that $\nabla^{x_j,y_j}_{\boldsymbol{k},\boldsymbol{l}}$ can be applied by using only regular simple shifts. When we apply $\nabla^{x_j,y_j}_{\boldsymbol{k},\boldsymbol{l}}$ with $(\boldsymbol{k},\boldsymbol{l}) = ([1,1,\mathbf{0}_{KL-2}],[1,1,\mathbf{0}_{KL-2}])$, we obtain a derivative set as depicted in Fig. 8c, and obtain the one in Fig. 8d with $(\boldsymbol{k},\boldsymbol{l}) = ([2,0,\mathbf{0}_{KL-2}],[0,2,\mathbf{0}_{KL-2}])$. Note that the derivative set in Fig. 8c obeys the N-zig-zag order while the one in Fig. 8d does not. Since there is only one $(\boldsymbol{k},\boldsymbol{l})$ pair resulting in an N-zig-zag ordered derivative set, (α_1,α_2) is quasi-unique.

Next we describe, in detail, the first two iterations of the recursive coalescence procedure, and then generalize the result to any iteration. Without loss of generality, we choose (x_n, y_n) as the pivot node for all the coalescences in the recursion, and coalesce it with the variable node z_i in the i^{th} coalescence from $i = 1$ to $n-1$. Let us define the set of remaining nodes before applying the j^{th} coalescence as $Z_j \triangleq \{(x_i, y_i) \mid i \in [j:n]\}$. For the first coalescence, let $M_1 = M$, and suppose we find a quasi-unique shift for order (α_1^*, α_2^*). We denote by $M_2 \triangleq C_{\boldsymbol{k}^*,\boldsymbol{l}^*}(M_1)\nabla^{x_1,y_1}_{\boldsymbol{k}^*,\boldsymbol{l}^*} M_1|_{x_1=x_n,y_1=y_n}$ the unique matrix such that $\mathcal{U}_{z_n,M_2}$ satisfies the N-zig-zag order, and define the set of matrices containing the rest of the interpolation matrices as

$$\Phi_2 \triangleq \Big\{ C_{\boldsymbol{k},\boldsymbol{l}}(M_1)\nabla^{x_1,y_1}_{\boldsymbol{k},\boldsymbol{l}} M_1\big|_{x_1=x_n,y_1=y_n} \; |(\boldsymbol{k},\boldsymbol{l}) \in \mathcal{R}_{M_1}(\alpha_1^*,\alpha_2^*) \setminus (\boldsymbol{k}^*,\boldsymbol{l}^*) \Big\}. \tag{22}$$

Then, from (21), we can write

$$\begin{aligned} D_2(Z_2) &= \frac{\partial^{\alpha_1^*+\alpha_2^*}}{\partial x_1^{\alpha_1^*}\partial y_1^{\alpha_2^*}} \det(M_1)\Big|_{x_1=x_n,y_1=y_n} \\ &= \det(M_2) + \sum_{\bar{M}\in\Phi_2} \det(\bar{M}). \end{aligned} \tag{23}$$

For the second coalescence, taking (x_n, y_n) as the pivot node and (x_2, y_2) as the variable node, we write the Taylor series expansion of $D_2(Z_2)$ as

$$D_2(Z_2) = \sum_{(\alpha_1,\alpha_2)\in\mathbb{N}^2} \frac{1}{\alpha_1!\alpha_2!}(x_2-x_n)^{\alpha_1}(y_2-y_n)^{\alpha_2} D_{\alpha_1,\alpha_2}(Z_3) \tag{24}$$

where

$$\begin{aligned} D_{\alpha_1,\alpha_2}(Z_3) = &\frac{\partial^{\alpha_1+\alpha_2}}{\partial x_2^{\alpha_1}\partial y_2^{\alpha_2}} \det(M_2)\Big|_{x_2=x_n,y_2=y_n} \\ &+ \sum_{\bar{M}\in\Phi_2} \frac{\partial^{\alpha_1+\alpha_2}}{\partial x_2^{\alpha_1}\partial y_2^{\alpha_2}} \det(\bar{M})\Big|_{x_2=x_n,y_2=y_n}. \end{aligned} \tag{25}$$

Next, we apply (21) to (25). This time, we find a quasi-unique shift (α_1^*, α_2^*) by only considering matrix M_2. Note that, the (α_1^*, α_2^*) pair is different for each recursion but for a clearer notation, we omit the recursion index. Since the choice of (α_1^*, α_2^*) only considers M_2, it does not imply the existence of quasi-unique shifts for all the other matrices in Φ_2. We denote by $M_3 \triangleq C_{\boldsymbol{k}^*,\boldsymbol{l}^*}(M_2)\nabla^{x_1,y_1}_{\boldsymbol{k}^*,\boldsymbol{l}^*} M_2|_{x_2=x_n,y_2=y_n}$ the unique matrix satisfying that $\mathcal{U}_{z_n,M_3}$ follows the N-zig-zag order, and define the set of matrices containing the rest of weighted interpolation matrices, originated from M_2 or from $\bar{M} \in \Phi_2$, as

$$\begin{aligned} \Phi_3 \triangleq &\Big\{ C_{\boldsymbol{k},\boldsymbol{l}}(M_2)\nabla^{x_2,y_2}_{\boldsymbol{k},\boldsymbol{l}} M_2\big|_{x_2=x_n,y_2=y_n} \\ &\qquad |(\boldsymbol{k},\boldsymbol{l}) \in \mathcal{R}_{M_2}(\alpha_1^*,\alpha_2^*) \setminus (\boldsymbol{k}^*,\boldsymbol{l}^*) \Big\} \\ &\cup \Big\{ C_{\boldsymbol{k},\boldsymbol{l}}(\bar{M})\nabla^{x_2,y_2}_{\boldsymbol{k},\boldsymbol{l}} \bar{M}\big|_{x_2=x_n,y_2=y_n} \\ &\qquad |(\boldsymbol{k},\boldsymbol{l}) \in \mathcal{R}_{M_2}(\alpha_1^*,\alpha_2^*), \bar{M} \in \Phi_2 \Big\}. \end{aligned} \tag{26}$$

Then, we can write

$$D_3(Z_3) = D_{\alpha_1^*,\alpha_2^*}(Z_3) = \det(M_3) + \sum_{\bar{M}\in\Phi_3} \det(\bar{M}). \tag{27}$$

We follow the same procedure until all nodes are coalesced with the pivot node and we reach $D_n(Z_n)$. In general, the expressions in this procedure are defined recursively as follows.

$$M_{i+1} \triangleq C_{\boldsymbol{k}^*,\boldsymbol{l}^*}(M_i)\nabla^{x_i,y_i}_{\boldsymbol{k}^*,\boldsymbol{l}^*} M_i\big|_{x_i=x_n,y_i=y_n}, i \in [1:n-1] \tag{28}$$

$$D_i(Z_i) \triangleq D_{\alpha_1^*,\alpha_2^*}(Z_i) = \det(M_i) + \sum_{\bar{M}\in\Phi_i} \det(\bar{M}). \tag{29}$$

$$D_i(Z_i) = \sum_{(\alpha_1,\alpha_2)\in\mathbb{N}^2} \frac{1}{\alpha_1!\alpha_2!}(x_i-x_n)^{\alpha_1}(y_i-y_n)^{\alpha_2} D_{\alpha_1,\alpha_2}(Z_{i+1}). \tag{30}$$

$$\Phi_{i+1} \triangleq \Big\{ C_{\boldsymbol{k},\boldsymbol{l}}(M_i) \nabla^{x_i,y_i}_{\boldsymbol{k},\boldsymbol{l}} M_i \big|_{x_i=x_n, y_i=y_n} \quad |(\boldsymbol{k},\boldsymbol{l}) \in \mathcal{R}_{M_i}(\alpha_1^*, \alpha_2^*) \setminus (\boldsymbol{k}^*, \boldsymbol{l}^*) \Big\} \cup \Big\{ C_{\boldsymbol{k},\boldsymbol{l}}(\bar{M}) \nabla^{x_i,y_i}_{\boldsymbol{k},\boldsymbol{l}} \bar{M} \big|_{x_i=x_n, y_i=y_n} \quad |(\boldsymbol{k},\boldsymbol{l}) \in \mathcal{R}_{M_i}(\alpha_1^*, \alpha_2^*), \bar{M} \in \Phi_i \Big\}. \tag{31}$$

Lemma 2: Consider the recursive Taylor series expansion procedure on a fixed pivot, (x_n, y_n). If, for every step $i \in [1:n]$, we can find a quasi-unique shift (α_1^*, α_2^*) for M_i in (28), then

$$D_n(Z_n) = \det(M_n), \tag{32}$$

where M_n depends only on $Z_n = \{(x_n, y_n)\}$. Therefore, the associated interpolation matrix M_n, and hence, M_1 are invertible for almost all choices of evaluation points.

The proof of Lemma 2 is given in Appendix B.

In the next lemma, we present a set of situations for which a quasi-unique shift exits in a coalescence step between a pivot node and a variable node. These are not the only situations for which quasi-unique shifts exit but are sufficient to derive the recovery threshold presented in Theorem 1, as we show in the next subsection.

Lemma 3: Assume that in the i^{th} coalescence step we have the variable node $z_i = (x_i, y_i)$ and the pivot node $z_n = (x_n, y_n)$. Define $r_f \triangleq |\mathcal{U}_{z_i,M_i}|(\mod \mu_B)$ and $l_e \triangleq \mu_B - |\mathcal{U}_{z_n,M_i}| (\mod \mu_B)$. That is, when depicted in the derivative order space, r_f is the number of elements in the rightmost partially-occupied column of the derivative set of the variable node, and l_e is the number of empty places in the rightmost partially-occupied column of the derivative set of the pivot node. Then, if $|\mathcal{U}_{z_i,M_i}| + |\mathcal{U}_{z_n,M_i}| > \mu_B K$ and one of the following conditions is satisfied:

1) $r_f = 0$,
2) $r_f = l_e$,
3) $l_e = 0$,

or

4) $|\mathcal{U}_{z_i,M_i}| + |\mathcal{U}_{z_n,M_i}| \leq \mu_B K$,

then there exists a quasi-unique shift for the coalescence of these nodes.

The proof of Lemma 3 is given in Appendix C.

C. Derivation of the Recovery Threshold Expression

The existence of a quasi-unique shift depends on the joint structure of the derivative sets of the pivot and the variable nodes. If the derivative sets of the pivot node and the variable node satisfy the conditions in Lemma 3, then, in a coalescence step, i.e., recursive Taylor series expansion, it is possible to find a quasi-unique shift for this recursive step and we can proceed to the next recursion. Otherwise, by simply ignoring specific computations provided by the worker whose evaluation point corresponds to the variable node under consideration, we can have the structure of the remaining computations satisfy the conditions in Lemma 3. This adds an overhead of ignored computations to the recovery threshold expression. In the following lemma, we provide an upper bound on the total number of computations we may need to ignore throughout the whole recursion process by analysing the worst-case scenario.

Lemma 4: Assume that the conditions of Lemma 3 hold in none of the coalescences in the recursive Taylor series expansion process. Then, in the worst case, by ignoring at most $(\mu_B - 2)(\frac{L}{\mu_B} - 1)$ computations throughout all the recursion steps suffices to guarantee decodability for almost all choices of evaluation points.

Proof: Assume that none of the conditions of Lemma 3 hold. If $r_f > l_e$, we can satisfy condition 2, i.e., $r_f = l_e$, in Lemma 3 by ignoring $r_f - l_e$ computations received from the worker whose evaluation point is the variable node. Thus, in the worst case, we ignore $\max(r_f - l_e) = (\mu_B - 1) - 1 = \mu_B - 2$ computations. Note that the minimum value of l_e is 1. Otherwise, condition 3 in Lemma 3 would be satisfied. Moreover, the maximum value of r_f is $\mu_B - 1$. Otherwise, condition 1 in Lemma 3 would be satisfied. On the other hand, if $r_f < l_e$, we can ignore r_f computations and satisfy the condition 1 in Lemma 3. Since $r_f < l_e < \mu_B$, in the worst case, we need to ignore $\max(r_f) = \mu_B - 2$ computations. Thus, in either case, the maximum number of computations we ignore is $\mu_B - 2$. Observe that generating a new block in the derivative order space as a result of a coalescence and not satisfying any of the conditions in Lemma 3 are possible only if when $|\mathcal{U}_{z_i,M_i}| + |\mathcal{U}_{z_n,M_i}| > \mu_B K$. Given that there are L/μ_B blocks in the whole derivative order space, the maximum number of coalescences for which $|\mathcal{U}_{z_i,M_i}| + |\mathcal{U}_{z_n,M_i}| > \mu_B K$ is at most $(L/\mu_B - 1)$. Thus, in the worst-case, the total number of ignored computations is $(\mu_B - 2)(\frac{L}{\mu_B} - 1)$. ∎

Since the polynomial we need to interpolate, $A(x)B(y)$, has KL coefficients, in the worst case the recovery threshold becomes $KL + (\mu_B - 2)(\frac{L}{\mu_B} - 1)$. Since this number guarantees the existence of a quasi-unique shift in every recursive Taylor series expansion, by Lemma 2, we can conclude that our original interpolation matrix is invertible for almost all choices of the evaluation points. This completes the proof of Theorem 1.

VII. Conclusion

In this work, we studied the memory-efficient exploitation of stragglers in distributed matrix multiplication where workers are allowed to have heterogeneous computation and storage capacities. We proposed bivariate polynomial coding schemes allowing efficient use of workers' memories. Bivariate polynomial coding poses the problem of invertibility of an interpolation matrix, which is highly non-trivial, unlike univariate polynomial codes. We first proposed a coding scheme based on the fact that the interpolation matrix of bivariate interpolation is always invertible if the evaluation points form a rectangular grid. However, in this scheme, some computations received by the master may not be useful since the information they provide is already obtained from previous responses. In order to tackle this problem, we showed that as long as the workers follow a specific computation order, the interpolation matrix is invertible for almost every choice of the interpolation points. Based on this, we proposed BPC-VO and BPC-HO solving the problem of redundant computations. However, the constraints imposed by the computation orders in BPC-VO and BPC-HO harm the average computation time when the storage capacities of the workers are limited. To overcome this, in BPC-NZO and BPC-ZZO, we relax these constraints by allowing a few redundant computations, which are still much less than those of B-PROC. The ability of the proposed schemes to exploit the workers' storage capacities is close to the optimal. For different storage capacities, we numerically showed that in terms of the average computation time, the proposed schemes in the paper outperform existing schemes in the literature.

The proof of the almost regularity of bivariate polynomial coding schemes is itself a theoretically interesting one, and it

may guide proofs of other multivariate interpolation schemes for distributed matrix multiplication in more general situations. Another interesting line of work is the application of bivariate polynomial coding to private matrix multiplication.

APPENDIX A
PROOF OF LEMMA 1

Proof: Given an interpolation matrix M, we first prove

$$\frac{\partial}{\partial x_j}\det(M) = \sum_{i=1}^{KL}\det\big(\partial_{i,x_j}M\big), \tag{33}$$

and

$$\frac{\partial}{\partial y_j}\det(M) = \sum_{i=1}^{KL}\det\big(\partial_{i,y_j}M\big). \tag{34}$$

They follow directly from the chain rule. Let $m_{i,j}$'s denote the elements of M, and S_{KL} the set of all permutations of the columns of M. We use the fact $\det(M) = \sum_{\pi\in S_{KL}}\operatorname{sgn}(\pi)\prod_{i=1}^{KL}m_{i,\pi(i)}$ [23, Definition 7.4], where π is a permutation, and $\operatorname{sgn}(\pi)$ is its parity. Then,

$$\begin{aligned}\frac{\partial}{\partial x_j}\det(M) &= \sum_{\pi\in S_{KL}}\operatorname{sgn}(\pi)\frac{\partial}{\partial x_j}\prod_{i=1}^{KL}m_{i,\pi(i)}\\ &= \sum_{\pi\in S_{KL}}\operatorname{sgn}(\pi)\sum_{i=1}^{KL}\left(\frac{\partial}{\partial x_j}m_{i,\pi(i)}\right)\prod_{j\in[1:KL]\setminus\{i\}}m_{j,\pi(j)}\\ &= \sum_{i=1}^{KL}\sum_{\pi\in S_n}\operatorname{sgn}(\pi)\left(\frac{\partial}{\partial x_j}m_{i,\pi(i)}\right)\prod_{j\in[1:KL]\setminus\{i\}}m_{j,\pi(j)}\\ &= \sum_{i=1}^{KL}\det\big(\partial_{i,x_j}M\big).\end{aligned} \tag{35}$$

The proof of (34) can be done similarly. Next, consider part of a derivative path $s \triangleq \partial_{i_l,y_j}\cdots\partial_{i_2,y_j}\partial_{i_1,y_j}$ of length $l < \alpha_2$ such that it has two identical rows or at least one zero row, resulting in $\det(sM) = 0$. Now let us consider the other sequences having s as the suffix. Applying (34) $m \le \alpha_2 - l$ times,

$$\frac{\partial}{\partial y_j^m}\det(sM) = \sum_{i_{l+m}=1}^{KL}\cdots\sum_{i_{l+1}=1}^{KL}\det\big(\partial_{i_{l+m},y_j}\cdots\partial_{i_{l+1},y_j}sM\big). \tag{36}$$

However, $\frac{\partial}{\partial y_j^m}\det(sM) = 0$ since $\det(sM) = 0$. The same applies to x directional derivatives. That is, while taking the derivatives of $\det(M)$, i.e., applying $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j}$, if we encounter a sub-sequence s such that $\det(sM) = 0$, then the sum of determinants of all matrices having sM as suffix, i.e., $\partial_{i_{l+m},y_j}\cdots\partial_{i_{l+1},y_j}sM$, adds up to zero. Thus, only the sequences in which all simple shifts are regular contribute to (21), while applying $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j}$. Given a $(\boldsymbol{k},\boldsymbol{l})$ pair, if $C_{\boldsymbol{k},\boldsymbol{l}}$ denotes the number of sequences composed of only regular simple shifts, we obtain (21). ∎

APPENDIX B
PROOF OF LEMMA 2

We first present another lemma that will be useful in the proof.

Lemma 5: No derivative set corresponding to the evaluation points z_i in $\Phi_{i+1}, \forall i \in [1{:}N-1]$, defined in (31) obeys N-zig-zag order.

Proof: From the definition of quasi-unique shift, it is clear that no elements of the first set in (31) obeys N-zig-zag order. To show the same for the second set, consider the elements of the variable node at the coalescence step i. (α_1^*, α_2^*) is chosen such that there is only one $(\boldsymbol{k}^*, \boldsymbol{l}^*)$ such that when the elements of the variable node are shifted according to $(\boldsymbol{k}^*, \boldsymbol{l}^*)$, and evaluated at (x_n, y_n), the resulting derivative set obeys the N-zig-zag order. Assume now that no element in Φ_i obeys the N-zig-zag order. Take any $\bar{M} \in \Phi_i$ and apply $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_i,y_i}\bar{M}$ for some $(\boldsymbol{k},\boldsymbol{l})$. If $(\boldsymbol{k},\boldsymbol{l}) \neq (\boldsymbol{k}^*, \boldsymbol{l}^*)$, then at least one of the elements of the variable node will be placed to a location whose priority score is larger than those of all the locations to which the elements of the variable node would be placed if $(\boldsymbol{k}^*, \boldsymbol{l}^*)$ were applied. This is because $\exists j$ such that $\boldsymbol{k}(j) > \boldsymbol{k}^*$ or $\exists j$ such that $\boldsymbol{l}(j) > \boldsymbol{l}^*(j)$. In $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_i,y_i}\bar{M}|_{x_i=x_n,y_i=y_n}$, if the derivative set of (x_n, y_n) obeyed the N-zig-zag order, the variable node would have to have more elements than it originally had since the largest priority score whose corresponding location is occupied is larger for $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_i,y_i}\bar{M}|_{x_i=x_n,y_i=y_n}$ than $\nabla_{\boldsymbol{k}^*,\boldsymbol{l}^*}^{x_i,y_i}M_i|_{x_i=x_n,y_i=y_n}$. Therefore, it is not possible that the derivative set of the pivot node for $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_i,y_i}\bar{M}|_{x_i=x_n,y_i=y_n}$ obeys the N-zig-zag order when $(\boldsymbol{k},\boldsymbol{l}) \neq (\boldsymbol{k}^*, \boldsymbol{l}^*)$. On the other hand, if $(\boldsymbol{k},\boldsymbol{l}) = (\boldsymbol{k}^*, \boldsymbol{l}^*)$, since we assume that no element of Φ_i obeys the N-zig-zag order, the derivative set of the pivot node for $\nabla_{\boldsymbol{k}^*,\boldsymbol{l}^*}^{x_i,y_i}\bar{M}|_{x_i=x_n,y_i=y_n}$ does not obey the N-zig-zag order. Since we know that no element in Φ_2 obeys the N-zig-zag order by definition, by induction, we conclude that none of the elements in $\Phi_{i+1}, \forall i \in [2:N]$ obeys the N-zig-zag order. ∎

The rows of M_n only depend on the pivot node (x_n, y_n), and thus, the derivative set associated to (x_n, y_n) has KL elements and satisfies the N-zig-zag order. Similarly, for all matrices in Φ_n, the derivative sets of the pivot node have KL elements. However, as Lemma 5 suggests, in this case, no elements of Φ_n satisfies the N-zig-zag order. This implies that all matrices in Φ_n have at least one duplicate row, or a zero row. Therefore, $\sum_{\bar{M}\in\Phi_n}\det(\bar{M})|_{x_{n-1}=x_n,y_{n-1}=y_n} = 0$. This proves (32). The proof of $\det(M_n) \neq 0$ follows, directly, from the fact that, for M_n, the derivative set of the pivot node obeys the N-zig-zag order. This means that each row of M_n corresponds to $\partial_k A(x_n)\partial_l B(y_n)$, $\forall k \in [0:K-1]$, $\forall l \in [0:L-1]$. Therefore, M_n can be written as an upper triangular matrix, and therefore, invertible, implying $D_n(Z_n) \neq 0$. Remember that $D_{i+1}(Z_{i+1}) \neq 0$ implies $D_i(Z_i) \neq 0$ for all $i \in [0:n-1]$ due to the linear independence between $(x_i - x_n)^{\alpha_1}(y_i - y_n)^{\alpha_2}$ for different (α_1, α_2) pairs. Thus, $D_n(Z_n) \neq 0$ implies $D_1(Z_1) \neq 0$ recursively, and thus, M_1 is invertible. This proves the claim of the lemma. ∎

APPENDIX C
PROOF OF LEMMA 3

First, note that the existence of a quasi-unique shift is only related to the structure of the uppermost blocks of the pivot and variable nodes' derivative sets. Therefore, even if the derivative sets of the pivot and variable nodes occupy more than one block, in the derivative order space, it is sufficient to consider only the uppermost blocks since the fully occupied blocks can be handled only by additional y-directional derivatives. Thus, we proceed as if there exist only the uppermost blocks of the derivative sets of the pivot and variable nodes.

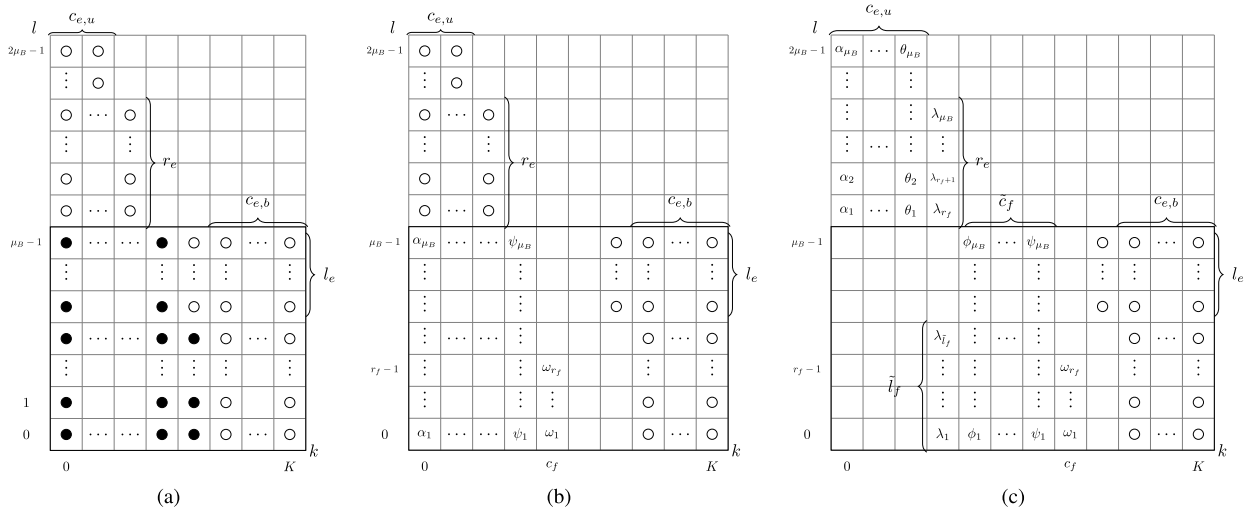

Fig. 9. Visualization of the derivative sets of the pivot and variable nodes.

Our proof is based on determining some sufficient conditions for the existence of a quasi-unique shift, which will reduce to the conditions claimed in the lemma. We first state our problem visually in the derivative order space in terms of the derivative sets and the derivatives of the evaluations, then we find the sufficient conditions on this visual problem statement.

In this part of the proof, we take all the y-directional derivatives before the x-directional ones. We depict the elements in the derivative set of the pivot node, $z_n = (x_n, y_n)$, in the derivative order space by filled circles in Fig. 9a. Since the sum of the elements in the derivative sets of the pivot and variable nodes is larger than the size of one block, i.e., $|\mathcal{U}_{z_n,M_i}| + |\mathcal{U}_{z_i,M_i}| > \mu_B K$, the coalescence generates a new block. The unfilled circles in Fig. 9a represent the locations of the elements of the variable node to be coalesced with the pivot node after the coalescence. Their locations are determined such that, after the coalescence, the resulting derivative set obeys the N-zig-zag order. Therefore, from the structure in the figure, we write $|\mathcal{U}_{z_i,M_i}| = l_e + (c_{e,b} + c_{e,u})\mu_B + r_e$.

Since, after determining the locations to which the elements of the variable node are placed, we no longer need the elements of the pivot node. Therefore, in Fig. 9b, we remove the elements of the pivot node from the picture, and, instead, we depict the elements of the variable node in their original places such that they obey N-zig-zag order. Note that in this proof, our goal is to find a quasi-unique shift (α_1^*, α_2^*) such that there is only one unique placement, characterized by $(\boldsymbol{k}^*, \boldsymbol{l}^*)$, of the elements of the variable node along with the elements of the pivot node. Therefore, we need to track the final location of each element of the variable node and make sure that to the location each element is placed, it is not possible to place another element from the variable node. Therefore, we denote the elements of the variable node by Greek letters and their subscripts. Note that the letters used for this purpose should not be mixed with the other uses of the Greek letters throughout the paper.

Given the depictions in Fig. 9b, the next step is to determine y-directional shifts such that all elements of the variable node are placed to the correct row in the derivative order space. Since, according to Lemma 1, only regular simple shifts are considered, while taking y-directional derivatives, the sequence of the elements having the same x-directional derivative order cannot change. Therefore, for example, α_{μ_B} stays always above the elements denoted by $\alpha_i, i \in [1 : \mu_B - 1]$. Thanks to this property, filling the locations determined to be filled in the new block is straightforward. Shifting the block composed of the variable node's elements with the same shape as the locations to be filled towards y-direction uniquely determines the elements to be moved to the new block. The remaining y-directional shifts will be of the remaining elements of the variable node in the lower block. In Fig. 9c, we depict the shifted elements to the upper new block and the remaining elements together. To have y-directional shifts which generate quasi-unique shifts, whenever we fill a row in the locations determined to be filled in the lower block, the elements to be placed there must be uniquely determined. For example, while filling the top l_e rows, for each row, there must be exactly $c_{e,b}+1$ columns among the elements of the variable node that are available to provide their top-most element. After filling top l_e rows, in the remaining rows, there must be exactly $c_{e,b}$ columns of the elements of the variable node that can provide their top-most element. Therefore, to guarantee this, a sufficient condition is that the shape of the remaining elements of the variable node and the shape of the remaining empty locations match. That is, $\tilde{c}_f = c_{e,b}$ and the remaining elements of the variable node have only one partially-occupied column with l_e elements. There might be several structures satisfying this condition. One of them is when $r_f = 0$ since this implies $l_e + (c_{e,b} + c_{e,u})\mu_B + r_e \equiv 0 \mod \mu_B$. Therefore, $\tilde{l}_f = l_e$. This proves condition 1 of the lemma. Another structure satisfying the sufficient condition is that $r_f = l_e$. When this is the case, $l_e + (c_{e,b} + c_{e,u})\mu_B + r_e = r_f + (c_{e,b} + c_{e,u})\mu_B + r_e = r_f + c_f\mu_B$, implying $\tilde{l}_f = 0$. This proves condition 2 of the lemma. For completeness, note that after the elements are aligned with their final rows via y-directional derivatives, necessary x-directional shifts can be easily applied such that the elements of the variable node are finally placed to their intended locations. Again, due to Lemma 1, we consider only regular simple shifts and therefore, while taking x-directional derivatives, the sequence of the elements having the same y-directional derivative order cannot change.

In the remaining of the proof, we take all x-directional derivatives before y-directional derivatives. In this case, Fig. 9a and Fig. 9b are still valid. However, since we are taking x-directional derivatives first, we first align all the elements of the variable node that are to stay in the lower block with their intended columns. We start with the rightmost column of the lower block, which is column K. When $|\mathcal{U}_{z_n,M_i}| + |\mathcal{U}_{z_i,M_i}| \leq \mu_B K$, this column are not intended to be fully occupied, let

us say only $\tilde{l}_e$ of them will be filled, but the empty locations start from the bottom and they are consecutive until the end. Therefore, the rows of the elements of the variable node that will provide elements to these locations are uniquely determined, namely the rows $[0 : \tilde{l}_e - 1]$ of the elements of the variable node from the bottom. Note that, if $|\mathcal{U}_{z_n,M_i}| + |\mathcal{U}_{z_i,M_i}| > \mu_B K$, then $\tilde{l}_e = \mu_B$, which does not break our argument. After the rightmost elements from the rows $[0 : \tilde{l}_e - 1]$ of the elements of the variable node are shifted to the K^{th} column via x-directional shifts, next, we fill the columns starting from column $K-1$ to column $K-c_{e,b}-1$. Note that since each of these columns are intended to be fully occupied, they are directly filled with the rightmost elements of each row via x-directional shifts. Finally, we fill the column $K - c_{e,b}$, which has l_e locations intended to be occupied after the coalescence. If $|\mathcal{U}_{z_n,M_i}| + |\mathcal{U}_{z_i,M_i}| \leq \mu_B K$, then the upper block is not generated and the number of remaining elements of the variable node is equal to l_e, each on different rows. Thanks to the property that the sequence of the elements having the same y-directional derivative orders cannot change by x-directional shifts, the elements to be placed to the l_e empty locations are uniquely determined. This proves condition 4 of the lemma. On the other hand, when $|\mathcal{U}_{z_n,M_i}|+|\mathcal{U}_{z_i,M_i}| > \mu_B K$, a new block is generated, so there will be always more than l_e remaining elements of the variable node. Therefore, to have a unique shift, in this case, we need $l_e = 0$, which proves condition 3 of the lemma. ■

REFERENCES

[1] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate polynomial coding for straggler exploitation with heterogeneous workers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Los Angeles, CA, USA, 2020, pp. 251–256.

[2] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate hermitian polynomial coding for efficient distributed matrix multiplication," in *Proc. IEEE Global Commun. Conf.*, Taipei, Taiwan, 2020, pp. 1–6.

[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, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4406–4416.

[5] 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. 2020.

[6] 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.

[7] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes for matrix multiplication," 2019. [Online]. Available: arXiv:1811.10751.

[8] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch computation," 2019. [Online]. Available: arXiv:1909.13873.

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

[10] A. Ramamoorthy and L. Tang, "Numerically stable coded matrix computations via circulant and rotation matrix embeddings," 2019. [Online]. Available: arXiv:1910.06515.

[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. Allerton Conf. Commun. Control Comput. (Allerton)*, Monticello, IL, USA, 2019, pp. 253–259.

[12] A. B. Das, A. Ramamoorthy, and N. Vaswani, "Efficient and robust distributed matrix computations via convolutional coding," 2020. [Online]. Available: arXiv:1907.08064.

[13] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, USA, 2018, pp. 1620–1624.

[14] M. M. Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Trans. Signal Process.*, vol. 67, no. 24, pp. 6270–6284, Dec. 2019.

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

[16] E. Ozfatura, S. Ulukus, and D. Gündüz, "Straggler-aware distributed learning: Communication–computation latency trade-off," *Entropy*, vol. 22, no. 5, p. 544, 2020.

[17] S. Kiani, N. Ferdinand, and S. C. Draper, "Hierarchical coded matrix multiplication," in *Proc. 16th Can. Workshop Inf. Theory (CWIT)*, Hamilton, ON, Canada, 2019, pp. 1–6.

[18] R. A. Lorentz, *Multivariate Birkhoff Interpolation*. Heidelberg, Germany: Springer, 2006.

[19] K. E. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed. New York, NY, USA: Wiley, 1988.

[20] T. Sauer and Y. Xu, "On multivariate hermite interpolation," *Adv. Comput. Math.*, vol. 4, no. 1, pp. 207–259, 1995.

[21] T. Sauer, "Computational aspects of multivariate polynomial interpolation," *Adv. Comput. Math.*, vol. 3, no. 3, pp. 219–237, 1995.

[22] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Toronto, ON, Canada, 2014, pp. 826–834.

[23] J. Liesen and V. Mehrmann, *Linear Algebra* (Springer Undergraduate Mathematics Series). Cham, Switzerland: Springer, 2015.