Bivariate Hermitian Polynomial Coding for Efficient Distributed Matrix Multiplication

Burak Hasırcıoğlu*, Jesús Gómez-Vilardebó† and Deniz Gündüz*
*Imperial College London, London, UK, {b.hasircioglu18, d.gunduz}@imperial.ac.uk
†Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Barcelona, Spain, jesus.gomez@cttc.es

***Abstract*—Coded distributed computing is an effective framework to improve the speed of distributed computing systems by mitigating stragglers (temporarily slow workers). In essence, coded computing allows replacing the computation assigned to a straggling worker by that at a faster worker by assigning redundant computations. Coded computing techniques proposed so far are mostly based on univariate polynomial coding. These codes are not very effective if storage and computation capacity across workers are heterogeneous and lose completely the work done by the straggling workers. For the particular problem of distributed matrix-matrix multiplication, we show how bivariate polynomial coding addresses these two issues.**

***Index Terms*—coded computation, distributed computation, distributed matrix multiplication, bivariate polynomial coding**

I. Introduction

Matrix multiplication has a major role in many machine learning and signal processing tasks. Matrices to be multiplied can be very sizable in some tasks such as training machine learning models over large data sets or running inference over large parametric models. Multiplication of massive matrices, especially, under latency constraints, requires substantial amount of computational resources, which are usually not available at a single machine. This motivates distributing the matrix multiplication task to external *workers*. However, due to unpredictable delays in their service times, some workers called *stragglers*, may complete their assigned tasks much slower than others, leading to serious delays. Thus, combating the adverse effects of stragglers is an important problem in distributed computing.

To address this problem one needs to provide some form of computation redundancy to the system, i.e., replicating tasks to other available workers. Recent results have demonstrated that error-correcting codes can be applied to this setting to obtain the needed computation redundancy very efficiently. In this context, polynomial codes are proposed in [1]. These codes are optimal in terms of the download communication costs, i.e., the number of bits that the workers need to send to communicate their results. MatDot codes are proposed in [2]. They require more computation and communication resources at workers, but they achieve smaller *recovery threshold*, which is defined as the minimum number of non-straggling workers required for completing the computation task. PolyDot codes [2] and entangled polynomials codes [3] provide a trade-off between the recovery threshold and the communication-computation costs. However, common to all these approaches is completely ignoring any computations completed by straggler nodes referred to as "under-utilization" in [4]. Specifically, the work assigned to a worker is communicated back to the master only if it is finished completely. This 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 [5], [6]. To exploit the partially completed work done by stragglers, a multi-message approach is considered in [4], [6], [7], where workers' tasks are divided into smaller subtasks, and the result of each subtask is communicated to the master as soon as it is completed. The multi-message approach is applied to the matrix-matrix multiplication in [7] using product codes [8].

In all of the aforementioned works, univariate polynomials are used. In [9], we show that for univariate polynomial coding, dividing a task into subtasks by a given factor, reduces the computation capacity of the workers by the same factor. In case of limited storage at the workers, this approach is not efficient in terms of memory exploitation and upload costs. The product codes proposed in [7], which are basically a combination of two univariate polynomial codes, partially address this issue. However, for product codes, the computations at workers are not one-to-any replaceable anymore. This results in poor performance in worst-case scenarios. Moreover, univariate polynomial codes, as well as product codes, require homogeneous workloads across workers.

In this work, we propose bivariate Hermitian polynomial coding for distributed matrix-matrix multiplication. The proposed scheme allows us to exploit stragglers in highly heterogeneous settings and to reduce the upload communication cost, while guaranteeing (almost surely) that subtasks at workers are one-to-any replaceable. In contrast to univariate coding, the computational capacity of workers is not affected by diving tasks into subtasks. However, decodability, which requires bivariate polynomial interpolation, is highly dependent on the order in which the computation subtasks are completed at the workers. In [9], [10], we proved almost surely decodability for specific storage configurations at workers. Here, we significantly extend those results by relaxing the constraints on the stored matrices at workers. We also numerically show that the new approach beats univariate polynomial codes in [1],

This work received support from the European Research Council (ERC) through Starting Grant BEACON (agreement 677854). The work of J. Gómez-Vilardebó was supported in part by the Catalan Government under Grant SGR2017-1479, and by the Spanish Government under Grant RTI2018-099722-B-100 (ARISTIDES).

product codes in [7] and our previous results in [9], [10] in terms of average computation time.

We organize the paper as follows. In Section II, we present the system model and the problem formulation. We introduce and discuss our coding scheme in Section III. In Section IV, we compare our scheme with previously proposed schemes, and in Section V, we conclude the paper. We give the proof of Theorem 1 in the Appendix.

II. System Model and Problem Formulation

We study the problem of distributed multiplication of two matrices $A \in \mathbb{R}^{r\times s}$ and $B \in \mathbb{R}^{s\times c}$, $r, s, c \in \mathbb{Z}^+$. A master, with access to both matrices, is interested in multiplying them by offloading partial computations to N workers with, possibly, heterogeneous data storage and computation capacities. To distribute the multiplication task, the master partitions matrix A horizontally, and B vertically, into K and L submatrices, 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 $A_i \in \mathbb{R}^{\frac{r}{K}\times s}$, $\forall i \in [K] \triangleq \{1, 2, \ldots, K\}$ and $B_j \in \mathbb{R}^{s\times \frac{c}{L}}$, $\forall j \in [L]$. We assume that worker i can store $m_{A,i} \in \mathbb{Z}^+$ partitions of A and $m_{B,i} \in \mathbb{Z}^+$ partitions of B, $i \in [N]$. Observe that we allow $m_{A,i}$ and $m_{B,i}$ to be different for different workers. This, in turn, allows us to distribute the computation load to workers with heterogeneous storage capacities. To every worker $i \in [N]$, the master sends coded submatrices $\tilde{A}_{i,k} \in \mathbb{R}^{\frac{r}{K}\times s}$, $k \in [m_{A,i}]$ and $\tilde{B}_{i,l} \in \mathbb{R}^{s\times \frac{c}{L}}$, $l \in [m_{B,i}]$ which are encoded by linearly combining the partitions of A and B, respectively.

Every worker multiplies its coded sub-matrices in a special order, to be specified later, and communicates the results one-by-one to the master as soon as each computation is completed. As soon as the minimum number of computations necessary to decode AB are collected by the master, it starts decoding.

The maximum number of (potentially useful) results that can be sent to the master from worker i, without updating its local storage, is referred to as the *maximum computation capacity*, and denoted by η_i. For univariate schemes with task partitioning [1]–[3] , η_i is limited to $\min(m_{A,i}, m_{B,i})$, whereas we will show that the proposed bivariate scheme achieves $\eta_i = m_{A,i}m_{B,i}$. A higher maximum computation capacity is key for minimizing the expected computation time, as it allows workers to provide more computations with the same storage capacity.

III. Proposed Scheme

A. Encoding

We restrict the values of $m_{B,i}$ to be multiples of a common factor m_B, where $m_B \mid L$, while we allow arbitrary values for $m_{A,i}$ if $m_{B,i} = m_B$, and require $m_{A,i} = K$ otherwise. The master partitions the matrices as described in Section II. Coded matrices are generated by evaluating the derivatives of the following univariate polynomials: $A(x) = \sum_{i=1}^{K} A_i x^{i-1}$ and $B(y) = \sum_{i=1}^{L} B_i y^{i-1}$. Specifically, for worker i, all the derivatives of $A(x)$ up to order $m_{A,i}-1$ are evaluated at some $x_i \in \mathbb{R}$. Similarly, we compute all the derivatives of $B(y)$ up to order $m_{B,i} - 1$ are evaluated at some $y_i \in \mathbb{R}$. We require that $x_i \neq y_i$, $x_i \neq x_j$ and $y_i \neq y_j$ for $i \neq j$, $\forall i, j \in [N]$. Thus, the master generates $\mathcal{A}_i \triangleq \{A(x_i), \frac{dA(x_i)}{dx}, \cdots, \frac{d^{m_{A,i}-1}A(x_i)}{dx^{m_{A,i}-1}}\}$ and $\mathcal{B}_i \triangleq \{B(y_i), \frac{dB(y_i)}{dy}, \cdots, \frac{d^{m_{B,i}-1}B(y_i)}{dy^{m_{B,i}-1}}\}, \forall i \in [N]$, and sends them to i^{th} worker. For brevity, in the remaining of the paper, we use $\partial_k A(x_i)$ instead of $\frac{d^k}{dx^k}A(x_i)$, and $\partial_l B(y_i)$ instead of $\frac{d^l}{dy^l}B(y_i)$.

B. Computation

Worker i computes the Cartesian product of the sets $\mathcal{A}_i$ and $\mathcal{B}_i$, i.e., it multiplies all the elements in $\mathcal{A}_i$ with all the elements in $\mathcal{B}_i$, one by one, in an order referred to as the *zig-zag order*, and sends the result of each individual computation to the master as soon as it is completed.

Definition 1. The **interpolation space** E of a polynomial is defined as the space of all possible derivative orders. For our bivariate polynomial $A(x)B(y)$, we have $E = \{(k, l) : 0 \leq k < K, 0 \leq l < L\}$. The tuple (k, l) represents the derivative $\partial_k A(x)\partial_l B(y)$. Note that E is a finite two-dimensional space.

Definition 2. A set of derivatives of $A(x)B(y)$ evaluated at the same point (x, y) constitutes a **node**. Thus, the computations from each worker constitute a node. The set of derivatives of a node can be depicted in the interpolation space. We provide an example of such a depiction of two nodes in the Appendix. Hereafter, we refer to a node by its evaluation point, i.e., node (x, y) is the node with evaluation point (x, y). We refer to the evaluations forming the node as the *derivative set* of the node.

Partial computations at each worker are done in the increasing order of their priority score. The **priority score** of partial computation $(\mathcal{A}_i)_k (\mathcal{B}_i)_l$ at worker i will be denoted by $S_i(k, l)$.

Definition 3. The priority scores for the **zig-zag computation order** are given by $S(k, l) \triangleq (K - 1)m_B\phi + m_B k + l + 1$, where $\phi \triangleq \left\lceil \frac{l+1}{m_B} \right\rceil - 1$ for all the workers. For the visualization of the zig-zag order in the 2-D interpolation space, please see Fig. 1. Observe that a completed partial computation $(\mathcal{A}_i)_k (\mathcal{B}_i)_l$ at the i^{th} worker can be mapped to the derivative order (k, l) in the 2-D interpolation space. In the zig-zag order, we divide the interpolation space row-wise into L/m_B equal horizontal **blocks** of consecutive rows.

C. Decoding

The master receives results from the workers, and decodes AB by solving a bivariate Hermite interpolation problem. The polynomial we interpolate is $A(x)B(y) = \sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{i-1} y^{j-1}$. For this, a necessary condition is to have KL evaluations of $\partial_k A(x_i)\partial_l B(y_i)$, where $k \in [m_{A,i} - 1]$, $l \in [m_{B,i} - 1]$, $i \in [N]$. However, unlike in univariate polynomial interpolation, there is no guarantee that any KL evaluations received from the workers define a unique polynomial. We show, in the next theorem, that if the workers follow the computation order defined in Section III-B, the number of evaluations needed is not more than $KL + \max\left\{0, (m_B - 2)(\frac{L}{m_B} - 1)\right\}$.

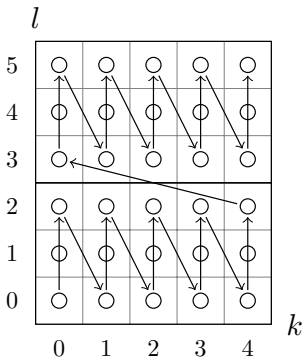

Fig. 1: Visualization of the zig-zag computation order for $K = 5, L = 6, m_B = 3$.

Theorem 1. *Let Z be the union of different nodes, the derivative set of each following the zig-zag order. If $|Z| = KL + \max\left\{0, (m_B - 2)(\frac{L}{m_B} - 1)\right\}$ and for node $(x_i, y_i), i \in [N]$, either $m_{B,i} = m_B$ or $m_{A,i} = K$ is satisfied, then for almost all choices of interpolation points at nodes, i.e., $(x_i, y_i) \in \mathbb{R}^2$, $\forall i \in [N]$, there exists a unique interpolation solution of the polynomial $A(x)B(y)$.*

Remark 1. *We could instead consider the case in which $m_{A,i} = m_A$ is fixed and $m_{B,i}$ is different for every user. In that case, the computation order at workers needs to follow a zig-zag type order in which the interpolation space is partitioned into vertical blocks and the workers go first in the right-horizontal direction. The number of computations that need to be received from the workers in that case is not more than $KL + \max\left\{0, (m_A - 2)(\frac{K}{m_A} - 1)\right\}$. Due to the space restrictions, we do not include this approach in this work, but the extension is trivial. For a discussion on the choice between these two approaches, we refer to [9].*

Remark 2. *We emphasize that by setting $m_B = L$ or $m_B = 1$, the zig-zag order reduces to the vertical or horizontal orders considered in [9], respectively. Similarly, in the horizontal extension of the zig-zag order, setting $m_A = K$ or $m_A = 1$ reduces to the horizontal and vertical orders in [9], respectively.*

Remark 3. *Instead of Hermite interpolation, we could also consider bivariate interpolation coding as in [9], [10], in which a worker multiplies the evaluations of $A(x)$ and $B(y)$ at different evaluation points instead of different derivatives evaluated at the same point. As shown in [9], to prove the almost regularity of such bivariate interpolation coding, we need first to transform it into a bivariate Hermitian interpolation problem. This is possible under almost regularity condition. Here, we directly describe Hermitian coding that result from such a transformation.*

Bivariate polynomial coding improves the upload communication cost, C_u, defined as the minimum number of bits the master needs to upload to recover AB. Suppose a bits are required to represent rows/columns of matrices. Then, in univariate polynomial coding, associated to each subtask there is an upload cost of $a(\frac{r}{K} + \frac{c}{L})$ bits. Since the full task requires KL subtasks to finish, $C_u = a(rL + cK)$. However, in bivariate polynomial coding, each evaluation of $A(x)$ and $B(y)$ may contribute to up to m_B and m_A subtasks, respectively, assuming homogeneous storage capacity. Thus, each subtask has an upload communication cost of $a(\frac{r}{Km_B} + \frac{c}{Lm_A})$, resulting in $C_u = a(r\frac{L}{m_B} + c\frac{K}{m_A})$.

In many works in the literature, storage capacity at the workers is assumed to be specified separately for each of the two matrices. However, in practice it is more reasonable 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 storage at workers. Let $m_i \in \mathbb{N}^+$ be total number of rows/columns i^{th} worker can store. The computation time is improved with a higher maximum computation capacity η_i and less discarded computations. In bivariate coding schemes, the computation capacity of a worker is given by $\eta_i = m_{A,i}m_{B,i}$. Accordingly, for given K and L, we choose $m_{A,i}$ and $m_{B,i}$ to maximize $m_{A,i}m_{B,i}$ subject to $m_{A,i}r/K + m_{B,i}c/L = m_i$.

In the scheme we propose, in terms of memory allocation, we significantly improve upon [9], [10] especially in the low-memory regimes, and come close to the optimal. We restrict $m_{B,i}$ to be a multiple of a common factor m_B with $m_B \mid L$ and we do not impose any conditions on $m_{A,i}$ if $m_{B,i} = m_B$ and set $m_{A,i} = K$ otherwise. Moreover, we allow heterogeneous storage capacities across the workers. In a worst-case situation, we may need to ignore up to $(m_B - 2)(\frac{L}{m_B} - 1)$ computations.

The bivariate extension of [7], which is presented in [9], [10], achieves the optimal memory allocation as there are no additional constraints on $m_{A,i}$ or $m_{B,i}$. However, it works only in case of homogeneous storage capacity at workers. Moreover, in the worst-case, there are $(n_A m_{B,i} - L)(K - 1) + (n_B m_{A,i} - K)(L - 1)$ useless computations, where n_A and n_B are design parameters such that $N = n_A n_B$. This is far larger than the number of computations that may be discarded by our scheme. This quantity can be significant and increases the average computation time.

On the other hand, in [9], [10], by forcing a vertical or a horizontal computation order, every computation is made useful. Moreover, it allows heterogeneous storage capacity across the workers. However, in the case of horizontal computation order, $m_{B,i} = K$ or $m_{A,i} = 1$, and in the case of vertical computation order, $m_{A,i} = L$ or $m_{B,i} = 1$ is required. These constraints incur a loss of optimality in the memory allocation between the partitions of A and B. This may result in a considerable increase in the average computation time in low-memory regimes. However, not discarding any computation dominates if the storage at the workers is sufficiently large and highly improves average computation time.

IV. Numerical Results

We compare the expected computation times for the proposed scheme, the scheme in [9], [10], the bivariate extension of the scheme in [7], which is presented in [9], [10] and the univariate polynomial coding scheme in [1] with partial

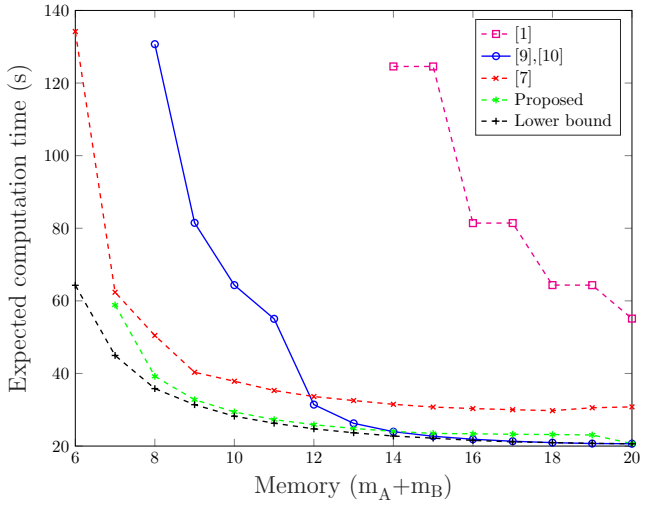

Fig. 2: Average computation times of our scheme and the previous schemes.

computations. We assume that all the workers start computations simultaneously and define the computation time as the time required to obtain sufficient responses from the workers to interpolate $A(x)B(y)$. We only focus on the computation time since the bivariate polynomial to be interpolated in all three schemes has the same number of coefficients, and thus the encoding and decoding complexities are very similar. We assume the communication time is negligible. We model the computation speed of the workers by the shifted exponential model [1], 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, $P(p,t) \triangleq F(p,t) - F(p+1,t)$ gives the probability that a worker completes exactly p computations at time t, where $F(0,t) = 1$, and $F(p_{max}+1,t) = 0$, where $p_{max} = \eta_i = \eta, \forall i \in [N]$. ν is the minimum duration in which a worker completes a partial computation. A smaller λ means a higher variance, and thus more heterogeneous computation speeds across the workers. To cover more heterogeneous cases, we choose $\nu = 0.01$ and $\lambda = 0.1$. Note that we use the same parameters for every worker. This means on average, the behavior of the workers are similar, but in every experiment, their speeds can differ a lot.

We run Monte Carlo simulations to find the expected computation times of the schemes for different memory sizes at the workers. We assume the sizes of the partitions of A and B are equal. We further assume that the workers have the same storage capacity, i.e. homogeneous case, as this is required by the scheme proposed in [7]. We partition both matrices into $K = L = 10$ pieces, and employ $N = 15$ workers. For each memory value, we run 10^4 experiments and take the average. The results are presented in Fig. 2. 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.

We observe that the proposed scheme results in a much lower expected computation time than previously proposed schemes for low storage capacities. Even though we allow partial computations, the univariate polynomial coding scheme in [1] performs far worse than all the others due to inefficient use of the memory resulting in a very limited computation capacity. Despite the optimality in the memory allocation between $m_{A,i}$ and $m_{B,i}$, we see that high number of useless computations aggravates the average computation time in the bivariate extension of [7]. For this code, increasing the storage capacity does not improve the average computation time beyond a certain point. Note that, while simulating [7], we use a random computation order at the workers, which is reported to perform well in [7]. That is, instead of assuming the worst-case, for every computation received by the master, we decide if it is useful or not. On the other hand, for our proposed scheme, we assume the worst-case, i.e., we always ignore $(m_B-2)(\frac{L}{m_B}-1)$ computations. Hence, we expect the improvement to be even more than what we observe in Fig. 2. We also observe that the scheme in [9], [10], also a bivariate polynomial coding scheme, performs well for the intermediate and large memory values. However, when the memory is limited, there is significant performance degradation. The proposed scheme solves the problem in small memory values and shows the effectiveness of bivariate polynomial coding. To better illustrate this, we also plot a lower bound on the average computation time of any bivariate polynomial-based coding scheme, assuming no computation is ever wasted and the memory allocation between $m_{A,i}$ and $m_{B,i}$ is optimal. Observe that the average computation time of our scheme is quite close to this lower bound.

V. Conclusion

We proposed a bivariate Hermitian polynomial coding strategy for memory-efficient straggler exploitation problem in distributed matrix-matrix multiplication. The non-trivial problem in bivariate polynomial interpolation is showing the invertibility of the interpolation matrix. We show that if workers follow a predetermined order for their partial computations, which we refer to as the zig-zag order, and allow for a few additional redundant computations, then decodability is guaranteed almost surely. The proposed strategy's ability to exploit the workers' computation capacity is close to the optimal, for highly heterogeneous storage and computation capacity at the workers, while keeping the number of wasted computations relatively small. We numerically showed that in terms of the average computation time, our scheme performs better than its rivals in the literature.

Appendix: Proof of Theorem 1

We first introduce some useful concepts and results.

Definition 4. The polynomial interpolation problem can be formulated as a system of linear equations, whose unknowns are the coefficients of $A(x)B(y)$, namely A_pB_q where $p \in [K], q \in [L]$. As such, it has a unique solution if and only if its coefficient matrix, here referred to as **interpolation matrix**, is invertible. We denote the interpolation matrix associated

with a set of nodes, Z, as $M(Z)$. Since $A(x)B(y)$ has KL coefficients, the interpolation matrix is a square matrix of size KL.

Definition 5. Given the interpolation matrix M, the Taylor series expansion of $\det(M)$ with respect to the evaluation point (x_j, y_j) around (x_i, y_i) is given by

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

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

We refer to node (x_i, y_i) as the **pivot node** and node (x_j, y_j) as the **variable node**.

Since bivariate monomials $(x_j - x_i)^{\alpha_1}(y_j - y_i)^{\alpha_2}$ are linearly independent for different values of α_1 and α_2, the determinant becomes zero $\forall(x_j, y_j)$ only if $D_{\alpha_1,\alpha_2}(x_i, y_i) = 0$, $\forall\alpha_1, \alpha_2$. For specific values of x_i, y_i, x_j, y_j, there might be cases in which determinant is zero even if $D_{\alpha_1,\alpha_2}(x_i, y_i) \neq 0, \forall\alpha_1, \alpha_2$, i.e. at the zeros of the polynomial in the Taylor expansion of $\det(M(x_j, y_j)$ in (1). However, the Lebesgue measure of the zeros of a multivariate polynomial in $\mathbb{R}^2$ is 0 if the polynomial is not zero everywhere. Thus, if we can prove that $D_{\alpha_1,\alpha_2}(x_i, y_i) \neq 0$ for some α_1, α_2 and almost all choices of the evaluation points $(x_k, y_k), \forall k \in [N]$, then $M(Z)$ is invertible almost surely. This relaxed invertibility notion is known as almost regularity [11] and our objective is to prove the almost regularity of $M(Z)$.

Note that, by definition, $D_{\alpha_1,\alpha_2}(x_i, y_i)$ does not depend on the variable node's evaluation point (x_j, y_j) anymore. This procedure of eliminating the variable node's evaluation point is known as **coalescence** of a pivot and a variable node.

In the Lemma 1, stated next, we compute $\frac{\partial^{\alpha_1+\alpha_2}}{\partial x_j^{\alpha_1}\partial y_j^{\alpha_2}} \det(M(x_j, y_j))$. However, first, we need to introduce the concept of regular shift.

Definition 6. Let $M(x_j, y_j) \in \mathbb{R}^{n\times n}$ be an interpolation matrix as a function of the evaluation point of node (x_j, y_j) and denote by r_i its i^{th} row. We define the k^{th} order derivative with respect to x_j and the l^{th} order derivative with respect to y_j of the i^{th} row as $\partial_{i,k,l}M(x_j, y_j) \triangleq \left[r_1^T, ..., \frac{\partial^{k+l}}{\partial x_j^k \partial y_j^l} r_i^T ..., r_n^T\right]^T$. We also refer to the operator $\partial_{i,k,l}$ as a **shift** in the i^{th} row of $M(x_j, y_j)$. We say that the shift is **simple** in the x direction if $(k, l) = (1, 0)$ and simple in the y direction if $(k, l) = (0, 1)$.

Notice that only the rows of M associated node (x_j, y_j), actually, depend on (x_j, y_j). Let the row r_i correspond to the derivative $\partial_{k_i}A(x_j)\partial_{l_i}B(y_j)$ of node (x_j, y_j), then the derivative set of node (x_j, y_j) in $\partial_{i,k,l}M(x, y)$ contains $\partial_{k_i+k}A(x_j)\partial_{l_i+l}B(y_j)$.

Definition 7. Given a pair of length$-n$ vectors $(\boldsymbol{k}, \boldsymbol{l})$ with $(\boldsymbol{k}(i), \boldsymbol{l}(i)) \in [N]^2$, we define the shift $(\boldsymbol{k}, \boldsymbol{l})$ as

$$\nabla_{\boldsymbol{k},\boldsymbol{l}} M(x_j, y_j) \triangleq \left(\prod_{i=1}^{n} \partial_{i,\boldsymbol{k}(i),\boldsymbol{l}(i)}\right) M(x_j, y_j).$$

Thus, the i^{th} entry of $\boldsymbol{k}$ and $\boldsymbol{l}$ denote the order of the derivative taken on the i^{th} row of M with respect to x_j and y_j, respectively. Let $\alpha_1 = \sum_{i=1}^n \boldsymbol{k}(i)$ and $\alpha_2 = \sum_{i=1}^n \boldsymbol{l}(i)$. We say that (α_1, α_2) is the **order of the shift**.

Definition 8. We say that a **shift** $(\boldsymbol{k}, \boldsymbol{l})$ **is regular** if it can be achieved by at least one sequence of simple shifts, and after each simple shift there are not two equal rows or a zero row in the shifted matrix. We denote the number of sequences of simple shifts of the regular shift $(\boldsymbol{k}, \boldsymbol{l})$ as $C_{\boldsymbol{k},\boldsymbol{l}}$ and the **set of all regular shifts** of order (α_1, α_2) as $\mathcal{R}(\alpha_1, \alpha_2)$.

Lemma 1. *(Lemma 3 of [9]) We have*

$$\frac{\partial^{\alpha_1+\alpha_2}}{\partial x_j^{\alpha_1}\partial y_j^{\alpha_2}} \det(M(x_j, y_j)) = \sum C_{\boldsymbol{k},\boldsymbol{l}} \det \nabla_{\boldsymbol{k},\boldsymbol{l}} M(x_j, y_j) \tag{2}$$

where the summation is over all regular shifts, i.e. $(\boldsymbol{k}, \boldsymbol{l}) \in \mathcal{R}(\alpha_1, \alpha_2)$.

Definition 9. If $|\mathcal{R}(\alpha_1, \alpha_2)| = 1$ then the one shift in $\mathcal{R}(\alpha_1, \alpha_2)$ is called **unique shift**, and (α_1, α_2) is called **unique shift order**.

Let us move back to the problem of proving $D_{\alpha_1,\alpha_2}(x_i, y_i) \neq 0$. If a unique shift $(\boldsymbol{k}, \boldsymbol{l})$ exists, then the problem of proving $D_{\alpha_1,\alpha_2}(x_i, y_i) \neq 0$ reduces to proving that $\nabla_{\boldsymbol{k},\boldsymbol{l}} M(x_j, y_j)\big|_{x_j=x_i, y_j=y_i}$ is invertible. To that end, we repeat the procedure.

Definition 10. Coalescence procedure on a fixed pivot. Choose a pivot node, and consider a sequence of coalescences in which after each coalescence, the resultant coalesced node is used as the pivot for the next coalesce in the sequence until all other nodes are coalesced into one with a derivative set covering the full interpolation space.

Following the procedure in Definition 10, if in every intermediate step, we can find a unique shift, then M is non-singular for almost all choices of evaluation points. Unfortunately, in our coding scheme, finding a unique shift in every step may not be possible. Next, we give a relaxation of the notion of unique shift which is useful to prove the almost regularity of our scheme.

Definition 11. A shift order (α_1, α_2) is called **quasi-unique** if the RHS of (2) evaluated at $(x_j, y_j) = (x_i, y_i)$ can be written as the sum of the determinant of one main interpolation matrix $\nabla_{\boldsymbol{k}_0,\boldsymbol{l}_0} M(x_j, y_j)\big|_{x_j=x_i, y_j=y_i}$ for which the derivative set of node (x_i, y_i) obeys the zig-zag order, and, possibly, the determinant of some other secondary interpolation matrices, for which the derivative set associated to node (x_i, y_i), does not obey the zig-zag order. We call the shift $(\boldsymbol{k}_0, \boldsymbol{l}_0)$ quasi-unique.

In the next lemma, we extend the successive coalescence procedure on unique shifts to quasi-unique shifts.

Lemma 2. *Given a set of nodes Z s.t. $|Z| = KL$. Consider the coalesce procedure on a fixed pivot described in Definition 10. If for each coalescence, we can find a shift which is quasi-unique for the main interpolation matrix of*

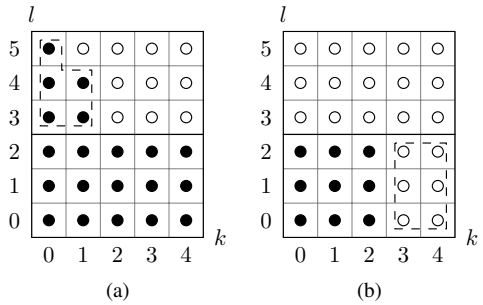

Fig. 3: Derivative sets of variable and pivot nodes in Example 1 depicted in the interpolation space.

the previous quasi-unique shift, then the last coalesce returns only one interpolation matrix defined by one node covering the complete interpolation space, and thus, the associated interpolation matrix is invertible.

The proof of Lemma 2 will be included in a longer version of this paper. Now, the problem reduces to showing if we can always find a series of quasi-unique shifts to coalesce all the nodes into one under the conditions of Theorem 1.

Definition 12. Remember, in the zig-zag order, we divide the interpolation space into L/m_B consecutive equal horizontal blocks. In a coalescence procedure, consider the occupation of the derivative set of the variable and the pivot node within the horizontally partitioned interpolation space. We denote by z, the number of elements in the uppermost partially filled horizontal block of the variable node's derivative set and denote by e, the number of empty spaces in the uppermost partially filled horizontal block of the interpolation space once filled with the derivative set of the pivot node.

Example 1. Let us assume $K = 5, L = 6$ and $m_B = 3$. Given that they follow the zig-zag order, a variable node having 20 computations and a pivot node having 9 computations are depicted in the interpolation space in Fig. 3a and Fig. 3b, respectively. We represent all possible 2-D derivative orders with the circles, and if a derivative order is present in the node's derivative set, then the circle is filled. In the figure, we can observe that the interpolation space is divided into 3 horizontal blocks. According to Definition 12, $z = 5$ and $e = 6$. These elements are shown within a dashed-lined polygon in the corresponding figures.

Lemma 3. *Given a variable node and a pivot node, which are zig-zag ordered, if at least one of the following conditions is satisfied,*

(a) $z \leq e$,

(b) $z > e$ *and the uppermost block of the variable node consists of only fully occupied columns,*

(c) $z > e$ *and the number of elements in the rightmost partially-occupied column in the uppermost block of the variable node is equal to the number of the empty spaces in the leftmost partially-occupied column in the uppermost block of the pivot node,*

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

The proof of Lemma 3 will be included in a longer version of this paper. Lemma 3 provides the situations in which quasi-unique shift exits. If we are in a situation not covered by Lemma 3, then we can always remove elements of the derivatives sets of the variables node just ignoring the associated partial computation at the master until we satisfy Lemma 3. This is possible since we are applying a sequence of coalescences on a fixed pivot node.

To obtain Theorem 1 from Lemma 3, we need to compute the number of computations we need to ignore along the successive coalescence procedure in a worst-case situation. Let $e = am_B + b$, $a, b \in \mathbb{Z}^+$, $b < m_B$ and $z = cm_B + d$, $c, d \in \mathbb{Z}^+$ and $d < m_B$. If $d > b$, we can satisfy condition c) in Lemma 3 by forcing $b = d$ by ignoring $d - b$ computations. Thus, in the worst-case, we ignore $\max(d - b) = (m_B - 1) - 1 = m_B - 2$ computations. On the other hand, if $d < b$, we ignore d computations and satisfy condition b) in Lemma 3. Since $d < b < m_B$, in the worst-case, we need to ignore $\max d = m_B - 2$ computations. Thus, in either case, the maximum number of computations we ignore is $m_B - 2$. Observe that if $z > e$, the resultant pivot node occupies the next horizontal block. Given that there are L/m_B horizontal blocks, the maximum number of coalescences for which $z > e$ is at most $(L/m_B - 1)$. For this to happen, all of the nodes from the workers must have strictly less than Km_B elements; and therefore, we need more than $\frac{L}{m_B}$ workers. Thus, in the worst-case, the total number of ignored computations is $(m_B - 2)(\frac{L}{m_B} - 1)$. This completes the proof of Theorem 1. □

REFERENCES

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

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

[3] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *IEEE International Symposium on Information Theory*, 2018.

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

[5] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *IEEE International Symposium on Information Theory*, 2018.

[6] M. M. Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6270–6284, 2019.

[7] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *IEEE International Symposium on Information Theory*, 2018.

[8] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *IEEE International Symposium on Information Theory*, 2017.

[9] B. Hasircioglu, J. Gomez-Vilardebo, and D. Gunduz, "Bivariate polynomial coding for exploiting stragglers in heterogeneous coded computing systems," *arXiv preprint arXiv:2001.07227*, 2020.

[10] B. Hasircioglu, J. Gomez-Vilardebo, and D. Gunduz, "Bivariate polynomial coding for straggler exploitation with heterogeneous workers," *IEEE International Symposium on Information Theory*, 2020.

[11] R. A. Lorentz, *Multivariate Birkhoff Interpolation*. Springer, 1992.