Bivariate Polynomial Coding for Straggler Exploitation with Heterogeneous Workers

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

***Abstract*—Polynomial coding has been proposed as a solution to the straggler mitigation problem in distributed matrix multiplication. Previous works employ univariate polynomials to encode matrix partitions. Such schemes greatly improve the speed of distributed computing systems by making the task completion time to depend only on the fastest workers. However, they completely ignore the work done by the slowest workers resulting in inefficient use of computing resources. In order to exploit the partial computations of the slower workers, we further decompose the overall matrix multiplication task into even smaller subtasks, and we propose bivariate polynomial codes. We show that these codes are a more natural choice to accommodate the additional decomposition of subtasks, and to exploit the heterogeneous storage and computation resources at workers. However, in contrast to univariate polynomial decoding, guarantying decodability with multivariate interpolation is much harder. We propose two bivariate polynomial coding schemes and study their decodability conditions. Our numerical results show that bivariate polynomial coding considerably reduces the computation time of distributed matrix multiplication.**

I. Introduction

Matrix multiplication is one of the most crucial building blocks of many machine learning tasks. Availability of massive datasets and large 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. Thus, to speed up learning, it is necessary to distribute the most demanding computation tasks, e.g. matrix multiplication, to multiple dedicated servers, called *workers*. However, due to unpredictable delays in their service time, some workers, called *stragglers*, may complete their assigned tasks much slower than the others, leading to serious delays. Mitigating the negative impact of stragglers on the *completion time* of the distributed matrix multiplication has recently been a very active research area [1]–[13].

One can reduce the effects of the stragglers in the completion time by employing redundant workers. It has been shown in [1] that, rather than simply assigning each computation task to multiple redundant workers, i.e., repetition coding, one can treat stragglers as erasures, and improve the completion time significantly by using ideas from channel coding. In [2], polynomial codes are employed to speed up matrix multiplication, i.e., $A \cdot B$. They propose partitioning A row-wise and B column-wise and encoding them into two separate polynomials of the same variable. Their design is optimal in terms of download rate, which is the ratio of the total number of bits needed to be downloaded from the workers to the number of bits needed to represent the result of the multiplication. In [3], the authors proposed MatDot codes with an alternative partitioning of matrices, in which A is split column-wise and B is split row-wise. They show that compared to [2], their approach improves the recovery threshold, which is defined in [3] as the minimum number of workers' responses needed to decode the result. However, in [3], the computation load at workers and the communication cost are higher than [2]. Also in [3], PolyDot codes are proposed for square matrices as an interpolation between polynomial codes in [2], and MatDot codes by trading off recovery threshold and cost of communication and computation. In [4], the same problem is studied for arbitrary matrices, and entangled polynomial codes are proposed improving the recovery threshold in [3]. Generalized PolyDot codes [5] are proposed for matrix-vector multiplication in the context of neural network training achieving the recovery threshold in [4]. Recently, in [6], batch multiplication of matrices, i.e., A_iB_i, $i \in [L]$ where $L > 1$, is studied and 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.

Another important aspect of distributed computation is the heterogeneity in workers' computational capacities. If the statistics about this heterogeneity are known to the code designer, then it can be used to balance the computational load of the workers. In [7] the authors assume that the computation times of workers follow a shifted exponential distribution, whose parameters differ across the workers. The authors also assume that a worker's responses are either used as a whole or not used at all. Under this setting, the optimal load allocation problem is solved in [7].

All of these works treat straggler nodes that fail to complete the assigned task as an erasure which implies ignoring completely the work done by them. However, it has been observed in practical implementations [8], [9] that, although the workers are not homogeneous, their computation capabilities are typically similar, and it is rare that a server is completely inactive. In [10], to exploit all the work done at the system, including stragglers, tasks of the workers are further

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 in part by the Spanish Government under Grant RTI2018-099722-B-100 (ARISTIDES).

divided into smaller partial computations, to allow workers to communicate their partial computations. Therefore, even if a worker is slow, and cannot complete all of its assigned tasks on time, some amount of the work done by this worker can still be exploited. Product codes [11] are used as the underlying coding scheme in [10], but polynomial codes can also be used instead. All of the aforementioned polynomial coding approaches use univariate polynomials for which the storage of the workers is not efficiently utilized.

To exploit partial computations, uncoded computation with scheduling is considered in [9]. In [12], a hybrid of uncoded and coded computation is proposed for the same problem in distributed gradient descent. In these works, it is shown that uncoded computation may be more beneficial if workers are relatively homogeneous, i.e., less diverse computation statistics, which is ofter the case in web services like AWS, Azure etc. However, in our work, we focus on heterogeneous systems, such as in volunteer computing, peer-to-peer applications, or edge computing which typically exhibit much more computational heterogeneity.

In [8], a hierarchical coding framework for straggler exploitation problem is proposed concerning decoding time, which is the time spent to recover the main computation task from partial computations, in addition to the computation time. The work in [8] is extended to matrix-vector and matrix-matrix multiplications in [13]. For both type of multiplications, they numerically and experimentally show that, while gaining in terms of the decoding time, the computation time of hierarchical coding is only slightly larger than [10] with univariate polynomial coding. Thus, the benefits of hierarchical coding are relevant mainly if the decoding time is comparable to the computation time.

In this work, we focus on the computation time in distributed matrix-matrix multiplication with an emphasis on the efficient use of storage capacities at workers. Similarly to [8]–[10], [12], [13], our main goal is to exploit partial computations carried out by stragglers. If the partial computations of workers are utilized and the workers are heterogeneous in their computational power and storage, the best we can do is to maximize the number of computations every worker can provide. Thus, any solution to the partial work exploitation problem is also applicable in the heterogeneous workers' case, which is also studied in [7]. We propose a novel bivariate polynomial coding technique that allows us to use data storage capacities of the workers more efficiently, and thus improves the computation time. To the best of our knowledge, this is the first time in the literature that multivariate polynomials are used for distributed matrix-matrix multiplication. Similar to [2], our focus is on the schemes with minimum download rate. Choosing the partitioning as in [2] is convenient for us since, as opposed to MatDot, PolyDot and entangled polynomial codes [3], [4], there are no useless terms in the final product and no need for interference alignment in polynomial codes, which is difficult with multivariate polynomials.

II. System Model and Problem Formulation

In our setup, a central server (CS) is requested to multiply two matrices $A \in \mathbb{R}^{r\times s}$ and $B \in \mathbb{R}^{s\times c}$ for some integers r, s, c, by offloading partial computations to N distributed workers, with possibly heterogeneous data storage and computation capacities. In order to distribute the computation work, matrices A and B are partitioned 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 $A_i \in \mathbb{R}^{\frac{r}{K}\times s}$, $\forall i \in [K] \coloneqq \{1, 2, \cdots, K\}$ and $B_j \in \mathbb{R}^{s\times \frac{c}{L}}$, $\forall j \in [L]$. The CS generates and sends to worker $i \in [N]$, $m_{A,i}$ and $m_{B,i}$ coded matrices $\tilde{A}_{i,k}$ and $\tilde{B}_{i,l}$ based on A and B, respectively, for $k \in [m_{A,i}]$ and $l \in [m_{B,i}]$, 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 $i \in [N]$ is assumed to have a capacity 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 matrices are generated depends on the specific coding scheme employed. In this work, coded matrices are obtained as linear combinations of the original matrix partitions.

Worker i computes the products of the coded submatrices assigned to it, i.e., $\tilde{A}_{i,k}\tilde{B}_{i,l}$, $k \in [m_{A,i}]$, $l \in [m_{B,i}]$. To exploit the partial work done by the workers, the results of these individual products are sent to the CS as soon as they are finished. The maximum number of results that can be sent to the CS from worker i, without updating the local storage is denoted by η_i, such that $\eta_i \leq m_{A,i}m_{B,i}$, which depends on the coding scheme and is thus a measure of the memory efficiency of the code. Finally, the CS collects all responses from the workers to decode the product AB. We define **the recovery threshold** R_{th} as the minimum number of responses the CS must receive from the workers to decode the product AB. For all the coding schemes discussed here, we have $R_{th} = KL$. The computational complexity of the partial product $\tilde{A}_{i,j}\tilde{B}_{i,l}$ is a fraction $C_{\text{part}} = \frac{1}{KL}$ of the computational complexity of the full product AB. Hence, the maximum work done at the worker i is a fraction $C_{\text{full},i} = \eta_i C_{\text{part}} = \frac{\eta_i}{m_{A,i}m_{B,i}} M_{A,i}M_{B,i}$ of the work required to compute AB.

The metric $C_{\text{full},i}$ captures how the maximum computation capacity of a worker changes with the size of the partial computations. Increase in the computation capacity of workers means faster workers can provide more fraction of the overall computation. Hence, high $C_{\text{full},i}$ implies less computation time. Thus it is an important performance metric to show the memory-efficiency of the proposed schemes. To better exploit the partial work done by the stragglers, size of the partial computations are decreased. While doing so, we do not want to jeopardize $C_{\text{full},i}$. In this sense, we are interested in high storage efficiency η_i schemes with low partial computation complexity C_{part} to better exploit the partial work done at stragglers.

III. Univariate Polynomial Coding

In this section, we review already existing coding schemes based on univariate polynomial interpolation. Using univariate polynomials for distributed matrix multiplication was first suggested in [2]. Here, we combine [2] with the ideas in [10] to exploit partial computations done at workers. By doing so,

we illustrate the limitations of univariate polynomial coding to exploit the partial work done by the stragglers.

Scheme 1: The CS encodes the submatrices using the following polynomials: $\tilde{A}(x) = A_1 + A_2 x + \cdots + A_K x^{K-1}$ and $\tilde{B}(x) = B_1 + B_2 x^K + \cdots + B_i x^{(i-1)K} + \cdots + B_L x^{(L-1)K}$. We allow worker i to store $m_i = m_{A,i} = m_{B,i}$ coded partitions of matrices A and B. That is $M_{A,i} = m_i/K$ and $M_{B,i} = m_i/L$. For worker i, the CS evaluates $\tilde{A}(x)$ and $\tilde{B}(x)$ at m_i distinct points $\{x_{i,1}, \cdots, x_{i,m_i}\}$ such that $x_{i,k} \neq x_{j,l}$ if $(i,k) \neq (j,l), \forall i,j \in [N]$ and $\forall k \in [m_i], \forall l \in [m_j]$. Worker i computes $\tilde{A}(x_{i,j})\tilde{B}(x_{i,j})$, consecutively, for $j \in [m_i]$ and sends the result to the CS after completion of every partial computation to exploit stragglers. Observe that multiplications are only allowed between $\tilde{A}(x)$ and $\tilde{B}(x)$ evaluated at the same points $x_{i,k}$, and thus $\eta_i = m_i$,$\forall i \in [N]$. The CS is able to interpolate $C(x) = \tilde{A}(x)\tilde{B}(x) = \sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{i-1+K(j-1)}$ of degree $KL-1$ as soon as it receives $R_{th} = KL$ responses from the workers. Thus $C_{\text{part}} = \frac{1}{KL} = \frac{M_{A,i}M_{B,i}}{m_i^2}$ and $C_{\text{full},i} = m_i C_{\text{part}} = \frac{M_{A,i}M_{B,i}}{m_i}$. Observe that, for fixed storage capacity at the workers, the maximum fraction of work done at worker i, which is $C_{\text{full},i}$, is inversely proportional to m_i. That means, for fixed memory allocated to each matrix, i.e., constant M_A and M_B, if we partition matrices into more pieces to utilize partial computations better, and thus increase KL, the maximum computation capacity of a worker decreases. This results in inefficient use of storage since with the same amount of memory, every worker can compute less. Bivariate schemes presented in the next section addresses this problem.

IV. Bivariate Polynomial Coding

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

Definition 1: Given a (multivariate) polynomial $C(z)$, and a set of points $Z = \{z_1, \cdots, z_s\}$, and the evaluations of the polynomial at these points, i.e., $C(z), z \in Z$, we can formulate the interpolation problem as a linear system of equations. The unknowns of these equations are the coefficients of the polynomial. We define the **interpolation matrix** as the coefficient matrix of this linear system. We denote interpolation matrix and its determinant by $M(Z)$ and $D(Z)$, respectively. □

Definition 2: A set of evaluation points Z is called **poised** if the interpolation matrix for these points is invertible. An interpolation scheme, i.e., a specific set of rules between the evaluation points, is called **regular** if every set of allowed evaluation points is poised. An interpolation scheme is **almost regular** if $D(Z) \neq 0$ for almost all sets of nodes, Z. This means there is no special structure making $D(Z)$ zero; and thus, if we would draw the elements of Z uniformly random from the space whose elements are in $\mathbb{R}$ and satisfies the set of rules imposed by interpolation scheme, then the measure of the event $\{D(Z) = 0\}$ becomes zero.

It is well known that univariate polynomial interpolation is regular. It follows from the fact that, given a set of n distinct points $Z = \{x_1, ..., x_n\}$, the interpolation matrix for univariate interpolation of degree $n-1$ is always invertible if x_i's are distinct. Instead, for bivariate interpolation there are very few cases for which sufficient conditions for regularity are known. Unfortunately, as we will illustrate in the next subsection, these known cases do not perfectly fit to distributed computing schemes. □

In univariate schemes, the reason behind storage inefficiency is that workers are limited to use the same evaluation points for $\tilde{A}(x)$ and $\tilde{B}(x)$ while computing $\tilde{A}(x)\tilde{B}(x)$. However, as we show next, in bivariate polynomial interpolation, we also exploit cross-products such as $\tilde{A}(x_{i,j})\tilde{B}(x_{i,k})$ when $j \neq k$. We encode partitions of A with $\tilde{A}(x) = A_1 + A_2 x + \cdots + A_K x^{K-1}$ and partitions of B with $\tilde{B}(y) = B_1 + B_2 y + \cdots + B_L y^{L-1}$. Thus, the CS needs to interpolate the bivariate polynomial $C(x,y) = \sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{i-1} y^{j-1}$. In this case, the row of the interpolation matrix $M(Z)$ associated to the evaluation point $Z_i = (x_i, y_i)$, is $[x_i, \cdots, x_i^{K-1}, y_i, \cdots, x_i^{K-1}y_i, \cdots, y_i^{L-1}, \cdots, x_i^{K-1}y_i^{L-1}]$.

To see the potential benefits of bivariate interpolation based strategies, suppose that the first KL evaluations returned from workers are poised for interpolating $C(x,y)$. Then, $C_{\text{part}} = \frac{1}{KL} = \frac{M_{A,i}M_{B,i}}{m_{A,i}m_{B,i}}$ and $C_{\text{full},i} = m_{A,i}m_{B,i}C_{\text{part}} = M_{A,i}M_{B,i}$. Observe that, unlike univariate schemes, for a given storage capacity $M_{A,i}$ and $M_{B,i}$, the maximum amount of work done at worker i, i.e., $C_{\text{full},i}$, does not decrease with KL anymore. Thus, we are now using available memory more efficiently. This is the advantage of the bivariate coding over the univariate coding. Unfortunately, to guarantee that the resultant interpolation problem is regular, we will need to add further constraints on $M_{A,i}$ and $M_{B,i}$. Thus based on the fact that KL does not change $C_{\text{full},i}$ in bivariate schemes, our performance metric reduces to η_i.

A. Regular bivariate interpolation on rectangular grids

It is well known that the bivariate interpolation problem is regular for any rectangular grid of points $\{x_1, x_2, \cdots, x_K\} \times \{y_1, y_2, \cdots, y_L\}$ satisfying $x_i \neq x_j$ and $y_i \neq y_j$, $\forall i \neq j$. The scheme described next is based on this result. It can be seen as the bivariate interpolation extension of the scheme proposed in [10] based on product codes.

Scheme 2: Assume all workers can equally store m_A partition of A and m_B partition of B and $N = n_A n_B$ such that $K \leq m_A n_B$ and $L \leq m_B n_A$. The CS generates n_B disjoint sets $X_j = \{x_{j,1}, x_{j,2}, \cdots, x_{j,m_A}\}$ for $j = 1, ..., n_B$ with $|X_j| = m_A$ distinct points and n_A disjoint sets $Y_i = \{y_{i,1}, y_{i,2}, \cdots, y_{i,m_B}\}$ for $i = 1, ..., n_A$ with $|Y_i| = m_B$ distinct points. To each worker, the CS assigns one of the $N = n_A n_B$ rectangular grids of points $X_j \times Y_i$ for $j = 1, ..., n_B$, $i = 1, ..., n_A$. We refer to this as worker (j,i). Worker (j,i) stores $\tilde{A}(x)$ $\forall x \in X_j$ and $\tilde{B}(y)$ $\forall y \in y_i$. Consequently, worker (j,i) can compute any of the $m_A m_B$ products $C(x,y) = \tilde{A}(x)\tilde{B}(y)$ with $x \in X_j$ and $y \in Y_i$. Observe that, all together, the set of evaluation points at workers form a rectangular grid of size $m_A n_B \times m_B n_A$. Next, notice that for a given $\hat{y}$, $C(x, \hat{y})$ is a univariate polynomial of degree $K-1$ on x, and thus $C(x, \hat{y})$ can be determined from K evaluations of $C(x, \hat{y})$. Similarly, for a given $\hat{x}$,

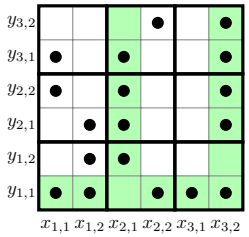


Fig. 1: An example set of responses at the CS for Scheme 2, when $N = 9$, $K = L = 4$, $m_A = m_B = 2$.

$C(\hat{x}, y)$ is a univariate polynomial of degree $L-1$ on y. Observe that for any point $\hat{x} \in X_j$ there are a total of n_A workers (j, i) for $i = 1, ..., n_A$ and each of them can compute m_B distinct evaluations of the univariate polynomial $C(\hat{x}, y)$. Once the first L of these evaluations are received from any worker, the univariate polynomial $C(\hat{x}, y)$ can be reconstructed everywhere. The same also applies to $C(x, \hat{y})$. Moreover, once we have the evaluations of $C(x, y)$ for any rectangular grid of size $K \times L$, either directly received from the workers or via univariate interpolation, bivariate interpolation problem can be solved. However, any computation received at the CS which was already interpolated from previous results is redundant. Figure 1 shows an example set of responses from the workers. Green areas show the decoded rows and columns.Although we need $K \cdot L = 16$ responses and there are 18 received, they are not enough to constitute a 4×4 rectangular grid. Thus, they are not poised. In [10], different heuristics for organizing the computations at workers in order to minimize redundant computations were discussed. Nevertheless, any of these heuristics cannot ensure that the first KL results arriving at the CS are poised. Next, we address this problem.

B. Almost regular bivariate interpolation schemes

In Theorem 1, we show the almost regularity of certain interpolation sets. Then, based on these sets, by introducing a specific computation order at the workers and choosing non-overlapping evaluation points at the workers as opposed to Scheme 2, we propose our storage-efficient interpolation schemes.

Theorem 1: For the interpolation problem of polynomial $C(x, y) = \tilde{A}(x)\tilde{B}(y)$, let

$$\mathcal{U} = \cup_{i \in N} \{x_{i,1}, x_{i,2}, \cdots, x_{i,K}\} \times \{y_{i,1}, y_{i,2}, \cdots, y_{i,L}\}$$

For any set $Z \subset \mathcal{U}$ with $|Z| = KL$, if at least one of the conditions a or b is satisfied, then $\det(M(Z)) \neq 0$ for almost all choices of sets Z and $\mathcal{U}$.

a) If $(x_{i,k}, y_{i,l}) \in Z$ then $(x_{i,m}, y_{i,n}) \in Z$ $\forall m, n$ such that $1 \leq m < k$, $1 \leq n \leq L$ and $(x_{i,k}, y_{i,m}) \in Z$ $\forall m$ such that $1 \leq m \leq l$.

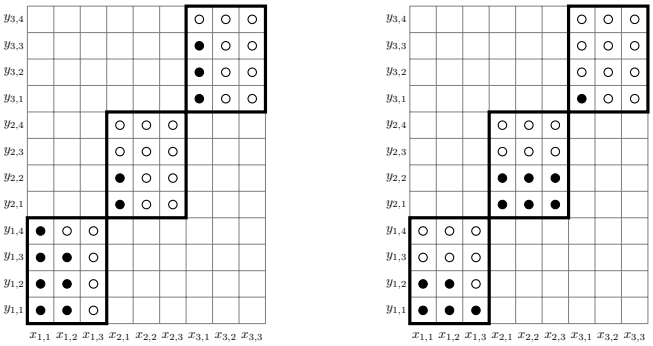
b) If $(x_{i,k}, y_{i,l}) \in Z$ then $(x_{i,m}, y_{i,n}) \in Z$ $\forall m, n$ such that $1 \leq m \leq K$, $1 \leq n < l$ and $(x_{i,m}, y_{i,l}) \in Z$ $\forall m$ such that $1 \leq m \leq k$ □

Due to space restrictions, we give the proof of this result in the extended version [14]. It is based on the Taylor series expansion of the determinant of $M(Z)$ [15].

Scheme 3: In this case, we require the computations at workers to be done in a vertical order, i.e., without completing L computations sharing common x coordinate, computations from the other columns are not allowed. We choose the evaluation points to satisfy condition a of Theorem 1. This is possible in two ways:

1) Store a single partition of A, and any number of partitions of B, i.e., $m_{A,i} = 1$ and $m_{B,i} \leq L$, or
2) Store full matrix B, and any number of partitions of A, i.e., $m_{A,i} \geq 1$ and $m_{B,i} = L$.

As usual, the CS sends to worker i, $m_{A,i}$ evaluations of $\tilde{A}(x)$ at points $\{x_{i,1}, x_{i,2}, \cdots, x_{i,m_{A,i}}\}$ and $m_{B,i}$ evaluations of $\tilde{B}(y)$ at points $\{y_{i,1}, y_{i,2}, \cdots, y_{i,m_{B,i}}\}$. In the computation phase, the worker i computes cross-products $\tilde{A}(x_{i,j})\tilde{B}(y_{i,k})$ with increasing order of (j, k). The increasing order is defined as $(j, k) \leq (\hat{j}, \hat{k})$ if $[(j < \hat{j}) \vee (j = \hat{j} \wedge k \leq \hat{k})]$. Such computation order guarantees condition a of Theorem 1 is satisfied.



(a) (b)

Fig. 2: Example sets of decodable responses at the CS for Scheme 3 (a) and Scheme 4 (b), when $N = 3$, $m_A = K = 3$, $m_B = L = 4$.

Scheme 4: In this case, the computations in the workers must be done in a horizontal order, i.e., without completing K computations sharing common y coordinate, computations from the other rows are not allowed. We choose the evaluation points to satisfy condition b of Theorem 1. This is possible in two ways:

1) Store a single partition of B, and any number of partitions of A, i.e., $m_{A,i} \leq K$ and $m_{B,i} = 1$, or
2) Store full matrix A, and any number of partitions of B, i.e., $m_{A,i} = K$ and $m_{B,i} \geq 1$

Similar to Scheme 3, the CS sends to worker i, $m_{A,i}$ evaluations of $\tilde{A}(x)$ and $m_{B,i}$ evaluations of $\tilde{B}(y)$. In the computation phase, worker i computes cross-products $\tilde{A}(x_{i,j})\tilde{B}(y_{i,k})$ with increasing order of (j, k). In this case, the increasing order is defined as $(j, k) \leq (\hat{j}, \hat{k})$ if $[(k < \hat{k}) \vee (k = \hat{k} \wedge j \leq \hat{j})]$. Such computation order guarantees condition b of Theorem 1 is satisfied. An example set of responses from the workers can be seen in Figure 2a and Figure 2b for Scheme 3 and Scheme 4 respectively. Note that bold 3×4 rectangles represents workers.

Note that Scheme 3 and Scheme 4 are mutually exclusive. For a matrix multiplication task, we need to choose one of them and use it in all workers. Using Scheme 3 in some workers, and using Scheme 4 in other ones does not guarantee that any of the conditions of Theorem 1 will be satisfied.

How to choose between them depends on the dimensions of the matrix partitions. Given a fixed sum storage capacity $rsM_{A,i} + scM_{B,i}, \forall i \in N$, if choosing $m_{A,i}$ and $m_{B,i}$ such that $(m_{A,i} = 1, m_{B,i} \leq L)$ or $(m_{A,i} \geq 1, m_{B,i} = L)$ is satisfied produces a larger $m_{A,i}m_{B,i}$ than choosing $m_{A,i}$ and $m_{B,i}$ such that $(m_{A,i} \leq K, m_{B,i} = 1)$ or $(m_{A,i} = K, m_{B,i} \geq 1)$ is satisfied, then we choose Scheme 3, otherwise Scheme 4 should be used. Observe that when the partitions of B are smaller than those of A, reducing $m_{A,i}$ by 1 will increase $m_{B,i}$ at least by 1. Similarly, when partitions of A are smaller than those of B, reducing $m_{B,i}$ by 1 will increase $m_{A,i}$ at least by 1. Hence, if $rs > sc$, then satisfying $(m_{A,i} = 1, m_{B,i} \leq L)$ or $(m_{A,i} \geq 1, m_{B,i} = L)$ is easier and Scheme 3 should be used, otherwise Scheme 4 should be preferred.

Note that unlike, Scheme 2, in Scheme 3 and 4, the storage capacities of the workers do not need to be the same. This makes the proposed coding scheme more useful for the cases with heterogeneous worker storage capacities.

V. Numerical Results

In this section, by running Monte Carlo simulations, we compare the introduced schemes in terms of average computation time, which is defined as the time passed until workers cumulatively complete enough number of computations to decode the result, under different memory availability. Since the order of the polynomial to be interpolated is common for all schemes, in the comparison, encoding and decoding times are discarded. Moreover, we assume the communication time is negligable compared to the computation time. We use shifted exponential model for finishing times of computations, which is typically used in coded computation problems [1]. 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 a worker can complete a unit computation. The smaller scale λ means more variance, and thus more heterogeneous computation speeds among the workers. To cover more heterogeneous cases per experiment, we choose $\nu = 0.01$ and $\lambda = 0.1$.

We assume that the size of partitions A and B are equal and $m_{A,i} = m_A$ and $m_{B,i} = m_B \ \forall i \in N$ for a fair comparison since this is required by Scheme 2. Thus $\eta_i = \eta$, $\forall i \in [N]$ We take $K = L = 10$ and $N = 15$. For each memory value, we run 10^4 experiments. The results are given in Figure 3. We plot the expected computation times starting from $m_A + m_B = 6$ for Scheme 2, from $m_A + m_B = 8$ for Scheme 3 and 4 and from $m_A + m_B = 14$ for Scheme 1 since for $N = 15$ workers, these values are the minimum memory values that can complete $KL = 100$ computations. We observe that especially for very small memory values, Scheme 2 is the fastest scheme. That is because, in Scheme 2, since there is no restriction on the allocation of m_A and m_B other than $m_A + m_B$ is constant, m_A and m_B can be chosen equal or close to each other meaning larger η and thus more efficient use of the memory. However, starting from the intermediate memory values, Scheme 3 and 4 start beating all other schemes. This is because the number of maximum computations, η, of Scheme 3 and 4 becomes comparable with that of Scheme 2, and there are no useless computations in Scheme 3 and 4. Moreover, if $m_A + m_B > 20$, Scheme 2, 3 and 4 do not improve further since then both A and B can be stored in one worker, reaching the maximum η possible per worker. For Scheme 1, this limit is 200, and until then, Scheme 3 and 4 beat Scheme 1. After that point, their performances are the same.

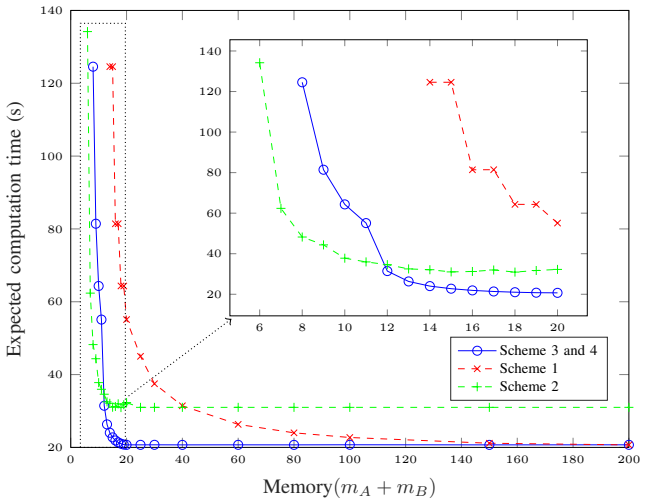


Fig. 3: Average computation times of univariate and bivariate polynomial codes

VI. Conclusion

In this work, we studied the memory-efficient exploitation of stragglers in distributed matrix multiplication with workers allowed to have heterogeneous computation and storage capacity. We proposed bivariate polynomial coding schemes allowing efficient use of workers' memories.

The bivariate polynomial coding poses the problem of invertibility of the interpolation matrix. 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 central server may not be useful since the information they provide is already obtained from previous responses. In order to tackle this problem, then, we showed that as long as workers follow a specific computation order, for almost every choice of the interpolation points, the interpolation matrix is invertible. Based on this, we proposed Scheme 3 and 4 solving the problem of redundant computations. The proof of the almost regularity in these 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. Our work is built on polynomial codes [2] and it can be extended to the cases of arbitrary matrix partitioning schemes, e.g. PolyDot codes, entangled polynomial codes. This extension may be an interesting future work. Extending the scheme to private matrix multiplication would be also an interesting line of work.

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, 2018.

[2] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4403–4413.

[3] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv:1801.10292 [cs, math]*, 2018. [Online]. Available: http://arxiv.org/abs/1801.10292

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

[5] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized PolyDot codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1585–1589.

[6] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch computation," *arXiv:1909.13873 [cs, math]*, 2019. [Online]. Available: http://arxiv.org/abs/1909.13873

[7] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[8] N. Ferdinand and S. Draper, "Hierarchical coded computation," *arXiv:1806.10250 [cs, math]*, 2018. [Online]. Available: http://arxiv.org/abs/1806.10250

[9] M. M. Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 8177–8181.

[10] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1988–1992.

[11] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2418–2422.

[12] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3492–3496.

[13] S. Kiani, N. Ferdinand, and S. C. Draper, "Hierarchical coded matrix multiplication," *arXiv:1907.08818 [cs, math]*, 2019. [Online]. Available: http://arxiv.org/abs/1907.08818

[14] 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. [Online]. Available: https://arxiv.org/abs/2001.07227

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