Bivariate Polynomial Codes for Secure Distributed Matrix Multiplication

Burak Hasırcıoğlu, Jesús Gómez-Vilardebó, and Deniz Gündüz

Abstract

We consider the problem of secure distributed matrix multiplication (SDMM). Coded computation has been shown to be an effective solution in distributed matrix multiplication, both providing privacy against workers and boosting the computation speed by efficiently mitigating stragglers. In this work, we present a non-direct secure extension of the recently introduced bivariate polynomial codes. Bivariate polynomial codes have been shown to be able to further speed up distributed matrix multiplication by exploiting the partial work done by the stragglers rather than completely ignoring them while reducing the upload communication cost and/or the workers' storage's capacity needs. We show that, especially for upload communication or storage constrained settings, the proposed approach reduces the average computation time of SDMM compared to its competitors in the literature.

I. Introduction

Matrix multiplication is a fundamental building block of many applications in signal processing and machine learning. For some applications, especially those involving massive matrices and stringent latency requirements, matrix multiplication in a single computer is infeasible, and distributed solutions need to be adopted. In such scenarios, the full multiplication task is first partitioned into smaller sub-tasks, which are then distributed across dedicated *workers*.

In this work, we address two main challenges in distributed matrix multiplication. The first one is referred to as the *stragglers* problem, which refers to unresponsive or slow workers.

Burak Hasırcıoğlu and Deniz Gündüz are with the Department of Electrical and Electronic Engineering, Imperial College London, UK. E-mail: {b.hasircioglu18, d.gunduz}@imperial.ac.uk

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

A preliminary version of this paper has been accepted for a presentation in 2021 IEEE International Symposium on Information Theory (ISIT) [1].

If completing the full task requires the completion of the computations assigned to all the workers, then straggling workers become a significant bottleneck. To avoid stragglers, additional redundant computations can be assigned to workers. It has been recently shown that the use of error-correcting codes, by treating the slowest workers as erasures instead of simply replicating tasks across workers, significantly lowers the overall computation time [2]. In the context of straggler mitigation, polynomial-type codes are studied in [3]–[6]. In these schemes, matrices are first partitioned and encoded using polynomial codes at the master server. Then, workers compute sub-products by multiplying these coded partitions and send the results back to the master for decoding. The minimum number of sub-tasks required to decode the result is referred to as the *recovery threshold* and denoted by R_{th}. All these works assume that only one sub-product is assigned to each worker, and therefore, any work done by the workers beyond the fastest R_{th} is completely ignored. This is sub-optimal, particularly when the workers have similar computational speeds. This problem is addressed by the *multi-message approaches* in [7]–[10]. In this works, multiple sub-products are assigned to each worker, and the result of each sub-product is communicated to the master as soon as it is completed. This results in faster completion of the full computation as it allows to exploit partial computations completed by stragglers. Moreover, the multi-message approach makes finishing the task possible even if there are not as many available workers as the recovery threshold. However, as discussed in [10], a direct extension of polynomial-type codes to the multi-message setting by simply assigning multiple sub-products to the workers increase the *upload communication costs*, which is defined as the number of bits sent from the master to each worker, or equivalently, the storage required per worker. To combat this effect, product codes are proposed in [7] for the multi-message distributed matrix multiplication problem. However, with product codes, every sub-product is not equally useful while decoding the full-product, i.e., they are not one-to-any replaceable, which degrades their performance. The *bivariate polynomial codes* are introduced in [10] to address this issue, achieving a better trade-off between the upload cost and average computation time.

The second challenge we tackle in this paper is privacy. The multiplied matrices may contain sensitive information, and sharing these matrices, even partially, with the workers may cause a privacy breach. Moreover, several workers can exchange information with each other in some settings to learn about the multiplied matrices. Such a collusion may result in a leakage even if no information is revealed to individual workers. The first application of polynomial codes to privacy-preserving distributed matrix multiplication is presented in [11]. To hide the matrices

from the workers, random matrix partitions are created, and linearly encoded together with the true matrix partitions using polynomial codes. This requires increasing the degree of the encoding polynomial and thus increasing the recovery threshold. The recovery threshold has been improved in subsequent works [12], [13], by carefully choosing the degrees of the encoding monomials so that the resultant decoding polynomial contains the minimum number of additional coefficients. In [14]–[16], lower recovery threshold values than [13] are obtained by using different matrix partitioning techniques and different choices of encoding polynomials, but this is achieved at the expense of a considerable increase in the upload cost. In [17], a novel coding approach for distributed matrix multiplication is proposed based on polynomial evaluation at the roots of unity in a finite field. It has constant time decoding complexity and a low recovery threshold compared to traditional polynomial-type coding approaches, but the sub-tasks are not one-to-any replaceable and its straggler mitigation capability is limited. In [18], a multi-message approach is proposed for SDMM by using rateless codes. Computations are assigned adaptively in rounds, and in each round, workers are classified into clusters depending on their computation speeds. Results from a worker in a cluster are useful for decoding only if the results of all the sub-tasks assigned to that cluster and also to the fastest cluster are collected, making computations not one-to-any replaceable. Still, the strategy exhibited good average computation times by estimating and adapting to the computation speeds of the workers.

In this work, we propose Secure Bivariate Polynomial (SBP) codes, for the multi-message, straggler-resistant, SDMM task based on bivariate Hermitian polynomial codes. We show that, especially, under a limited upload cost budget, or when the number of fast workers is limited, SBP codes outperform other schemes in the literature in terms of the average computation time. We also show that this scheme retains its low average computation time when the computation speeds of the workers significantly differ, i.e., heterogeneous computation speeds, or when they are close to each other, i.e., homogeneous computation speeds. In addition, we also propose an extension of GASP codes [13] to the multi-message setting and evaluate its performance. We show that when a sufficiently large upload cost budget is available, employing this proposed extension could considerably lower the average computation time of the SDMM task.

II. Problem Setting

We study distributed matrix multiplication with strict privacy requirements. The elements of our matrices are in a finite field $\mathbb{F}$, and we denote the size of the finite field by q. The *master*

wants to multiply statistically independent matrices $A \in \mathbb{F}^{r\times s}$ and $B \in \mathbb{F}^{s\times t}$, $r, s, t \in \mathbb{Z}^+$, with the help of N dedicated *workers*, which possibly have heterogeneous computation speeds and storage capacities.

To offload the computation to several workers, the master divides the multiplication task into smaller sub-tasks, which are then assigned to workers. The master partitions A into K sub-matrices as $A = \begin{bmatrix} A_1^T & A_2^T & \cdots & A_K^T \end{bmatrix}^T$, where $A_i \in \mathbb{F}^{\frac{r}{K}\times s}$, $\forall i \in [K] \triangleq \{1, 2, \ldots, K\}$, and B into L sub-matrices as $B = \begin{bmatrix} B_1 & B_2 & \cdots & B_L \end{bmatrix}$, where $B_j \in \mathbb{F}^{s\times \frac{t}{L}}$, $\forall j \in [L]$. The master sends coded versions, i.e., linear combinations, of these partitions to the workers. We assume that there is an *upload cost* constraint per worker, denoted by u_i for worker i, which limits the maximum number of bits that can be transmitted from the master to each worker. This upload cost is a limiting factor on the number of coded partitions of A, denoted by $m_{A,i}$, and of B, denoted by $m_{B,i}$, that can be sent to each worker. More specifically, for worker i, $m_{A,i}$ and $m_{B,i}$ must satisfy $(m_{A,i}rs/K + m_{B,i}st/L)\log_2(q) \leq u_i$. Provided that they comply with the upload cost constraint, $m_{A,i}$ and $m_{B,i}$ are chosen depending on the underlying coding scheme and the master sends coded partitions $\tilde{A}_{i,k}$ and $\tilde{B}_{i,l}$ to worker i, where $i \in [N]$, $k \in [m_{A,i}]$ and $l \in [m_{B,i}]$. For simplicity, we describe a static setting, in which all the coded matrices are sent to the workers before they start computations. In a more dynamic scenario, matrix partitions can be delivered when they are needed, which would reduce the memory required at the workers. The workers multiply the received coded partitions of A and B as instructed by the underlying coding scheme and send the result of each computation to the master as soon as it is completed. Once the master receives a number of computations equal to the *recovery threshold*, R_{th}, it can decode the desired multiplication AB.

In our threat model, all the workers are honest but curious. They follow the protocol, but they can use the received encoded matrices to obtain information about the original matrices, A and B. We also assume that any T workers can collude, i.e., exchange information among themselves. Our privacy requirement is that no T workers are allowed to gain any information about the content of the multiplied matrices. That is,

$$I\left(A, B; \{\tilde{A}_{i,k}, \tilde{B}_{i,l} \mid i \in \mathcal{N}, k \in [m_{A,i}], l \in [m_{B,i}]\}\right) = 0,$$

where I is the mutual information and $\mathcal{N}$ is the any subset of $[N]$ with cardinality at most T.

Under this setting, the main problem we attempt to solve in this work is minimizing the *average computation time*, which is defined as the time required for the master to collect sufficiently many

computations to decode the desired computation AB. We assume that the workers' computation speeds can be homogeneous, i.e., the average speed of each available worker is close to each other, or heterogeneous, in which the average speeds of the workers vary. Workers can also straggle, i.e., become unresponsive temporarily.

III. Extension of Bivariate Polynomial Codes for Secure DMM

As a first attempt to improve the upload cost efficiency of SDMM, in this section, we provide the naive extension of bivariate polynomial codes proposed in [10] to SDMM. In [10], the partitioning of the matrices is as described in Section II and the two encoding polynomials are generated as

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

$$B(y) = B_1 + B_2 y + \cdots + B_L y^{L-1}. \tag{2}$$

Therefore, at the master, the goal is to interpolate the following polynomial.

$$A(x)B(y) = \sum_{i=1}^{K} \sum_{j=1}^{L} A_i B_j x^{i-1} y^{j-1}. \tag{3}$$

Since worker $i \in [N]$ can store $m_{A,i}$ partitions of A and $m_{B,i}$ partitions of B, the master sends the first $m_{A,i}$ derivatives of $A(x)$ and first $m_{B,i}$ derivatives of $B(y)$, evaluated at x_i and y_i, respectively, which are evaluation points of the encoding polynomials chosen distinct for each worker. Each worker multiplies the received encoded partitions of $A(x)$ and $B(y)$ following a specific order from the smaller-order derivatives to larger-order derivatives and sends the results of each computation as soon as it is finished. Then, once the master receives KL computations from the workers, it instructs all the workers to stop and starts decoding $A(x)B(y)$.

In order to provide a simple direct extension of this scheme to SDMM in which T worker collude, we limit the analysis to the case $m_{A,i} = m_A$ and $m_{B,i} = m_B$, $\forall i \in [N]$. Thus, from a security point of view, each worker learns m_A coded partitions of A and m_B coded partitions of B and since up to T workers collude, in total, $m_A T$ coded partitions of A and $m_B T$ coded partitions of B are leaked to the workers. To protect such a leakage, we need to add $m_A T$ and $m_B T$ random matrix partitions to $A(x)$ and $B(y)$, respectively. Thus, the encoding polynomials for this naive extension of bivariate polynomial codes to SDMM becomes

$$A(x) = A_1 + A_2 x + \cdots + A_K x^{K-1} + \sum_{i=1}^{m_A T} R_i x^{K+i-1}, \tag{4}$$

$$B(y) = B_1 + B_2 y + \cdots + B_L y^{L-1} + \sum_{i=1}^{m_B T} S_i x^{L+i-1}, \tag{5}$$

where R_i and S_i are matrix partitions chosen uniformly at random from the elements of $\mathbb{F}_q$. Therefore, the polynomial to be interpolated at the master becomes

$$\begin{aligned} A(x)B(y) = &\sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{i-1} y^{j-1} + \sum_{i=1}^{K}\sum_{j=L+1}^{m_B T} A_i S_j x^{i-1} y^{j-1} \\ &+ \sum_{i=K+1}^{m_A T}\sum_{j=1}^{L} R_i B_j x^{i-1} y^{j-1} + \sum_{i=K+1}^{m_A T}\sum_{j=L+1}^{m_B T} R_i S_j x^{i-1} y^{j-1}. \end{aligned} \tag{6}$$

Therefore, considering the number of monomials of $A(x)B(y)$ in (6), $R_{th} = (K + m_A T)(L + m_B T)$ evaluations of $A(x)B(y)$ are needed to interpolate it, which has a quadratic dependence on T.

Observe that in this naive extension, for a worker to provide $m = m_A m_B$ computations, uploading m_A coded partitions of A and m_B coded partitions of B are enough. This means that the upload cost of the scheme is on the order of $\sqrt{m}$. However, the price we pay for such a reduced upload cost is a quadratic dependence of the recovery threshold on the number of colluding workers. Such dependence on T may quickly become restrictive for typical T values and hence, the benefits of the naive extension of bivariate polynomial codes to SDMM may be out-weighted by its drawbacks. Thus, we need more sophisticated schemes that can keep this low upload cost with a better scaling behaviour for the recovery threshold with respect to m and T. In the next section, we present our proposed solution for such a problem.

IV. Secure Bivariate Polynomial (SBP) Codes

Our coding scheme is based on bivariate polynomial codes [10]. Compared to their univariate counterparts, bivariate polynomial codes allow workers to complete more sub-tasks under the same upload cost budget, which, in turn, improves the average computation time and helps to satisfy the privacy requirements.

A. Encoding

In SBP coding scheme, coded matrices are generated by evaluating the following polynomials and their derivatives:

$$A(x) = \sum_{i=1}^{K} A_i x^{i-1} + \sum_{i=1}^{T} R_i x^{K+i-1}, \tag{7}$$

$$B(x,y)=\sum_{i=1}^{L}B_i y^{i-1}+\sum_{i=1}^{T}\sum_{j=1}^{m}S_{i,j}x^{K+i-1}y^{j-1}, \tag{8}$$

where $m \leq L$ is the maximum number of sub-tasks any worker can complete. Matrices $R_i \in \mathbb{F}_q^{\frac{r}{K}\times s}$ and $S_{i,j} \in \mathbb{F}_q^{s\times\frac{t}{L}}$ are independent and uniform randomly generated from their corresponding domain for $i \in [T]$ and $j \in [m]$. For each worker i, the master evaluates $A(x)$ at x_i and the derivatives of $B(x,y)$ with respect to y up to the order $m-1$ at (x_i,y_i). We only require these evaluation points to be distinct. Thus, the master sends to worker i, $A(x_i)$ and $\mathcal{B}_i=\{B(x_i,y_i),\partial_1 B(x_i,y_i),\ldots,\partial_{m-1}B(x_i,y_i)\}$, where ∂_i denotes the i^{th} partial derivative with respect to y. Thus, we require $m_{A,i}=1$ and $m_{B,i}=m$.

In (7) and (8), the role of R_i's and $S_{i,j}$'s is to mask the actual matrix partitions for privacy. The following theorem states that the evaluations of $A(x)$, $B(x,y)$ and its derivatives do not leak any information about A and B to any T colluding workers.

Theorem 1. *For the encoding scheme described above, we have*

$$I(A,B;\{A(x_i),\mathcal{B}_i : i\in\mathcal{N}\})=0, \tag{9}$$

$\forall \mathcal{N}\subset[N]$ *such that* $|\mathcal{N}|\leq T$.

Proof. Since A and B are independent, we have

$$I(A,B;\{A(x_i),\mathcal{B}_i \,:\, i\in\mathcal{N}\}) = I(A;\{A(x_i)\,:\,i\in\mathcal{N}\})+I(B;\{\mathcal{B}_i\,:\,i\in\mathcal{N}\}). \tag{10}$$

Let us first bound $I(A;\{A(x_i)|i\in\mathcal{N}\})$ as follows.

$$\begin{aligned}
I\left(A;\{A(x_i):i\in\mathcal{N}\}\right) &= H\left(\{A(x_i):i\in\mathcal{N}\}\right)-H\left(\{A(x_i):i\in\mathcal{N}\}|A\right) && (11)\\
&= H\left(\{A(x_i):i\in\mathcal{N}\}\right)-H\left(\{R_i:i\in[T]\}|A\right) && (12)\\
&\overset{(a)}{=} H\left(\{A(x_i):i\in\mathcal{N}\}\right)-T\frac{rs}{K}\log(q) && (13)\\
&\overset{(b)}{\leq} \sum_{i=1}^{|\mathcal{N}|}H(A(x_i))-T\frac{rs}{K}\log(q) && (14)\\
&= |\mathcal{N}|\frac{rs}{K}\log(q)-T\frac{rs}{K}\log(q)\overset{(c)}{\leq}0, && (15)
\end{aligned}$$

where (a) follows from the fact that R_i's are independent from each other and from A, (b) is due to the fact that joint entropy of several random variables is upper bounded by the sum of the individual entropies of these random variables and (c) is due to $\mathcal{N}\leq T$.

We can bound $I(B;\{\mathcal{B}_i : i \in \mathcal{N}\})$ similarly as follows.

$$I\left(B;\{\mathcal{B}_i : i \in \mathcal{N}\}\right) = H\left(\{\mathcal{B}_i : i \in \mathcal{N}\}\right) - H\left(\{\mathcal{B}_i : i \in \mathcal{N}\}|B\right) \tag{16}$$

$$= H\left(\{\mathcal{B}_i : i \in \mathcal{N}\}\right) - H(\{S_{i,j} : i \in [T], j \in [m]\}|B) \tag{17}$$

$$= H\left(\{\mathcal{B}_i : i \in \mathcal{N}\}\right) - Tm\frac{st}{L}\log(q) \tag{18}$$

$$\leq \sum_{i=1}^{|\mathcal{N}|}\sum_{j=1}^{m} H(B(x_i,y_j)) - Tm\frac{st}{L}\log(q) \tag{19}$$

$$= |\mathcal{N}|m\frac{st}{L}\log(q) - Tm\frac{st}{L}\log(q) \leq 0. \tag{20}$$

The claim follows by substituting (15) and (20) into (10). □

B. Computation

Worker i multiplies $A(x_i)$ and $\partial_{j-1}B(x_i,y_i)$ with the increasing order of $j \in [m]$. That is, j^{th} completed computation is $A(x_i)\partial_{j-1}B(x_i,y_i)$. As soon as each multiplication is completed, its result is communicated back to the master.

C. Decoding

After collecting sufficiently many computations from the workers, the master can interpolate $A(x)B(x,y)$. Note that, in our scheme, every computation is equally useful; that is, the sub-tasks are one-to-any replaceable. In the following theorem, we give the recovery threshold expression, which specifies the minimum number of required computations and characterizes the probability of decoding failure, i.e., bivariate polynomial interpolation, due to the use of a finite field.

Theorem 2. *Assume the evaluation points (x_i, y_i) are chosen uniform randomly over the elements of $\mathbb{F}$. If the number of computations of sub-tasks received from the workers, which obey the computation order specified in Subsection IV-B is greater than the recovery threshold $R_{th} \triangleq (K+T)L + m(K+T-1)$, then with probability at least $1 - d/q$, the master can interpolate the unique polynomial $A(x)B(x,y)$, where*

$$d \triangleq \frac{m}{2}\left(3(K+T)^2 + m(K+T) - 8K - 6T - m + 3\right) + \frac{(K+T)L}{2}\left(K+L+T-2\right). \tag{21}$$

We give the proof sketch of Theorem 2 in Section VII. Theorem 2 states that we can make the probability of failure arbitrarily small by increasing the order q of the finite field.

Theorem 3. *The total upload cost of the SBP coding scheme is* $N\left(rs/K + mst/L\right)\log_2(q)$ *bits.*

Proof. The SBP coding scheme assigns m computations to each worker, by sending one coded partition of A and m coded partitions of B. Remember that each coded partition of A is a matrix of size $\frac{r}{K} \times s$ and each coded partition of B is a matrix of size $s \times \frac{t}{L}$. Since there are N workers, the master uploads $N\left(rs/K + mst/L\right)$ elements of the field $\mathbb{F}$. Since, in total, there are q elements in $\mathbb{F}$, the total upload cost is $N\left(rs/K + mst/L\right)\log_2(q)$ bits. □

Remark 1. *The SBP scheme, as described in this section, does not exploit any parameter of the underlying statistical model of the workers' speeds. Under a total upload cost constraint, if no prior information about the computation speeds of the workers is available, then assigning more computation load,* m *to every worker is a favorable approach. Although this increases the recovery threshold as well, i.e., the term* $m(K + T - 1)$*, the faster workers do not run out of computations easily, avoiding the slowest workers dominating the computation time. The benefit of this prevails over the detriment due to the increase in the recovery threshold. Surely, if prior information about the computation speeds of the workers is available, we could exploit it assigning more computations to the faster workers and fewer computations to the slower workers, which would result in fewer number of coded partitions leaked to the colluding workers. In such a case, the recovery threshold would be lower and hence, would further increase the protection against stragglers. Please note that to extend our scheme to such a scenario, we should still satisfy a constraint that the maximum number of assigned computations to a worker is less than* L*. However, the SBP scheme has been designed as agnostic to the delay model of the workers and specifically to maximize the number of sub-tasks delivered by a worker under an upload cost constraint. Thanks to the extra computations at workers, we show in simulation results that a model-independent version of SBP scheme is enough to beat model-dependent schemes such as the coding scheme in [18]. Thus, we expect the SBP coding scheme to work for large varieties of statistical models of the worker's speeds.*

V. Extension of GASP codes to multi-message setting

State of the art schemes in SDMM [11], [12] are combined and improved in [13], referred to as GASP codes. Originally, GASP codes are designed for the single-message scenario, in which each worker is assigned a single computation task. In this section, we extend the GASP

codes to the multi-message setting, which we call multi-message GASP (MM-GASP) scheme. The encoding polynomials for the GASP codes are

$$A(x) = \sum_{i=1}^{K} A_i x^{\alpha_i} + \sum_{i=1}^{T} R_i x^{\alpha_{K+i}},$$
$$B(x) = \sum_{i=1}^{L} B_i x^{\beta_i} + \sum_{i=1}^{T} S_i x^{\beta_{L+i}},$$

where R_i's and S_i's are random matrix partitions, and α_i's and β_i's are determined such that A_iB_j, $\forall i \in [K], \forall j \in [L]$ can be decoded and the number of monomials whose coefficients consist of undesired terms, such as multiplications involving R_i's and S_i's, in $A(x)B(x)$, are minimized. We do not cover the details of how α_i's and β_i's are determined, but as a result of the process detailed in [13], the recovery threshold becomes

$$R_{th}^{GASP}(K,L,T) = \begin{cases} KL + K + L & 1 = T < L \leq K \\ KL + K + L + T^2 + T - 3 & 1 < T < L \leq K \\ (K+T)(L+1) - 1 & L \leq T < K \\ 2KL + 2T - 1 & L \leq K \leq T. \end{cases} \tag{22}$$

For the extension of GASP codes to the multi-message setting, i.e., MM-GASP, we assign $m > 1$ tasks to each worker. Thus, a worker can see m evaluations of $A(x)$ and $B(x)$, and any T colluding workers can see mT evaluations. Thus, to make the scheme secure against T colluding workers, we need to add mT random matrix partitions to each encoding polynomial, instead of T. Thus, we have

$$A(x) = \sum_{i=1}^{K} A_i x^{\alpha_i} + \sum_{i=1}^{mT} R_i x^{\alpha_{K+i}},$$
$$B(x) = \sum_{i=1}^{L} B_i x^{\beta_i} + \sum_{i=1}^{mT} S_i x^{\beta_{L+i}}.$$

Theorem 4. *The recovery threshold of MM-GASP is given by the following expression.*

$$R_{th}^{MM-GASP}(K,L,T) = \begin{cases} KL + K + L & 1 = mT < L \leq K \\ KL + K + L + (mT)^2 + mT - 3 & 1 < mT < L \leq K \\ (K+mT)(L+1) - 1 & L \leq mT < K \\ 2KL + 2mT - 1 & L \leq K \leq mT. \end{cases} \tag{23}$$

Proof. We can define $\tilde{T} = mT$ and write

$$A(x) = \sum_{i=1}^{K} A_i x^{\alpha_i} + \sum_{i=1}^{\tilde{T}} R_i x^{\alpha_{K+i}}, \quad B(x) = \sum_{i=1}^{L} B_i x^{\beta_i} + \sum_{i=1}^{\tilde{T}} S_i x^{\beta_{L+i}}. \tag{24}$$

Now let us consider

$$\begin{aligned} A(x)B(x) = &\sum_{i=1}^{K}\sum_{j=1}^{L} A_i B_j x^{\alpha_i+\beta_i} + \sum_{i=1}^{K}\sum_{j=1}^{\tilde{T}} A_i S_j x^{\alpha_i+\beta_{L+i}} \\ &+ \sum_{i=1}^{\tilde{T}}\sum_{j=1}^{L} R_i B_j x^{\alpha_{K+i}+\beta_i} + \sum_{i=1}^{\tilde{T}}\sum_{j=1}^{\tilde{T}} R_i S_j x^{\alpha_{K+i}+\beta_{L+i}}. \end{aligned} \tag{25}$$

According to the proof of (22) in [13], α_i's and β_i's are chosen such that: 1) from the evaluations of $A(x)B(x)$, A_iB_j's $\forall i \in [K], j \in [L]$ are decodable and 2) the number of monomials in $A(x)B(x)$ whose coefficients are undesired terms are minimized. For this, we only need to consider the structure of $A(x)B(x)$ and m itself is not related other than determining the value of $\tilde{T}$. Therefore, the problem reduces to deriving the recovery threshold of a classical GASP coding scheme when $\tilde{T}$ workers collude, which is $R_{th}^{GASP}(K, L, \tilde{T})$ by (22). If we substitute $\tilde{T} = mT$, then we obtain (23).

□

Remark 2. *In multi-message univariate polynomial coding schemes, such as in MM-GASP codes we have just introduced, if a worker is assigned m sub-tasks, then m coded partitions of both A and B are required. Thus, the total upload cost of MM-GASP is $Nm\,(rs/K + st/L)\log_2(q)$ bits, which is larger than that of SBP coding scheme.*

The recovery thresholds of the MM-GASP codes and SBP codes can be compared as a function of the number of coded partitions m, by direct inspection of the recovery thresholds in (23) and Theorem 2. Observe that SBP coding scheme's recovery threshold is smaller than that of the MM-GASP code if $L \leq mT < K$, and $K \leq TL + 1$, which is satisfied as K and L become close to each other, or, if $L \leq K \leq mT$, and $(K - T)(L - m) \geq (1 - m)$ is satisfied. In Fig. 1, we provide the recovery thresholds of the two schemes as a function of the number of computations allocated to each worker for $K = L = 100$ and $T = 30$.

We note that such a comparison may only be meaningful in the unlimited upload cost budget scenario. Otherwise, comparing the recovery thresholds for the same m might be misleading since, for a given upload cost constraint, each scheme provides a different number of sub-tasks,

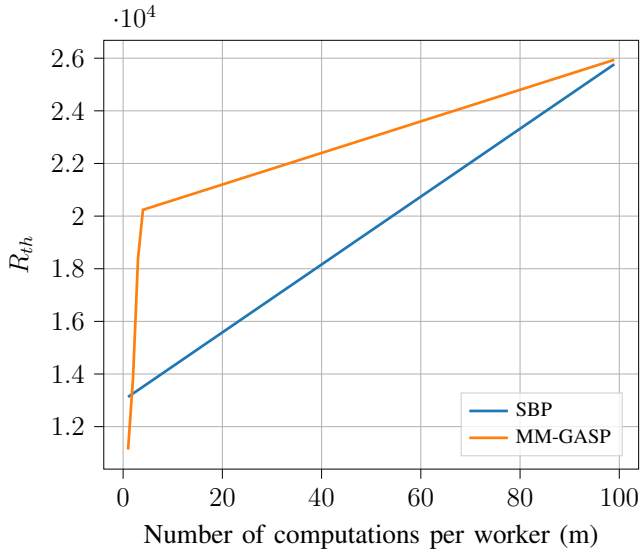

Fig. 1: R_{th} vs. the number of computations assigned to each worker for SBP coding scheme and the MM-GASP scheme for $K = L = 100$ and $T = 30$.

m, to workers, as detailed in Theorem 3 and Remark 2, for SBP and for MM-GASP, respectively. We provide further discussion on this issue in the next section, see Fig. 3, where we show the recovery thresholds as a function of the total upload cost budget for a scenario with $K = L = 100$ and $T = 30$.

VI. Simulation Results and Discussion

In this section, we compare SBP codes with MM-GASP and the rateless coding scheme proposed in [18] in terms of the trade-off between the average computation time (ACT) and the total upload cost budget (UCB), under the scenarios with heterogeneous and homogeneous workers.

The comparison between the MM-GASP scheme and SBP coding scheme is direct, as both are based on the same set of assumptions. They achieve different recovery thresholds as a function of the L, M, T and m, but they both assume that the coded submatrices are uploaded only once before the computations start, no prior knowledge of the computation speeds of workers is needed or can be exploited, and the first R_{th} results received from any subset of the workers allow recovering the desired computation. However, the setting and the assumptions in [18] are slightly different. In the rateless coding scheme proposed in [18], computations are organized in rounds. If the speeds of the workers are not already known, in the first round, every worker is assigned one computation to estimate their speeds. Then, based on the known or estimated

speeds, workers are grouped into c clusters, such that the workers with similar speeds are in the same cluster. We denote by n_u the number of workers belonging to cluster u, $u \in [c]$. In each round, for any computation within a cluster to be useful for decoding, we need that all the workers in that cluster and also all workers in cluster one, which is special, to finish their assigned tasks. Once all the workers in cluster u and cluster 1 finish their tasks, they provide d_u, and d_1 useful computations to the master, where $d_1 = \lfloor (n_1 - 2T + 1)/2 \rfloor$ and $d_u = \lfloor (n_u - T + 1)/2 \rfloor$ for $2 \leq u \leq c$. No further synchronization is needed among clusters, and a new task can be assigned to a worker as soon as it finishes its assigned task. Once $KL(1+\epsilon)$ useful computations are collected by the master from different clusters across multiple rounds, the decoding procedure can start. Here, ϵ is the overhead due to the Fountain codes used in [18], which in our simulations is set to 0.05. The performance of this scheme depends critically on how good the distribution of the workers' speeds can be estimated. Observe that, in the event that a worker in a "fast" cluster straggles, the finishing time of the overall cluster can be arbitrarily delayed. This is the main drawback of this scheme in comparison with SBP coding scheme and the MM-GASP, for which any computation at any worker is equally useful. The clear advantage of the rateless coding scheme is that the computation load m_i, i.e., the number of tasks assigned to worker i, does not need to be specified in advance, and tasks can be dynamically allocated to workers in each cluster across rounds. Moreover, the recovery threshold is not dominated by the maximum computation load $m = \max m_i$, as is the case for SBP coding and the MM-GASP schemes. Therefore, in order to allow the rateless coding scheme to benefit from this flexibility, in our simulations, we consider a total upload cost for [18], i.e., the computations are assigned to clusters until the total upload communication budget is met, while for MM-GASP and SBP we impose an upload cost constraint per worker. We emphasize that this is a relaxation of the problem formulation introduced in Section II, and is only applied to the rateless coding scheme. Although SBP coding scheme and the MM-GASP code can also benefit from this relaxation when the computation statistics of the workers are known, such optimization is out of the scope of this paper and will be considered in future work.

In our simulations, following the literature [2], [19], we assume that the time for a worker to finish one sub-task is distributed as a shifted exponential random variable with density $f(t) = \lambda e^{-\lambda(t-\nu)}$ for $t \geq \nu$, and $f(t) = 0$ otherwise, where the scale parameter λ controls the speed of the worker and the shift parameter ν is the minimum time duration for a task to be completed. Smaller λ implies slower workers and more tendency to straggle. In each scenario, we run 1000

experiments independently with the given parameters and present the average computation time. We assume that the partitions of matrices A and B have the same size, i.e., $\frac{r}{K} = \frac{t}{L}$, in all of the scenarios considered. Given that the computation time per sub-task is a fraction $\frac{1}{KL}$ of the complete task, to facilitate the comparison between different configurations, we choose $\lambda \propto KL$, and $\nu \propto \frac{1}{KL}$, in all simulation setups.

A. *Heterogeneous Workers*

In this subsection, we assume that the computation speeds of the workers are heterogeneous. Specifically, we assume six *heterogeneity classes*, with scale parameters $\lambda_1 = 10^{-1} \times KL$, $\lambda_2 = 10^{-1.5} \times KL$, $\lambda_3 = 10^{-2} \times KL$, $\lambda_4 = 10^{-2.5} \times KL$, $\lambda_5 = 10^{-3} \times KL$ and $\lambda_6 = 10^{-3.5} \times KL$, and a common shift parameter of $\nu = 10/(KL)$ seconds. There are 75 workers for each class summing up to $N = 450$ workers in total, and assume that any subset of at most $T = N/15$ workers can collude. We divide both matrices A and B into $K = L = 100$ partitions. We evaluate the scheme in [18] for several numbers of clusters, c, to observe the effect of the mismatch between the actual number of heterogeneity classes and the chosen c value. While generating the clusters, we simply assign around N/c workers to each cluster, according to the estimated speeds in the first round. We do not change the parameter c across rounds.

First, we assume that workers' scale parameters do not deviate at all from the given parameters across the rounds. We call such workers as *stable workers*. In Fig. 2, we plot the ACT of the compared schemes versus the total UCB by assuming stable workers. In Fig. 3, we also present the actual recovery thresholds of SBP coding scheme, MM-GASP, as well as, the average recovery threshold of the rateless coding scheme for different c values. As the name suggests, the rateless scheme does not have a constant recovery threshold. The actual value depends on the computation speeds of the workers, the number of clusters, and the number of workers assigned to them. Therefore, we present the average recovery threshold for this scheme.

As observed in Fig. 2, the SBP coding scheme is able to finish the overall task for much lower UCB values than the other two schemes. This is thanks to the fact that SBP is able to provide more computations, m, to workers for the same UCB, as highlighted in Remark 1. Moreover, although by increasing the total UCB we increase m and therefore R_{th}, as observed in Fig. 3, and thus workers need to provide more computations to the master, the benefit from having more computations at workers pays off and the ACT decreases when UCB increases. The reason for this is the heterogeneity of the workers' speeds. That is, for a low total UCB, m is so small

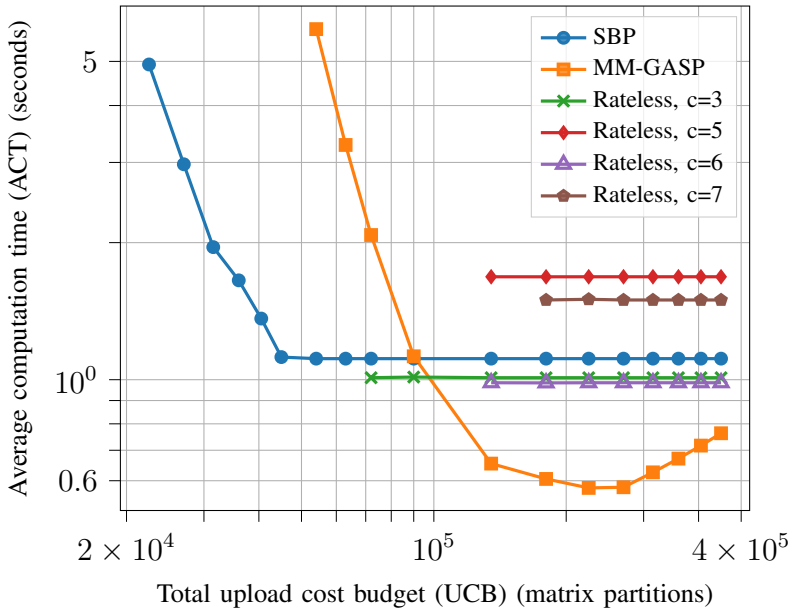

Fig. 2: ACT vs. total UCB trade-off of the compared schemes with heterogeneous and stable workers.

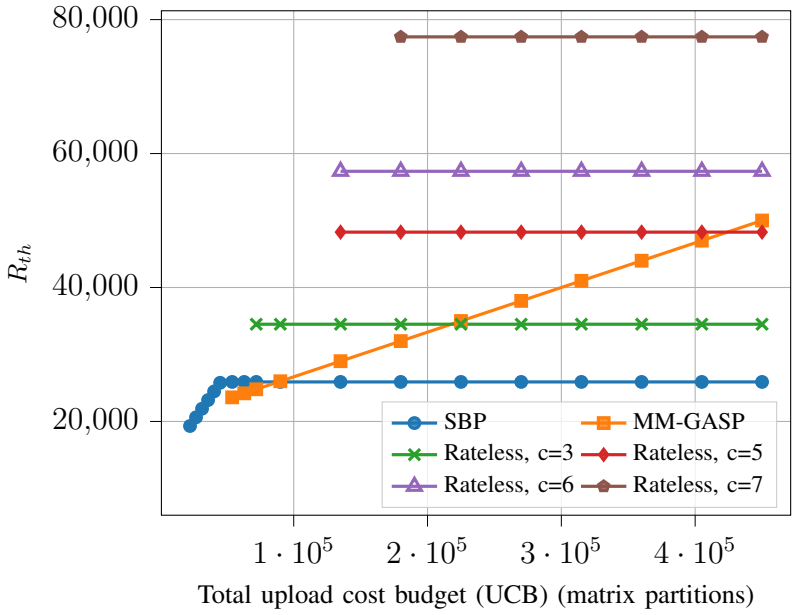

Fig. 3: Average R_{th} of the compared schemes with heterogeneous and stable workers

that the master cannot complete all R_{th} computations from only fast workers. When we increase m, the maximum number of computations the fast workers can provide also increases, and the benefit of this increase dominates over the increase in the R_{th}. For the SBP scheme, this is so until we reach a total UCB value corresponding to $m = L$ i.e., total UCB of $L \times N = 45000$.

After this point, the ACT of SBP coding scheme stays constant. This is an inherent limitation of SBP coding scheme since the maximum value of m is L. Beyond that value, we cannot benefit from the additional UCB.

For MM-GASP codes, we observe that, although their recovery threshold is close to that of SBP coding scheme in the low UCB regime, as seen in Fig. 3, the minimum total UCB for which MM-GASP codes are able to complete the overall task is larger than SBP coding scheme. That is because the MM-GASP scheme is a univariate scheme; and thus, for the same total UCB, the maximum number of computations a worker can provide is less than the one in SBP coding scheme. For the same reason, at intermediate total UCB availability, i.e., values less than 9×10^4 partitions, the ACT of the MM-GASP scheme is quite large compared to SBP coding scheme. However, for larger values of total UCB, we observe in Fig. 2 that MM-GASP's ACT decreases rapidly, substantially outperforming the other two schemes. However, after a critical point, if the total UCB further increases, then the ACT starts to increase again. After that critical point, the increase in the recovery threshold is not compensated by the additional computations at workers and the ACT starts to increase. Unfortunately, operating at this point may not be always possible. Especially when we do not have any prior information about the statistics of the workers' speeds. Nevertheless, some heuristics can still be useful to approximate it and even if the optimal point cannot be found, a sufficiently close point can still be beneficial. Thus, we can conclude that, if a good heuristic can be found to identify a near-optimal m value, for large UCB values, MM-GASP codes can complete the overall task faster than SBP coding scheme as well as the rateless coding scheme. This makes MM-GASP codes a good alternative for the scenarios with high UCB availability.

Finally, for the rateless codes, as observed for MM-GASP codes, we observe that this scheme starts being able to complete the overall task only at a relatively high total UCB value. That is because the rateless coding scheme assigns a new sub-task to a worker as soon as it finishes its task without waiting for the other clusters to finish. Thus, the UCB is greedily invested in the fastest cluster. However, despite its speed, in terms of the number of useful computations provided, the fastest cluster is less efficient than the other clusters. Please remember that $d_1 = \lfloor (n_1 - 2T + 1)/2 \rfloor$, but $d_u = \lfloor (n_u - T + 1)/2 \rfloor$ for $2 \leq u \leq c$. Therefore, if the number of workers in the fastest cluster is limited, then for the low UCB values, the rateless scheme cannot complete the overall task since it runs out of the necessary upload resources before the master receives the minimum number of useful computations to decode AB, which is $KL(1 + \epsilon)$.

Moreover, we observe in Fig. 3 that when the number of clusters is low, the recovery threshold is also lower, and the rateless scheme starts completing the overall task at a lower value of total UCB. That is because when c is low, since we assign N/c workers per cluster, there are more workers in the fastest cluster. However, in Fig. 2, we also observe that this does not always have a positive impact on the ACT. On the other hand, when UCB is large enough for rateless codes to complete the overall task, its ACT is slightly better than SBP coding scheme for $c = 3$ and $c = 6$, but for $c = 5$ and $c = 7$, SBP coding scheme performs better. In general, we expect that the rateless coding scheme performs well when c is equal to the number of heterogeneity classes, but, in this case, we also observe that it performs equally well for $c = 3$. That is because, for $c = 3$, there is no λ_i, $i \in [6]$ appearing in more than one cluster, i.e., workers in the same heterogeneity class are allocated to the same cluster. Therefore, for rateless codes, it is important to choose the design parameter c carefully. In practice, we may not know the number of heterogeneity classes, such a clear grouping of computation statistics may not be possible. For such cases, SBP coding scheme or the MM-GASP may be preferable over the rateless coding scheme.

In addition to choosing c optimally, estimating the instantaneous speeds of the workers is another issue we need to address in rateless codes. In real-world scenarios, the speeds of the workers can occasionally change due to temporary failures, parallel job assignments, etc. To model this, we introduce another simulation scenario, in which workers' scale parameters can deviate from their original values with a very low probability ρ. We refer to such workers as *mostly-stable workers*. That is, in any round, a worker with λ_i sticks to λ_i with probability $1-\rho$, but with a small probability ρ, it draws its scale parameter uniform randomly from $\{\lambda_j \mid j \in [6]\}$. We consider such a scenario to model the instantaneous changes in workers' speeds since the detection of such changes by the master and putting this worker to the correct cluster takes at least one round. Taking $\rho = 0.001$, we plot the ACT of the compared schemes in Fig. 4.

We observe that even with such a small probability deviation from the estimated scale parameters, the performance of [18] degrades considerably. Thus, we can argue that, in addition to the substantial improvement in the low and the intermediate UCB values, SBP coding scheme can be advantageous over [18] in the presence of a high UCB as well depending on the statistics of the workers' speeds.

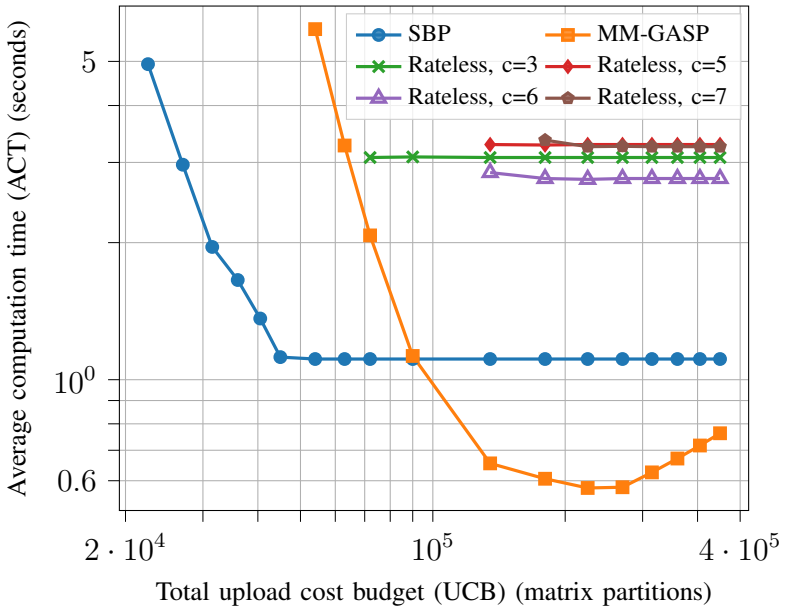

Fig. 4: ACT vs. total UCB trade-off of the compared schemes with heterogeneous and mostly-stable workers.

B. Homogeneous Workers

In this subsection, we assume that the computation speeds of the workers are homogeneous, and we compare the ACTs of the considered schemes with respect to the available total UCB. That is, we have 450 workers as in Subsection VI-A, but this time, all the workers follow the same computation statistics with $\lambda = 10^{-2} \times KL$ and $\nu = 10/(KL)$. We assume at most $T = N/15$ workers can collude, and we divide A and B into $K = L = 100$ partitions. For the rateless scheme in [18], although, the workers' speeds are homogeneous, we consider different numbers of clusters $c \in [3]$ in order to analyse its effect. In Fig. 5, we present the ACT versus UCB plot for this setting.

Similarly to the heterogeneous case discussed in Subsection VI-A, due to the upload cost efficiency of the bivariate polynomial codes, we observe that the minimum UCB for which SBP can complete the overall task is smaller than for the other schemes. Moreover, in this homogeneous case, we observe that the ACTs of SBP and MM-GASP only increase with the total UCB. That is because, due to the similarity in workers' speeds, there is no need for the faster workers to compensate for the slower ones. Therefore, rather than improving the ACT, increasing m beyond the minimum value, for which the schemes complete the overall task, results in a higher ACT since it also increases R_{th}. Therefore, we depict the best ACT for SBP

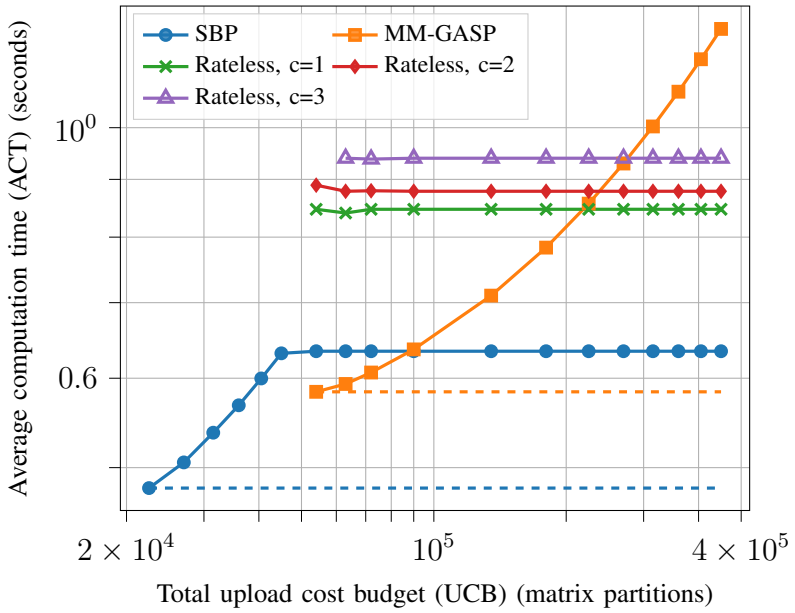

Fig. 5: ACT vs. total UCB trade-off of the compared schemes with homogeneous and stable workers.

and MM-GASP coding schemes in Fig. 5 and Fig. 6 by flat dashed lines.

On the other hand, for the rateless codes, we observe that, regardless of the number of clusters, c, considered, they perform significantly worse than SBP coding scheme for all UCB values. That is because, while the sub-tasks are one-to-any replaceable in SBP coding scheme, i.e, the result of any sub-task can compensate for the absence of any other sub-task, this is not the case in the rateless coding scheme. Since we consider the homogeneous workers in their speeds, there is not much difference between the clusters in the rateless coding scheme. Since, to decode the sub-tasks in a cluster, all of the workers in that cluster must finish their sub-tasks, the ACT increases.

As we stated in Subsection VI-A, in a real-world scenario, the speeds of workers can occasionally change. To model this effect, in Fig. 6, we provide the ACT versus UCB trade-off in the scenario in which the workers are mostly-stable with a transition probability $\rho = 0.001$. Since there is only one heterogeneity class in the homogeneous case, to simulate mostly-stable workers, we assume that a worker sticks to $\lambda = 10^{-2} \times KL$ with probability ρ, but with probability $1-\rho$, its λ parameter is chosen uniformly between $\lambda = 10^{-3} \times KL$ and $\lambda = 10^{-4} \times KL$.

We observe that the effect of such a low probable deviation from the original parameters is considerable in the rateless codes since in order to utilize the computations in a cluster, all the

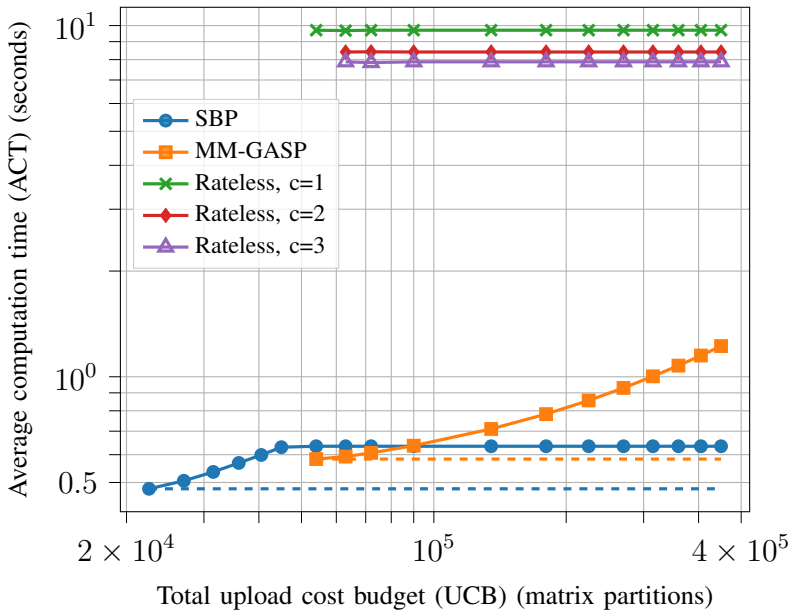

Fig. 6: ACT vs. total UCB trade-off with homogeneous and mostly-stable workers.

workers in that cluster must complete their sub-tasks. If some of these workers straggle even only for one round, it can delay the overall computation significantly.

To conclude, we observe that, in the cases in which the workers' speeds are known to be close to each other, i.e., homogeneous, SBP coding scheme is preferable over both the rateless coding and the MM-GASP schemes.

VII. Proof of Theorem 2

In Fig. 7, we visualize the degrees of the monomials of $A(x)B(x,y)$ in the $\deg(x) - \deg(y)$ plane. From Fig. 7, we see that the number of monomials of $A(x)B(x,y)$ is $(K+T)L + m(K+T-1)$. We need to show that every possible combination of so many responses from the workers interpolates to a unique polynomial, implying $(K+T)L + m(K+T-1)$ is the recovery threshold.

Definition 1. Bivariate polynomial interpolation problem can be formulated as solving a linear system of equations, whose unknowns are the coefficients of $A(x)B(x,y)$ and whose coefficient matrix consists of the monomials of $A(x)B(x,y)$ and their derivatives with respect to y evaluated at the evaluation points of the workers, $(x_i, y_i), i \in [N]$. We refer to this coefficient matrix as the **interpolation matrix** and denote it by M. Since the number of monomials of $A(x)B(x,y)$ is $R_{th} = (K+T)L + m(K+T-1)$, we require R_{th} equations to interpolate it, and hence,

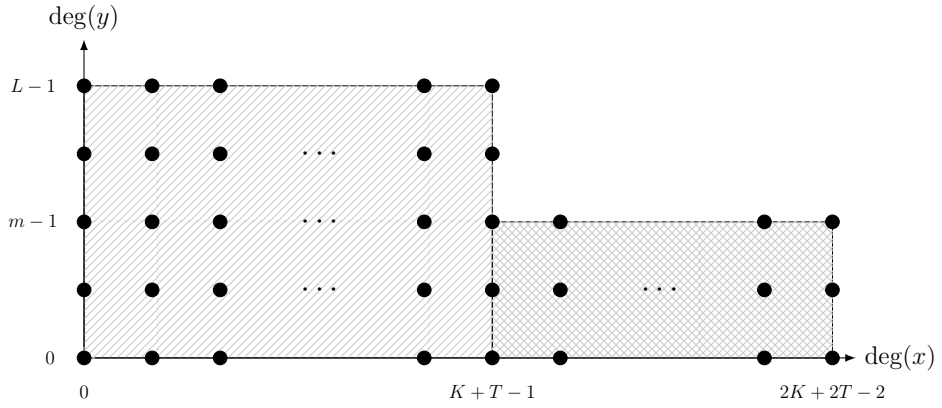

Fig. 7: The visualization of the degrees of the monomials of $A(x)B(x,y)$ in the $\deg(x)-\deg(y)$ plane.

$M \in \mathbb{R}^{R_{th}\times R_{th}}$. Each row of M corresponds to the result of one sub-task sent by a worker to the master. For example, when $K = L = 2$, $m = 2$ and $T = 1$, we have $R_{th} = 10$, and one possible interpolation matrix formed by any 5 workers, each of which provides $m = 2$ computations, is as follows:

$$M = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 & y_1 & x_1y_1 & x_1^2y_1 & x_1^3y_1 & x_1^4y_1 \\ 0 & 0 & 0 & 0 & 0 & 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_5 & x_5^2 & x_5^3 & x_5^4 & y_5 & x_5y_5 & x_5^2y_5 & x_5^3y_5 & x_5^4y_5 \\ 0 & 0 & 0 & 0 & 0 & 1 & x_5 & x_5^2 & x_5^3 & x_5^4 \end{bmatrix}.$$

Observe that the first row represents the direct evaluation $A(x_1)B(x_1,y_1)$ from worker 1, and the second row represents $A(x_1)\partial_1 B(x_1,y_1)$, again from worker 1. In general, any interpolation matrix formed by $R_{th} = 10$ computations received from any subset of workers is also valid, as long as the workers follow the computation order specified in Section IV-B.

The problem of showing that any R_{th} responses from the workers interpolates to a unique polynomial is equivalent to showing that the corresponding interpolation matrix is non-singular. The theorem claims that this is the case with high probability. First, we need to show that there exist some evaluation points for which the determinant of the interpolation matrix is not zero. That is equivalent to showing that $\det(M)$ is not the zero polynomial of the evaluation points. In [10], such a result for the same type of interpolation matrices is shown for the real field $\mathbb{R}$. Here, we extend this proof to $\mathbb{F}$. We show that $\det(M)$ is non-zero for some evaluation points

by using Taylor series expansion of $\det(M)$, as done in [10]. This can be done since Taylor series expansion is also applicable in $\mathbb{F}$, as long as, the degree of the polynomial $A(x)B(x,y)$ is smaller than the field size q. This can be guaranteed by choosing a large q. For further details on the applicability of Taylor series expansion in finite fields, see [20] and [21].

Without losing generality, let us assume first that n workers with $n \leq N$, provide, together, enough responses, i.e., R_{th}, to interpolate $A(x)B(x,y)$. Let us assume (x_i, y_i) and (x_j, y_j) are two evaluation points for which the evaluations of $A(x)B(x,y)$ and some of its derivatives at these points are received by the master. We 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:

$$\det(M) = \sum_{(\alpha_1,\alpha_2)\in\mathbb{N}^2} \frac{1}{\alpha_1!\alpha_2!}(x_j - x_i)^{\alpha_1}(y_j - y_i)^{\alpha_2} D_{\alpha_1,\alpha_2}(\tilde{Z}), \tag{26}$$

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

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

We call (x_i, y_i) the **pivot node** and (x_j, y_j) the **variable node**.

Remark 3. *The monomials $(x_j - x_i)^{\alpha_1}(y_j - y_i)^{\alpha_2}$ are linearly independent for different (α_1, α_2) pairs if there is no relation between x and y coordinates of the evaluation points, i.e., x_i and x_j do not depend on y_i and y_j. Thus, $\det(M)$ is a zero polynomial of all evaluation points, if and only if $D_{\alpha_1,\alpha_2}(\tilde{Z}) = 0, \forall(\alpha_1, \alpha_2) \in \mathbb{N}^2$. Therefore, in order to show that M is non-singular, it suffices to show that there exists at least one (α_1, α_2) making $D_{\alpha_1,\alpha_2}(\tilde{Z})$ nonzero.*

Before looking into $D_{\alpha_1,\alpha_2}(\tilde{Z})$ in more detail, let us define some notions which will help us understanding its structure.

Definition 2. Derivative Set. In an interpolation matrix M, there might be several rows each corresponding to a different derivative order of $A(x)B(x,y)$ associated with the evaluation point $z_i \triangleq (x_i, y_i)$, which is assigned to worker i. We define the derivative set of z_i, denoted by $U_{z_i,M}$ as the set of derivative orders of $A(x)B(x,y)$ with respect to x and y associated to z_i in M. That is, $(d_x, d_y) \in U_{z_i,M}$ if and only if M has a row corresponding to $\frac{\partial^{d_x+d_y}}{\partial x_i^{d_x} \partial y_i^{d_y}} A(x_i)B(x_i, y_i)$.

Definition 3. Derivative order space. The derivative order space of a bivariate polynomial $A(x)B(x,y)$ is defined as the 2-dimensional space of all its possible derivative orders. Since the

largest derivative order of a bivariate polynomial is its largest monomial degree, the derivative order space has the same shape as $\deg(x) - \deg(y)$ plane depicted in Fig. 7. For example, consider the 2-D derivative order $(K+T, m)$. Since all the monomials of $A(x)B(x,y)$ having a degree larger than m with respect to y have a degree less than or equal to $K+T-1$ with respect to x, the 2-D derivative order $(K+T, m)$ results in a zero polynomial. Thus, this is not an element of derivative order space. The derivative set of each evaluation point can be depicted in the derivative order space separately.

Definition 4. Let M be an interpolation matrix for which some of its rows depend on x_j and y_j. Let us denote by r_i its i^{th} row and define a *simple shift* as

$$\partial_{i,x_j} M \triangleq \left[r_1^T, \ldots, \frac{\partial}{\partial x_j} r_i^T, \ldots, r_{KL}^T \right]^T \tag{27}$$

and

$$\partial_{i,y_j} M \triangleq \left[r_1^T, \ldots, \frac{\partial}{\partial y_j} r_i^T, \ldots, r_{KL}^T \right]^T. \tag{28}$$

That is, ∂_{i,x_j} and ∂_{i,y_j} transform M into another matrices by taking the derivative of its i^{th} row with respect to x_j and y_j, respectively. If in the resulting derivative set, i.e., $U_{x_j,\partial_{i,x_j}M}$ or $U_{y_j,\partial_{i,y_j}M}$, each element has a multiplicity of one, then the shift is called a *regular simple shift*.

Definition 5. Let $\boldsymbol{k}$ and $\boldsymbol{l}$ be vectors such that $\boldsymbol{k} \in \{0,1,\cdots,K-1\}^{R_{th}}$ and $\boldsymbol{l} \in \{0,1,\cdots,L-1\}^{R_{th}}$. We define *the composition of simple shifts* as

$$\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M = \partial_{1,x_j}^{\boldsymbol{k}(1)} \partial_{2,x_j}^{\boldsymbol{k}(2)} \cdots \partial_{R_{th},x_j}^{\boldsymbol{k}(R_{th})} \partial_{1,y_j}^{\boldsymbol{l}(1)} \partial_{2,y_j}^{\boldsymbol{l}(2)} \cdots \partial_{R_{th},y_j}^{\boldsymbol{l}(R_{th})} M. \tag{29}$$

That is, the i^{th} element of $\boldsymbol{k}$ denotes the order of the derivative of i^{th} row of M with respect to the variable x_j, and the i^{th} element of $\boldsymbol{l}$ denotes the order of the derivative of i^{th} row of M with respect to the variable y_j. In fact, (29) is not the only way to compute $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M$ since the derivative operation is commutative. One can compute $\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M$ in any other order. Each of these possible orders are referred to as a derivative path. If a derivative path involves only regular simple shifts, i.e. after each derivative there are not two equal rows, then it is called a *regular derivative path*. We denote the number of regular derivative paths by $C_{\boldsymbol{k},\boldsymbol{l}}(M)$.

Based on these definitions, we have the following lemma.

Lemma 1 (Lemma 1 in [10]). *Let* $\boldsymbol{k} \in \{0,1,\cdots,K-1\}^{R_{th}}$, $\boldsymbol{l} \in \{0,1,\cdots,L-1\}^{R_{th}}$ *and* $\alpha_1 = \sum_{i=1}^{R_{th}} \boldsymbol{k}(i)$ *and* $\alpha_2 = \sum_{i=1}^{R_{th}} \boldsymbol{l}(i)$. *Then, we have*

$$\frac{\partial^{\alpha_1+\alpha_2}}{\partial x_j^{\alpha_1} \partial y_j^{\alpha_2}} \det(M)\Big|_{x_j=x_i, y_j=y_i} = \sum_{(\boldsymbol{k},\boldsymbol{l}) \in \mathcal{R}_M(\alpha_1,\alpha_2)} C_{\boldsymbol{k},\boldsymbol{l}}(M) \det\left(\nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M\right)\Big|_{x_j=x_i, y_j=y_i} \tag{30}$$

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.*

Definition 6. If $\mathcal{R}_M(\alpha_1,\alpha_2)$ has only one element, i.e., there is only one $(\boldsymbol{k},\boldsymbol{l})$ resulting in a regular simple shift, then (α_1,α_2) is called a *unique shift order* and $(\boldsymbol{k},\boldsymbol{l})$ is called a *unique shift*.

Now, let us go back to $D_{\alpha_1,\alpha_2}(\tilde{Z})$. Recall that we would like to show that at least for one (α_1,α_2), $D_{\alpha_1,\alpha_2}(\tilde{Z})$ is non-zero. Observe that it does not depend on (x_j,y_j) since after taking the derivatives with respect to (x_j,y_j), the resulting expression is evaluated at $x_j = x_i, y_j = y_i$. If (α_1,α_2) is a unique shift order, then according to (30), it is enough to show that $\det(M_1)$ is not the zero polynomial, where $M_1 \triangleq \nabla_{\boldsymbol{k},\boldsymbol{l}}^{x_j,y_j} M \mid_{x_j=x_i,y_j=y_i}$. Notice that, M_1 no longer depends on the evaluation point (x_j,y_j). We call such a procedure of transforming an interpolation matrix into another interpolation matrix via unique shifts as the *coalescence* of the variable node and the pivot node. After obtaining M_1, we can employ the same idea to show M_1 is non-singular. Namely, we can write the Taylor series expansion of $\det(M_1)$ by choosing a new variable node and keeping the same pivot node (x_i,y_i). If there is a unique shift for the coalescence, the resultant matrix M_2 will not depend on neither the previous variable node (x_j,y_j) nor the current variable node. We can apply such coalescences successively as long as we can find a unique shift order (α_1,α_2) at each coalescence, until M_{final} depends only on one evaluation point, which is the pivot node, (x_i,y_i). In M_{final}, the derivative set of (x_i,y_i) has all possible elements of the derivative order space. Thus, M_{final} is a triangular matrix, and hence, non-singular.

To summarize, to prove that all possible interpolation matrices, M, generated from our scheme are non-singular in general, we need to show that we can always find at least one unique shift for all the steps of the coalescence procedure. Our strategy to show that we can always find a unique shift in all coalescence steps is based on the idea of keeping the derivative set of the pivot node to be a lower set. A lower set is defined as a set in which the presence of an element implies the presence of all possible elements smaller than this element. To decide if an element is smaller than any other element, we need to define an ordering rule. For our case,

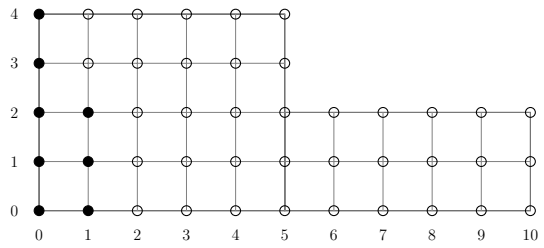

Fig. 8: Depiction of $U_{z_i,M_{p-1}}$ in Example 1

we define such an ordering as follows. Assume we denote our pivot node as $z_i = (x_i, y_i)$ and take two derivative orders $(a, b) \in U_{z_i,M}$ and $(c, d) \in U_{z_i,M}$, where a and c are the orders of the derivative with respect to x_i and b and d are the orders of the derivative with respect to y_i. We say $(a, b) < (c, d)$ if and only if $a < c$ or $a = c$ and $b < d$.

Before formally stating our strategy to find a unique shift in all the coalescence steps, we describe it with two simple examples.

Example 1. Assume $K = L = 5$, $T = 1$ and $m = 3$ and we are at the beginning of p-th coalescence step. Let us choose z_i as the pivot node and z_j as the variable node. Further, assume at the beginning we have $U_{z_i,M_{p-1}} = \{(a, b) : (a, b) \leq (1, 2)\}$ and $U_{z_j,M_{p-1}} = \{(a, b) : (a, b) \leq (0, 2)\}$. We depict the derivative sets of $U_{z_i,M_{p-1}}$ and $U_{z_j,M_{p-1}}$ in Fig. 8 and Fig. 9. We will take smallest possible shift (α_1, α_2) such that the resultant U_{z_i,M_p} after the coalescence. Knowing the number of elements in U_{z_i,M_p} after the coalescence, its shape is uniquely determined under the condition that it must be a lower set and shown in Fig. 10. In Fig. 9 and Fig. 10, we assign to each element of $U_{z_j,M_{p-1}}$ either the letter "a", "b" or "c" so that we can track its location during and after the coalescence procedure. Recall that taking derivatives corresponds to shifting the elements of the derivative set in the derivative order space. Thus, in order to shift the elements of $U_{z_j,M_{p-1}}$ to their locations in the final shape in Fig. 10, we need to the total number of shifts in both x and y directions is 4, implying we need to choose $(\alpha_1, \alpha_2) = (4, 4)$. For this choice, we have $\boldsymbol{k}(i_a) = 2$, $\boldsymbol{k}(i_b) = 1$, $\boldsymbol{k}(i_c) = 1$, $\boldsymbol{l}(i_a) = 0$, $\boldsymbol{l}(i_b) = 2$ and $\boldsymbol{l}(i_c) = 2$ where i_a is the row-index of the element a. Given this choice of (α_1, α_2), there is no other possible resulting shape for U_{z_i,M_p} resulting a non-singular M_p. To see this, observe that, if we write the derivative sets of U_{z_i,M_p} after-the-coalescence for all possible $(\boldsymbol{k}, \boldsymbol{l})$ such that $\sum_i \boldsymbol{k}(i) = \alpha_1$ and $\sum_i \boldsymbol{l}(i) = \alpha_2$, then all, except the one depicted in Fig. 10 will have overlapping elements making the corresponding interpolation matrix singular. Therefore, $(\alpha_1, \alpha_2) = (4, 4)$ is a unique shift order.

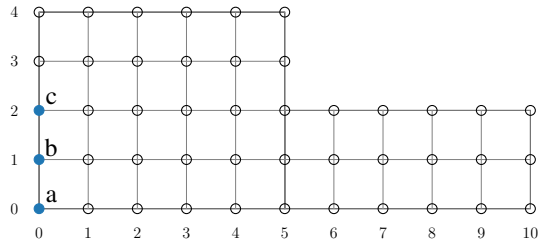

Fig. 9: Depiction of $U_{z_j,M_{p-1}}$ in Example 1

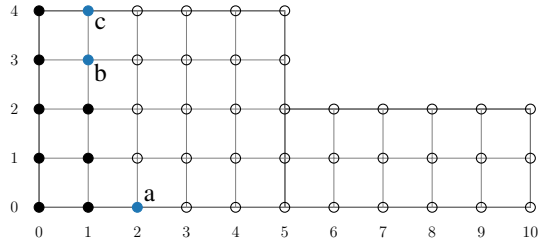

Fig. 10: Depiction of U_{z_i,M_p} after coalescence in Example 1

Example 2. Let us consider the same setting as in Example 1, but now assume that we have $U_{z_i,M_{p-1}} = \{(a,b) : (a,b) \leq (6,1)\}$. Since the maximum number of computations a worker can provide is $m = 3$, the cardinality of the derivative set of the variable node $U_{z_j,M_{p-1}}$, in this example, is at its maximum. Thus, we can directly follow the same procedure as in Example 1. Note that after the coalescence, $U_{z_iM_p}$ will have 34 elements, and the lower set having 34 elements is unique and well defined. To obtain the shape in Fig. 11, we need $(\alpha_1, \alpha_2) = (0, 19)$ with $\boldsymbol{k}(i_a) = 7$, $\boldsymbol{k}(i_b) = 6$ and $\boldsymbol{k}(i_c) = 6$, and it is a unique shift order since any other assignment of 19 shifts to a, b and c results in a non-singular M_p.

Next, we formally state our strategy for an arbitrary coalescence step p. Since we choose one pivot node and use it for every coalescence step, we guarantee that the variable node's derivative set has always at most m elements. To generalize the procedure in Example 1 and

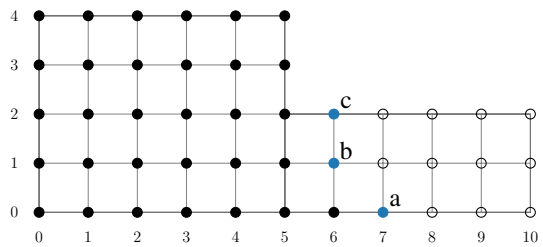

Fig. 11: Depiction of U_{z_i,M_p} after coalescence in Example 2

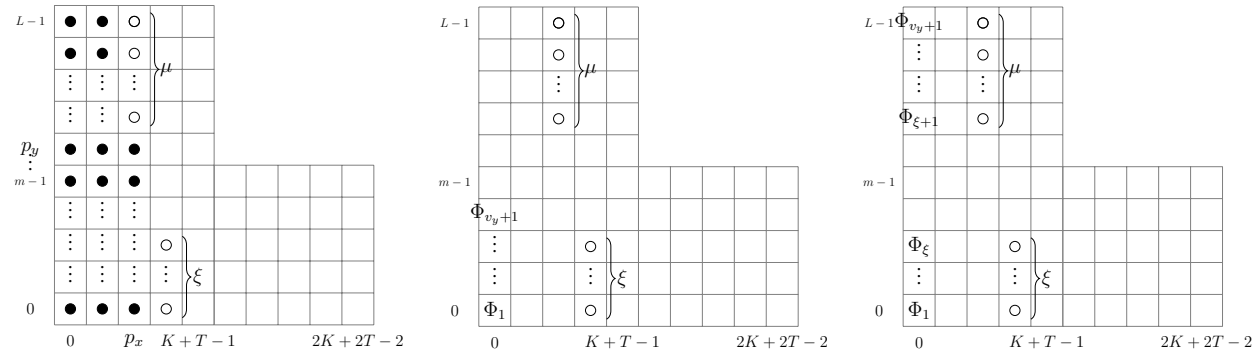

Fig. 12: Visualization of the pivot node and the variable node during a coalescence

Example 2, let us assume (p_x, p_y) is the largest element of the derivative set of the pivot node z_i, i.e., $U_{z_i,M_{p-1}} = \{(a,b) : (0,0) \leq (a,b) \leq (p_x,p_y)\}$ and $(0,v_y)$ is the largest element of the derivative set of the variable node z_j, i.e., $U_{z_j,M_{p-1}} = \{(0,b) : 0 \leq b \leq v_y\}$. While calculating (α_1,α_2) pair, we first determine α_2, which is the total derivative order with respect to y_j, or equivalently the number of shifts towards y-direction in the derivative order set. This means that we first take the derivatives with respect to y_i, and then with respect to x_i. In Fig. 12a, in the derivative order space, for $p_x \leq K+T-1$, we depict the derivative set of the pivot node, i.e., $U_{z_i,M_{p-1}}$, by filled circles, and the locations to which the elements of $U_{z_j,M_{p-1}}$ will be placed after the coalescence by unfilled circles. Note that we determine these locations by inserting the elements of $U_{z_j,M_{p-1}}$ into $U_{z_i,M_{p-1}}$ such that the derivative set of the pivot after the coalescence, i.e., U_{z_i,M_p}, satisfies the lower set property. In Fig. 12b, instead of the elements of $U_{z_i,M_{p-1}}$, we depict the elements of $U_{z_j,M_{p-1}}$ together with the locations they will be placed after the coalescence to facilitate visualization of the necessary shifts. To be able to keep track of the elements, we depict each one of them by Φ_i, $i \in [v_y+1]$. We denote the number of elements in $U_{z_j,M_{p-1}}$ to be shifted towards y-direction, by μ. We further define $\xi \triangleq v_y+1-\mu$. As shown in Fig. 12a, when the number of empty spaces in the rightmost partially occupied column of $U_{z_i,M_{p-1}}$ is smaller than $|U_{z_j,M_{p-1}}| = v_y+1$, μ becomes this number, i.e., $\mu = L-p_y-1$, since these spaces must be filled. Otherwise, to fill as many as spaced possible, all elements of the derivative set of the pivot node are shifted towards y-direction and μ becomes v_y+1. Thus, $\mu = \min\{L-p_y-1, v_y+1\}$ if $p_x \leq K+T-1$. When $p_x > K+T-1$, in fact, the same logic also applies but the maximum number of elements that can be placed in a column in Fig. 12a would be m instead of L. Thus, the expression for μ is modified as $\mu = \min\{m-p_y-1, v_y+1\}$,

which is obtained by replacing L with m.

Next, please remember that only regular simple shifts are considered for unique shifts. Thus, while taking y-directional derivatives, i.e., shifts towards y-direction in Fig. 12b, the sequence of the elements in the y-axis does not change. For instance, Φ_{v_y+1} stays always on top of the elements denoted by $\Phi_i, i \in [v_y]$. If, for example, as a result of some shifts, Φ_{v_y} is placed on top of Φ_{v_y+1}, then this would be possible only if the element Φ_{v_y} is located in the same location as Φ_{v_y+1} at some point, and this would contradict the assumption of regular simple shifts. Therefore, thanks to this property, there is only one resulting order after shifting the uppermost μ elements towards y-direction. We show the elements of the variable node's derivative set after y-directional shifts in Fig. 12c. All the remaining shifts, now, are the ones towards x-direction so that the elements of $U_{z_j,M_{p-1}}$ are located to their intended locations, i.e., unfilled circles in Fig. 12c. Notice that each Φ_i is already aligned with its final location in y-direction, and hence, each one of them will be shifted towards x-direction by a sufficient amount. Therefore, these shifts also result in a unique shape. From these observations, we can conclude that whenever the derivative sets of the pivot node and the variable node are lower sets, there exists a unique shift for their coalescence.

From this discussion, we can conclude that $\det(M)$ is not zero polynomial for large enough q. Next, we need to find the upper bound on the probability $\det(M) = 0$, when the evaluation points are sampled uniform randomly from $\mathbb{F}$.

Lemma 2. [22, Lemma 1] *Assume P is a non-zero, v-variate polynomial of variables $\alpha_i, i \in [v]$. Let d_1 be the degree of α_1 in $P(\alpha_1, \ldots, \alpha_v)$, and $P_2(\alpha_2, \ldots, \alpha_v)$ be the coefficient of $\alpha_1^{d_1}$ in $P(\alpha_1, \ldots, \alpha_v)$. Inductively, let d_j be the degree of α_j in $P_j(\alpha_j, \ldots, \alpha_v)$ and $P_{j+1}(\alpha_{j+1}, \ldots, \alpha_v)$ be the coefficient of α_j in $P_j(\alpha_j, \ldots, \alpha_v)$. Let S_j be a set of elements from a field $\mathbb{F}$, from which the coefficients of P are chosen. Then, in the Cartesian product set $S_1 \times S_2 \times \cdots \times S_v$, $P(\alpha_1, \ldots, \alpha_v)$ has at most $|S_1 \times S_2 \times \cdots \times S_v| \left(\frac{d_1}{|S_1|} + \frac{d_2}{|S_2|} + \cdots + \frac{d_v}{|S_v|} \right)$ zeros.*

In our case, since the elements of M are the monomials of $A(x)B(x,y)$ and their derivatives with respect to y, evaluated at some (x_i, y_i), $\det(M)$ is a multivariate polynomial of the evaluation points (x_i, y_i). Thus, v is the number of different evaluation points in M. We choose the evaluation points from the whole field $\mathbb{F}$. Thus, $S_j = \mathbb{F}$ and $|S_j| = q, \forall j \in [v]$, and $|S_1 \times S_2 \times \cdots \times S_v| = q^v$. Then, the number of zeros of $\det(M)$ is at most $q^{v-1}(d_1 + d_2 + \cdots + d_v)$. If we sample the evaluation points uniform randomly, then the probability that

$\det(M) = 0$ is $(d_1 + d_2 + \cdots + d_v)/q$, since we sample a v-tuple of evaluation points from $S_1 \times S_2 \times \cdots \times S_v$. To find $d_1 + d_2 + \cdots + d_v$, we resort to the definition of determinant, that is $\det(M) = \sum_{i=1}^{R_{th}} (-1)^{1+i} m_{1,i} M_{1,i}$, where $m_{1,i}$ is the element of M at row 1 and column i and $M_{1,i}$ is the minor of M when row 1 and column i are removed [23, Corollary 7.22]. Thus, to identify the coefficients in Lemma 2, in the first row of M, we start with the monomial with the largest degree. Assuming the monomials are placed in an increasing order of their degrees, the largest degree monomial is at column R_{th}. If that monomial is univariate, then d_1 is the degree of the monomial and the coefficient of $\alpha_1^{d_1}$ is $P_2(x_2, \ldots, x_v) = \det(M_{1,1})$. If the monomial is bivariate, then we take the degree of the corresponding evaluation of x, i.e., α_1, as d_1, and the degree of the corresponding evaluation of y, i.e., α_2, as d_2. In this case, the coefficient of α^{d_2} is $P_3(\alpha_3, \ldots, \alpha_v) = \det(M_{1,1})$. Next, we take $M_{1,1}$, and repeat the same procedure. We do so until we reach a monomial of degree zero. In this procedure since we visit all the monomials of $A(x)B(x,y)$ evaluated at different evaluation points, i.e., α_i's, the sum $d_1 + d_2 + \cdots + d_v$ becomes the sum of degrees of all the monomials of $A(x)B(x,y)$. The next lemma helps us in computing this.

Lemma 3. *Consider the polynomial $P(x,y) = \sum_{i=0}^{a} \sum_{j=0}^{b} c_{ij} x^i y^j$, where $c_{i,j}$'s are scalars. The sum of the degrees of all the monomials of $P(x,y)$ is given by $\xi(a,b) \triangleq \frac{a(a+1)}{2}(b+1) + \frac{b(b+1)}{2}(a+1)$.*

Proof: The sum of the degrees of all the monomials are given by

$$\sum_{i=0}^{a}\sum_{j=0}^{b}(i+j) = \sum_{i=0}^{a} i(b+1) + \sum_{i=0}^{a}\sum_{j=0}^{b} j = \frac{a(a+1)}{2}(b+1) + \frac{b(b+1)}{2}(a+1).$$

■

By using Lemma 3, the sum of monomial degrees in the diagonally shaded rectangle in Fig. 7 is

$$\begin{aligned}\xi(K+T-1, L-1) =& \frac{(K+T-1)(K+T)}{2} L + \frac{(L-1)L}{2}(K+T) \\ =& \frac{(K+T)L}{2}(K+L+T-2).\end{aligned}$$

The sum of monomial degrees in the rectangle shaded by crosshatches in Fig. 7 is given by

$$
\begin{aligned}
&\xi(2K+2T-2,m-1)-\xi(K+T-1,m-1)\\
=&\frac{(2K+2T-2)(2K+2T-1)}{2}m+\frac{(m-1)m}{2}(2K+2T-1)\\
&-\frac{(K+T-1)(K+T)}{2}m-\frac{(m-1)m}{2}(K+T)\\
=&\frac{m}{2}\left(3(K+T)^2+m(K+T)-8K-6T-m+3\right).
\end{aligned}
$$

By summing them we obtain $d_1+d_2+\cdots+d_v=\frac{m}{2}\left(3(K+T)^2+m(K+T)-8K-6T-m+3\right)+\frac{(K+T)L}{2}(K+L+T-2)$, which concludes the proof.

VIII. Conclusion

In this work, we have proposed storage- and upload-cost-efficient bivariate Hermitian polynomial codes named as SBP coding for straggler exploitation for SDMM. Although the previous works usually assume the availability of at least as many workers as the recovery threshold, the multi-message approach allows the completion of the task even if the number of workers is less than the recovery threshold. Compared to univariate polynomial coding based approaches including MM-GASP codes, SBP coding scheme has a lower upload cost and less storage requirement, making the assignment of several sub-tasks to each worker more resource efficient. Thanks to these properties, SBP codes improve the average computation time for SDMM, especially when the number of workers, the upload cost budget, or the storage capacity is limited.

REFERENCES

[1] B. Hasircioglu, J. Gómez-Vilardebó, and D. Gunduz, "Speeding up private distributed matrix multiplication via bivariate polynomial codes," *2021 IEEE International Symposium on Information Theory (ISIT)*, pp. 1853–1858, 2021.

[2] 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.

[3] 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.

[4] 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.

[5] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Transactions on Information Theory*, vol. 66, pp. 1920–1933, 2020.

[6] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch computation," *IEEE Transactions on Information Theory*, vol. 67, pp. 2821–2846, 2021.

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

[8] 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.

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

[10] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate polynomial coding for efficient distributed matrix multiplication," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 814–829, 2021.

[11] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[12] J. Kakar, S. Ebadifar, and A. Sezgin, "Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation," *arXiv preprint arXiv:1810.13006*, 2018.

[13] R. G. L. D'Oliveira, S. El Rouayheb, and D. Karpuk, "GASP codes for secure distributed matrix multiplication," *IEEE Transactions on Information Theory*, vol. 66, no. 7, pp. 4038–4050, 2020.

[14] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2722–2734, 2020.

[15] Z. Jia and S. A. Jafar, "On the capacity of secure distributed batch matrix multiplication," 2019.

[16] J. Kakar, A. Khristoforov, S. Ebadifar, and A. Sezgin, "Uplink cost adjustable schemes in secure distributed matrix multiplication," *2020 IEEE International Symposium on Information Theory (ISIT)*, pp. 1124–1129, 2020.

[17] N. Mital, C. Ling, and D. Gunduz, "Secure distributed matrix computation with discrete Fourier transform," *arXiv preprint arXiv:2007.03972*, 2020.

[18] R. Bitar, M. Xhemrishi, and A. Wachter-Zeh, "Rateless codes for private distributed matrix-matrix multiplication," *arXiv preprint arXiv:2004.12925*, 2020.

[19] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 826–834.

[20] K. Hoffman and R. Kunze, "Linear algebra," *Englewood Cliffs, New Jersey*, 1971.

[21] F. Fontein, "The Hasse derivative," Aug 2009. [Online]. Available: https://math.fontein.de/2009/08/12/the-hasse-derivative/

[22] J. T. Schwartz, “Fast probabilistic algorithms for verification of polynomial identities,” *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.

[23] J. Liesen and V. Mehrmann, *Linear algebra*, ser. Springer Undergraduate Mathematics Series.