Straggler-free Coding for Concurrent Matrix Multiplications

Pedro Soto
School of Computing and Information Sciences
Florida International University
Miami, FL
Email: *psoto004@fiu.edu*

Jun Li
School of Computing and Information Sciences
Florida International University
Miami, FL
Email: *junli@cs.fiu.edu*

***Abstract*—Matrix multiplication is a fundamental building block in various distributed computing algorithms. In order to compute the multiplication of large matrices, it is common practice to distribute the computation into multiple tasks running on different nodes. In order to tolerate potential stragglers among such nodes, various coding schemes have been proposed which add additional coded tasks. However, most existing coding schemes for the matrix multiplication are constructed for only one matrix multiplication, while it is common to compute multiple matrix multiplications concurrently in large-scale distributed computing workloads. In this paper, we propose a novel coding framework where the results of multiple multiplications can be obtained within one job concurrently. Compared with running the multiplications separately with multiple jobs, our work demonstrates that the same number of stragglers can be tolerated with much fewer tasks.**

***A full version of this paper is accessible at:* https://users.cs.fiu.edu/~junli/papers/pedro-isit20-full.pdf**

I. Introduction

Recent advances in large-scale distributed computing have demonstrated its success in various applications, such as machine learning and data analytics. With the massive size of modern datasets, large-scale computing jobs become inevitable to take advantage of the parallelism by running multiple tasks in parallel on a large number of nodes. However, it is well known that nodes in a distributed infrastructure are typically built with commodity hardware and are subject to various faulty behaviors [1]. For example, nodes may experience temporary performance degradation, due to load imbalance or resource congestion [2]. It is measured in an Amazon EC2 cluster that a virtual machine affected can have a performance reduction by up to 5 times [2], [3]. A node may even fail to complete a task due to hardware failures, network partition, or power failures. In a Facebook data center, it has been reported that up to more than 100 such failures can happen on a daily basis [4], [5]. Therefore, when the computation is distributed onto multiple servers, its progress can be significantly affected by the tasks running on such slow or failed nodes, which we call *stragglers*.

The adversarial effects of stragglers can be mitigated by launching redundant tasks in advance. A native application in this principle is to replicate each task on multiple nodes. For example, if we run each task on three nodes, the results of any two tasks affected by stragglers can be simply disregarded while all the other tasks can continue without being delayed. This naive method, however, will further significantly increase the resource consumption including computing, communication, and storage, with only a limited number of stragglers tolerable. Specifically, in order to tolerate any r stragglers, we have to replicate each task on $r+1$ nodes.

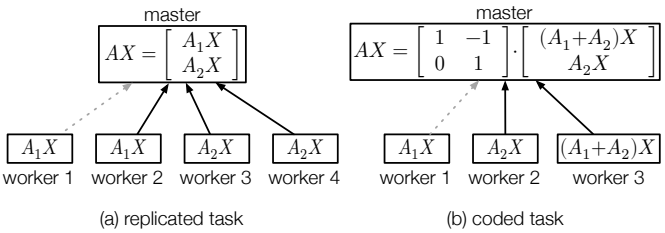

Fig. 1. Examples of distributed matrix multiplication with additional workers (running replicated or coded tasks) to tolerate one single straggler, represented with a gray dotted arrow.

On the other hand, it has been demonstrated that we can tolerate the same number of stragglers with much fewer tasks if we run *coded* tasks as redundant tasks. Fig. 1 illustrates an example of the distributed matrix multiplication with replicated and coded tasks. We calculate AX on four worker nodes in Fig. 1a. The matrix A is split into two submatrices, A_1 and A_2, and then AX can be obtained from the results of two tasks, *i.e.*, A_1X and A_2X. In Fig. 1a, such two tasks are replicated on two workers, respectively. Therefore, any single straggler among the total four workers can be tolerated without affecting the overall performance. In Fig. 1b, however, a third worker executes a coded task $(A_1 + A_2)X$, which can be decoded to recover A_1X or A_2X if any other task runs on a straggler. Therefore, compared to replicating the two tasks in Fig. 1a, coded matrix multiplication in Fig. 1b can save the number of additional workers by 50% and tolerate the same number of stragglers.

Although significant research attention has been attracted to coded computing, especially for the coded matrix multiplication, existing coding schemes have been focusing on constructing coding for one single matrix multiplication so far. In this paper, we consider a more general scenario where multiple matrix multiplications need to be computed at the same time. Conventionally, we can launch multiple distributed

jobs for each multiplication, and additional tasks need to be added into each job to tolerate its own stragglers, *i.e.*, a coded task can only tolerate a straggler inside the same job.

In this paper, we propose a novel coding framework for distributed matrix multiplications where the results of multiple matrix multiplications can be obtained concurrently inside one job. In this coding framework, the additional coded tasks can be used for tolerating stragglers affecting multiple matrix multiplications.

II. Motivating Examples

We start with a toy example to demonstrate the advantages of our coding framework. In this toy example, we do not apply coding in each matrix multiplication but across different ones. In this section, we will discuss this simple case with two multiplications only and present the general construction in Sec. III.

Assume that there are two matrix multiplications, *i.e.*, A_1B_1 and A_2B_2. We assume that A_1 and A_2 are of the same size, and also B_1 and B_2. If the two multiplications are computed as two tasks, we can replicate each task on $r+1$ nodes, such that any r stragglers can be tolerated. In other words, we need to have $2(r+1)$ tasks to tolerate any r stragglers and complete the two matrix multiplications, since the replicated tasks for one job cannot be used in the other job.

A possible way to add coded tasks for the two jobs is to embed the two matrix multiplications into one larger job as

$$\hat{A} \cdot \hat{B} \triangleq \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_2 \end{bmatrix} = \begin{bmatrix} A_1B_1 & A_1B_2 \\ A_1B_2 & A_2B_2 \end{bmatrix}. \quad (1)$$

In this way, the result of A_1B_1 and A_2B_2 can be obtained as submatrices of $\hat{A}\hat{B}$. However, the complexity of $\hat{A}\hat{B}$ becomes four times as A_iB_i. In order to generate coded tasks with the same complexity as A_iB_i, we can apply polynomial codes [6], a polynomial-based coding scheme for matrix multiplication, to the job of $\hat{A}\hat{B}$, by encoding $\hat{A}$ as $\tilde{A}(x) = A_1x^0 + A_2x^2$ and $\hat{B}$ as $\tilde{B}(x) = B_1x^0 + B_2x^1$, and then the sizes of $\tilde{A}(x)$ and $\tilde{B}(x)$ equal those of A_i and B_i, respectively. A coded task can then be generated as a polynomial of $\tilde{C}(x) \triangleq \tilde{A}(x)\tilde{B}(x)$, *i.e.*,

$$\tilde{C}(x) = A_1B_1x^0 + A_1B_2x^1 + A_2B_1x^2 + A_2B_2x^3. \quad (2)$$

Given any 4 coded tasks $\tilde{C}(x)$ with different values of x, the coefficients of this polynomial can be solved with interpolation or Reed-Solomon decoding. In other words, we can tolerate r stragglers with a total of $4+r$ tasks.

On the other hand, we propose a coding framework in this paper that requires significantly fewer tasks than replication and polynomial codes, tolerate the same number of stragglers with the same complexity in the coded tasks. Given the two matrix multiplications above, $\tilde{A}(x)$ and $\tilde{B}(x)$ can be generated differently where $\tilde{A}(x) = A_1x^0 + A_2x^1$, and $\tilde{B}(x) = B_1x^0 + B_2x^1$. Hence, $\tilde{C}(x) = A_1B_1x^0 + (A_1B_2 + A_2B_1)x^1 + A_2B_2x^2$. In this way, we only need the results of any three coded tasks (with different values of x), and the total number of tasks becomes $3+r$.

Although when $n = 2$ the number of tasks can only be saved by no more than 25%, the saving can be more significant with a general value of n. Compared to other works that construct polynomial-based coding for concurrent matrix multiplications [7]–[10], our coding scheme achieves the lowest possible complexity among all polynomial-based coding schemes, as the input matrices directly become the coefficients in the polynomial. On the other hand, the number of tasks required is more than other coding schemes. In Sec. III, we will present the general coding framework for n matrix multiplications.

III. General Coding Framework

Given n matrix multiplications, *i.e.*, A_1B_1, A_2B_2, ..., and A_nB_n, we assume that $A_1, \ldots, A_n$ are of the same size, and $B_1, \ldots, B_n$ also have the same sizes. In this paper, we propose a polynomial-based coding scheme which optimizes the number of tasks required to tolerate any r stragglers with the lowest encoding complexity. In particular, we focus on the case where the input matrices are not split, *i.e.*, the coding is only applied across matrix multiplications, instead of in each matrix multiplication.[1] In other words, the complexity of each coded task remains the same as that of the original matrix multiplications.

In our coding framework, the parameters of the coding scheme can be described by four vectors with n elements, M, N, P, and Q. In particular, M and N are permutations of $\{1, \ldots, n\}$. The values in P and Q can be arbitrary integers, which will be used as the exponents in the polynomial. The n matrix multiplications can be encoded into coded tasks which multiply $\tilde{A}(x)$ and $\tilde{B}(x)$, where $\tilde{A}(x) = \sum_{i=1}^{n} A_{M_i}x^{P_i}$ and $\tilde{B}(x) = \sum_{i=1}^{n} B_{N_i}x^{Q_i}$. Therefore, a coded task can be generated as $\tilde{C}(x) = \tilde{A}(x)\tilde{B}(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{M_i}B_{N_j}x^{P_i+Q_j}$, which is a polynomial of x. With an appropriate choices of M, N, P, and Q, we can find A_iB_i from the coefficients of $\tilde{C}(x)$. In other words, their corresponding exponents of x should be unique.

To illustrate the code scheme, in this paper, we use a $n \times n$ table to depict a particular choices of M, N, P, Q, as shown in Fig. 2a. In this table, the entry in the i-th row and the j-th column is filled with $P_i + Q_j$, the exponent of $A_{M_i}B_{N_j}$. We also place A_{M_i} and B_{M_i}, $i = 1, \ldots, n$, as the head of each row and each column, respectively. We demonstrate three examples of feasible coding schemes under our coding framework in Fig. 2b-Fig. 2d.

As shown in Fig. 2b, a naive choice of parameters in the coding framework is to let $M = N = (1, \ldots, n)$, $P = (0, \ldots, n-1)$, and $Q = (0, n, \ldots, (n-1)n)$, corresponding to the example of polynomial codes in Sec. II. In this way, we have $\tilde{A}(x) = \sum_{i=1}^{n} A_ix^{i-1}$ and $\tilde{B}(x) = \sum_{i=1}^{n} B_ix^{(i-1)n}$. Therefore, in $\tilde{C}(x)$, there are n^2 terms whose coefficients are A_iB_j, $1 \le i, j \le n$. As shown in Fig. 2b, the exponents of the four terms in $\tilde{C}(x)$ ranges between 0 and 3. In other words,

[1]The extension of applying coding both in each multiplication and across different multiplications simultaneously can be found in the full version of this paper.

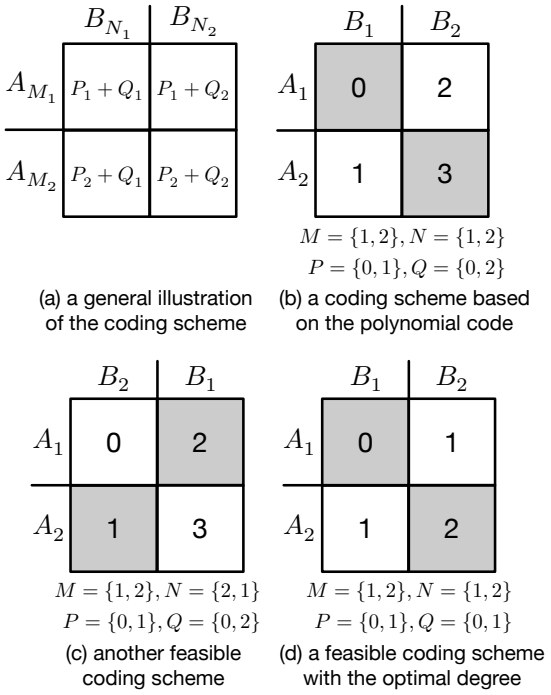

Fig. 2. The illustrations of the coding schemes achieved under the coding framework with $n = 2$.

we need to have the results of any 4 tasks to obtain the results of A_1B_1 and A_2B_2, as the coefficients of x^0 and x^3. Hence, the polynomial code can be seen as a special and non-optimal scheme that is feasible in our framework.

In Fig. 2c, we present another possible way to construct the coding scheme where $N = (2, 1)$. Hence, we can see that the exponents of A_1B_2 and A_2B_2 are placed in the counter diagonal, which are highlighted in the table. We also highlight the entries of A_1B_1 and A_2B_2 in the other examples. As M and N can be any permutations of $\{1, \ldots, n\}$, the pattern of highlighted entries can be more flexible in our coding framework. However, there should be one and only one highlighted entry in each row or each column.

Furthermore, we demonstrate an optimal coding scheme for $n = 2$ in Fig. 2d, which minimizes the number of exponents. To prove its optimality, we consider the number of exponents needed in the table. The highlighted entries must have unique exponents, and the other two entries can share the same exponents. Hence, there needs to be at least three exponents, proving the optimality of the corresponding coding scheme. This scheme is also demonstrated as the example in Sec. II. We can see that in a feasible coding scheme, the exponents in highlighted entries in the corresponding table must be unique, while the exponents in other entries can coincide which help to achieve the optimal coding scheme.

IV. Optimal Coding Scheme

A. Scopes of Parameters

In this section we present the algorithm to find the coding scheme with the optimal number of tasks to obtain the results of the job, and prove its optimality. To make it easy, we first narrow down the scopes of the parameters. Without loss of generality, we assume that elements in P and Q are non-decreasing, *i.e.*, $P_1 \leq \ldots \leq P_n$ and $Q_1 \leq \ldots \leq Q_n$. In fact, to make the exponent of A_iB_i unique, the elements in P and Q should be strictly increasing. Otherwise, if there exist two distinct integer j_1 and j_2 such that $Q_{j_1} = Q_{j_2}$, since there must exist an integer i such that $M_i = N_{j_1}$, we have $P_i + Q_{j_1} = P_i + Q_{j_2}$, *i.e.*, the exponent of $A_{M_i}B_{N_{j_1}}$ equals that of $A_{M_i}B_{N_{j_2}}$. In other words, $A_{M_i}B_{N_{j_1}}$ cannot be obtained after decoding. Therefore, we have $P_1 < \ldots < P_n$ and $Q_1 < \ldots < Q_n$.

Without loss of generality, we can also assume that $P_0 = Q_0 = 0$, or we can easily get an equivalent coding scheme by subtracting P_0 (and Q_0) from all elements in P (and Q). Moreover, in order to minimize the degree of $\tilde{C}(x)$, we can assume that $P = (0, \ldots, n-1)$ and then find the optimal Q, since otherwise the degree of corresponding $\tilde{C}(x)$ will only be larger.[2]

Given a placement of highlighted entries in the table, there can be multiple possible choices of M and N that lead to the same placement. For example, in Fig. 3a, $M = (1, 2, 3, 4)$ and $N = (2, 1, 3, 4)$, where we place corresponding A_{M_i} and B_{N_j} as the title of each row and each column, respectively. However, there can be multiple choices of M and N that lead to the same placement of highlighted entries. If we switch A_i with A_j and meanwhile B_i with B_j, for any $1 \leq i \neq j \leq n$, the highlighted entries will remain unchanged. After such a switch, the new coded tasks will remain equivalent as the original coded tasks, only having i and j switched. For example, the same entries will be highlighted if $M = (2, 1, 3, 4)$ and $N = (1, 2, 3, 4)$. Hence, we can assume, without loss of generality, that $M = (1, 2, 3, 4)$, so that the coding scheme will only depend on the value of N.

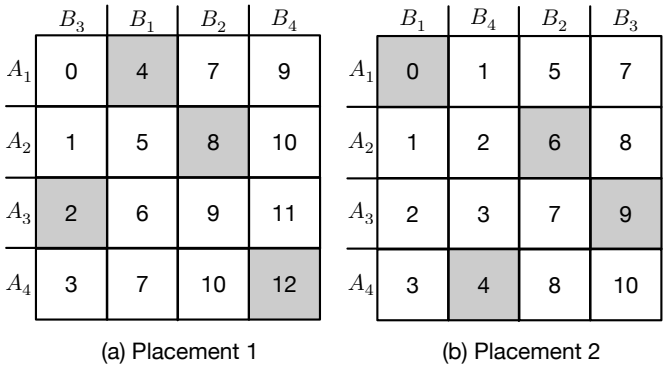

Fig. 3. Two placements of highlighted entries and their corresponding optimal coding schemes.

B. Achieving the Optimal Degree of $\tilde{C}(x)$

We first construct a coding scheme with the optimal degree of $\tilde{C}(x)$ from a given placement of highlighted entries. In

[2]It is equivalent to have $Q = (0, \ldots, n-1)$ and then choose an optimal Q. In this paper, we simply choose the value of P first and then optimize the value of Q

Alg. 1, we propose an algorithm that achieves such a coding scheme. The optimal coding scheme can then be found in two steps: 1) finding the optimal placement of highlighted entries; and 2) finding the values of Q that achieve the optimal degree in $\tilde{C}(x)$. We now discuss the second step and leave the first step in Sec. IV-C.

Algorithm 1 The optimal values of Q with a given placement of highlighted entries.

Input: N (with M fixed, the placement of highlighted entries only depends on N

Output: Q

$Q_1 = 0$
for $i \leftarrow 2$ to n **do**
  if $n - N_{i-1} \leq N_i - 1$ **then**
    $Q_i \leftarrow Q_{i-1} + N_{i-1}$
  else
    $Q_i \leftarrow Q_{i-1} + n - N_i + 1$
  end if
end for

The intuition of Alg. 1 is to make the overlaps of exponents as large as possible. As shown in Fig. 3, within two neighboring columns, the overlapped exponents go up (down) from the bottom (top) entry until reaching a highlighted entry. For example, in Fig. 3a the two entries at the bottom in the third column share the same exponents with the top two entries in the last column. We can also find exponents of 1, 2, and 3 shared in the same way in Fig. 3b, as well as 7 and 8. In the i-th column, the highlighted entry is in the N_i-th row since $M_{N_i} = N_i$. Hence, there are $N_i - 1$ entries in the gap above it and $n - N_i$ entries below it. To determine the value of Q_i, we consider if the number of entries above it is greater or less than the number of entries below the highlighted entry in the $(i-1)$-th column, and we illustrate such two cases in Fig. 4. In order to make overlaps of exponents as large as possible, if the gap at the top in the i-th column is smaller than the gap at the bottom of the $(i-1)$-th column, *i.e.*, $N_i - 1 < n - Ni - 1$, there can be at most $N_i - 1$ entries with the same exponents as the entries at the bottom in the $(i-1)$-th column. Since the last exponent in the $(i-1)$-th column is $Q_{i-1} + n - 1$, the first exponent in the i-th column should be $Q_{i-1} + n - 1 - (N_i - 2) = Q_{i-1} + n - N_i + 1$, which also equals Q_i as $P_1 = 0$.

On the other hand, if the gap at the bottom in the $(j-1)$-th column is smaller than the gap at the top of the j-th column, then the first exponent in the j-th column should be at least greater than the exponent of the highlighted entry in the $(j-1)$-th column, which equals $Q_{j-1} + N_{j-1} - 1$. In other words, Q_j should be $Q_{j-1} + N_{j-1}$.

If $n - N_{i-1} \leq N_i - 1$, then $N_{i-1} \geq n - N_i + 1$. Therefore, we can simplify Alg. 1 as $Q_i = Q_{i-1} + \max\{N_{i-1}, n - N_i + 1\}$, and the degree of $\tilde{C}(x)$ is $\sum_{i=2}^{n} \max\{N_{i-1}, n - N_i + 1\} + n - 1$.

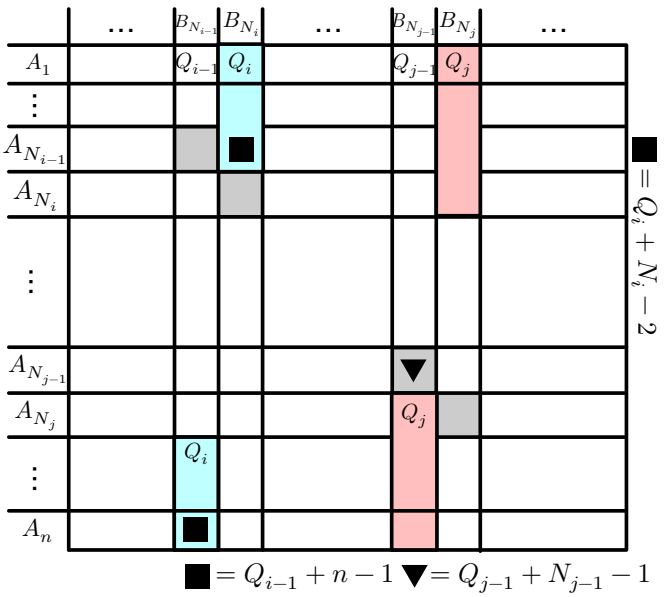

Fig. 4. Two placements of highlighted entries and their corresponding optimal coding schemes.

C. Optimal Placement of Highlighted Entries

With Alg. 1 that minimizes the exponents given a placement of highlighted entries, we now discuss how to find the optimal placement of highlighted entries. Applying Alg. 1 to two different placements with $n = 4$ in Fig. 3a and Fig. 3b, we can see that different placements of highlighted entries can lead to different degrees of $\tilde{C}(x)$.

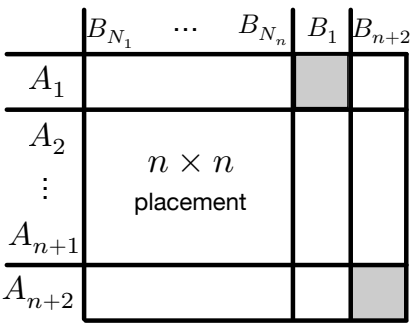

Fig. 5. The illustration of the algorithm to find the optimal placement of highlighted entries.

We now propose a placement of highlighted entries which can be proved to achieve the optimal degree in $\tilde{C}(x)$. The placement can be obtained by induction. When $n = 1$, there is one and only one possible placement which is the only entry itself, as shown in Fig. 6a. When $n = 2$, Q has two permutations, leading to two patterns of highlighted entries in Fig. 2c and Fig. 2d. We can see that the placement in Fig. 2c does not have any overlapped exponent, and hence the placement in Fig. 2d is optimal.

We now construct the placement with $n + 2$ from a placement constructed from the $n \times n$ table. As shown in Fig. 5,

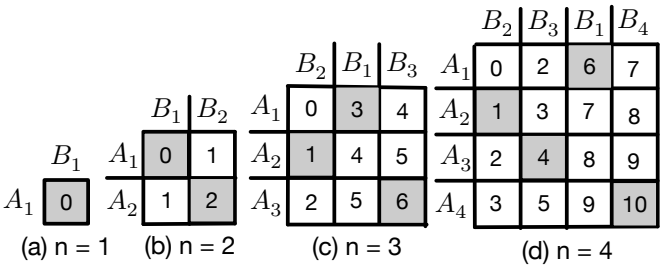

Fig. 6. Examples of the optimal placements of highlighted entries with $n = 1, 2, 3, 4$.

we first construct a placement for the $n \times n$ table and place it between the 2nd and the $(n+1)$-th row and between the 1st and the n-th column. We then highlight the top entry in the $(n+1)$-th column and the bottom entry in the $(n+2)$-th column. In Fig. 6c and Fig. 6d, we show the two placements with $n = 3$ and $n = 4$ constructed from the placement in Fig. 6a and Fig. 6b, respectively. With M fixed, N can also be determined after getting the optimal placement, and then we apply Alg. 1 to get the exponents in the table and hence obtain the value of Q. The prove of the optimality of the placement can be found in the full version of this paper.

D. Analysis

To analyze the degree of $\tilde{C}(x)$, we first count the number of unhighlighted entries with unique exponents as $U(n)$. When $n = 1$ and $n = 2$, we can directly get $U(1) = U(2) = 0$ from Fig. 6a and Fig. 6b.

For other values of n, we can get the value of $U(n)$ recursively. With the construction in Fig. 5, we can see that the right-bottom entry is always highlighted. In the two rightmost rows, all unhighlighted entries share the same exponents as those in the other row. For example, in Fig. 6d, the exponents of unhighlighted entries in the rightmost two rows are both 7, 8, and 9. Moreover, given an $(n+2) \times (n+2)$ table, in the n columns on the left, entries on the top row and the bottom row are always unhighlighted. Except the top entry in the first column and the bottom entry in the n-th column, the top entry in the i-th column can share the same exponent with the bottom entry in the $(i-1)$-th column, $i = 2, \ldots, n$. Therefore, there are only two additional unhighlighted entries with unique exponents, *i.e.*, $U(n+2) = U(n)+2$. For general values of n, we thus have

$$U(n) = \begin{cases} n-1 & n \text{ is odd;} \\ n-2 & n \text{ is even.} \end{cases}$$

Among a total of n^2 entries, there are n highlighted entries and $U(n)$ unhighlighted entries with unique exponents. Then the number of unhighlighted entries with shared exponents is $n^2-n-U(n)$. As each exponent can be shared by at most two entries (since both P and Q are strictly increasing), the total degree of $\tilde{C}(x)$ is $\frac{n^2-n-U(n)}{2}+n+U(n)-1 = \frac{n^2+n+U(n)}{2}-1$, which equals $\frac{n^2}{2}+n-\frac{3}{2}$ if n is odd or $\frac{n^2}{2}+n-2$ if n is even. In other words, we need $\frac{n^2}{2}+n-\frac{1}{2}$ (or $\frac{n^2}{2}+n-1$) tasks to decode the n matrix multiplications if n is odd (or even). To tolerate r stragglers, we only need to launch r more coded tasks with different values of x.

V. Evaluation

To evaluate the performance in terms of the total number of tasks needed, we compare the optimal coding scheme with the other two schemes we present in Sec. II. By replicating all n multiplications into $r+1$ copies, we need to have $(r+1)n$ tasks in total. If we apply the polynomial code to construct coded tasks, there need to be n^2+r tasks. Hence, the optimal coding scheme can save the number of tasks by 50% as $n \to \infty$.

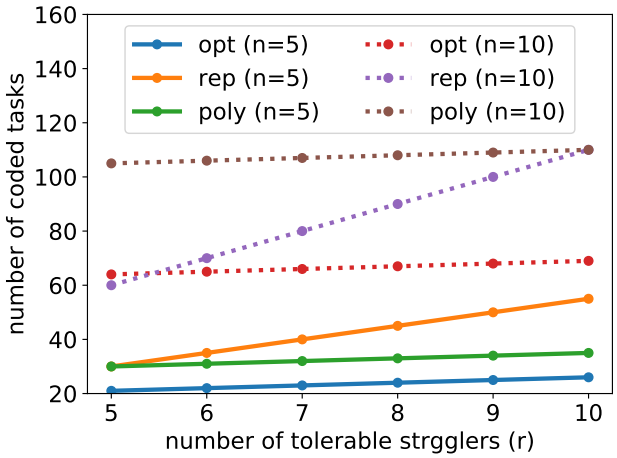

Fig. 7. Comparison of the number of coded tasks.

As shown in Fig. 7, we compare the number of coded tasks of the three schemes above. We can see that although replicated tasks (rep) can save a limited number of tasks with a small number of stragglers ($r = 5$), the number of tasks required increases significantly faster than the other two schemes. The coding scheme based on polynomial codes (poly) requires up to 42.9% and 64.1% more tasks than the optimal coding scheme (opt) when $n = 5$ and $n = 10$, respectively.

VI. Conclusion

Coded computing for distributed matrix multiplication has been demonstrated to efficiently tolerate stragglers. However, existing coding schemes can only create coded tasks to tolerate stragglers within one matrix multiplication only. In this paper, we propose a coding framework that can optimally create coded tasks across multiple matrix multiplications. We demonstrate that compared to existing schemes, our proposed scheme can save the number of coded tasks by up to 64.1%.

Acknowledgments

This paper is based upon work supported by the National Science Foundation under Grant No. CCF-1910447.

REFERENCES

[1] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray Failure: The Achilles' Heel of Cloud-Scale Systems," in *USENIX Conference on Hot Topics in Operating Systems (HotOS)*, 2017.

[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, 2018.

[3] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 3368–3376.

[4] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster," in *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2013.

[5] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," *Proceedings of the VLDB Endowment*, vol. 6, no. 5, pp. 325–336, 2013.

[6] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[7] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy," in *Workshop on Systems for ML and Open Source Software at NeurIPS 2018*, 2018.

[8] Z. Jia and S. A. Jafar, "Cross Subspace Alignment Codes for Coded Distributed Batch Computation," *arXiv:1909.13873*.

[9] Q. Yu and A. S. Avestimehr, "Entangled Polynomial Codes for Secure, Private, and Batch Distributed Matrix Multiplication: Breaking the "Cubic" Barrier," in *IEEE International Symposium on Information Theory (ISIT)*, 2020.

[10] Z. Chen, Z. Jia, Z. Wang, and S. A. Jafar, "GCSA Codes with Noise Alignment for Secure Coded Multi-Party Batch Matrix Multiplication," in *IEEE International Symposium on Information Theory (ISIT)*, 2020.