Sequence-aware Coding for Matrix Multiplication with Arbitrary Recoverability

Yuchun Zou
Graduate Center
City University of New York
New York, NY

Jun Li
Queens College & Graduate Center
City University of New York
New York, NY

***Abstract*—Matrix multiplication is a crucial operation in many data-intensive workloads. Given the large size of matrices in today's workloads, it is common to split the computation into tasks executed on different servers. As stragglers are common in distributed computing, various coding schemes have been proposed to mitigate stragglers, including some even leveraging the partially completed results from stragglers by splitting each task into subtasks. However, existing schemes have ignored the order of execution, making them unnecessarily complex for encoding and decoding. In this paper, we propose a series of constructions of straggler-leveraging coding schemes for matrix multiplication. We consider the execution order of subtasks and then construct the coding schemes based on the probability of an uncoded subtask being recovered by a coded subtask. As a result, our coding schemes can significantly save the encoding and decoding complexities while maintaining an arbitrarily controllable recoverability of incomplete uncoded subtasks.**

I. Introduction

In many data-intensive workloads, such as machine learning [1], data analytics [2], and signal processing [3], [4], matrix multiplication is a common operation that dominates the total time of computation. With the fast-increasing sizes of datasets, the size of matrix multiplication has also grown beyond the capability of a single server. Therefore, it is common to parallelize the computation into multiple *tasks* that are executed on different servers called *workers*. As a toy example, to compute the multiplication of a matrix A and a vector x, we can split A into two submatrices such that $A \cdot x = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot x = \begin{bmatrix} A_1 \cdot x \\ A_2 \cdot x \end{bmatrix}$, and compute $A_1 \cdot x$ and $A_2 \cdot x$ on two workers.

Ideally, when all workers have the same configuration and the overall workload is equally split into all tasks, all workers should complete their tasks at the same time. However, it is often observed in practice that some workers may become *stragglers* whose performance is significantly slower than others [5], [6]. For example, it has been measured in AWS EC2 that 5% of servers may be stragglers whose performances are 2x-5x worse than other servers [7], which can easily become the bottleneck of the workload.

To mitigate stragglers, a common approach is to launch additional tasks on more workers to replace those executed on stragglers [8]–[14]. For example, if one straggler is anticipated, we can duplicate each task on two workers such that any single task running on a straggler can be ignored. However, this replication-based approach suffers from significantly high resource overhead, especially when there are more stragglers.

Besides replication, many coding-based approaches have been proposed recently (*e.g.*, [1], [15]–[17]). Instead of directly replicating the input of existing tasks, coded tasks can be created such that their input matrices are encoded from the input matrices of existing tasks. In the above example of computing $A \cdot x$, a coded task may be created as $(A_1 + A_2) \cdot x$, which equals the sum of $A_1 x$ and $A_2 x$. Hence, we can tolerate any single straggler by adding only one additional worker. By an MDS code, we can easily tolerate any r stragglers with just r additional workers [1].

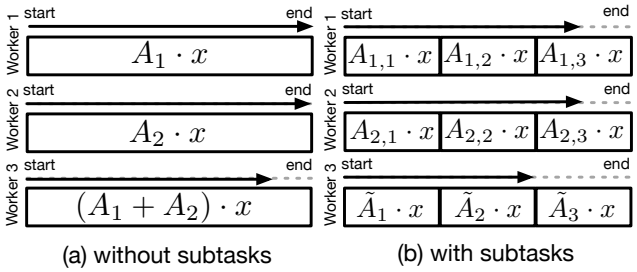


Fig. 1: A comparison between coded matrix multiplication without and with subtasks.

With additional coded tasks, tasks executed on stragglers are typically disregarded. Fig. 1a illustrates that Ax can be recovered from any two tasks, while the last one is disregarded no matter if the straggler makes no progress or is just slightly slower. Hence, the resources on workers with disregarded tasks are wasted, which could have been used to further lower the completion time. For example, if we can predict stragglers, we can then assign a lower amount of workload on such workers. However, predicting stragglers is impractical as they may be caused by many factors, such as resource sharing, I/O bottlenecks, and maintenance activities [1], [5], [7], [18]. An incorrect prediction may further affect the overall performance. Therefore, approaches that fit the workload dynamically with the worker's performance are desirable.

To dynamically adjust the workload of each worker, a task needs to be further split into *subtasks*, and then coded subtasks

This article was written with support from the 2023 CUNY Faculty Fellowship Publication Program. We thank Jane Alexander, William Carr, Matthew Junge, Bianca Sosnovski, Joshua Tan, and Nicholas Vlamis for their many suggestions for improving this paper.

can be encoded from such (uncoded) subtasks [13], [19]–[34]. As shown in Fig. 1b, we can create three uncoded subtasks in each task by further splitting each A_i, $i = 1, 2$, horizontally into $A_{i,j}$, $j = 1, 2, 3$. The coded task also correspondingly includes three coded subtasks $\tilde{A}_j x, j = 1, 2, 3$, where each $\tilde{A}_j$ is encoded from all four submatrices of A, $j = 1, 2, 3$, *e.g.*, by an MDS code. In this way, we can recover Ax from any 6 among the total 9 subtasks. In other words, slower workers can dynamically complete a lower amount of workload without being predicted as stragglers in advance. However, all coded subtasks are encoded from all uncoded subtasks equally, making the complexities of encoding and decoding very high. It has been measured in Microsoft Azure that the time spent on encoding and decoding can match or even exceed the time of computation [19].

In this paper, we argue that coded subtasks should not be encoded in the same way. We consider the order of subtasks executed within a task. As subtasks within a task are typically executed in a fixed order, a subtask executed earlier naturally has a lower probability to be incomplete than another one executed later. As a coded subtask is added to recover the overall result when some uncoded subtask is incomplete, a subtask with a higher probability to be incomplete should be encoded into coded subtask with a higher priority. Therefore, we propose a coding framework of coding schemes where coded subtasks can be created based on their order in the task. Specifically, our framework allows a flexible parameter to control how likely a coded subtask can be used to recover the overall result. Fan *et al.* [34] have also proposed coding constructions that leverage the sequence of execution, but the constructions are *ad hoc* which only works for limited combinations of parameters and only only works for matrix-vector multiplication. To the best of our knowledge, this is the first work that can arbitrarily achieve a tradeoff between the mitigation of stragglers and the complexity of the coding scheme. Moreover, we demonstrate that our coding scheme can support not only matrix-vector multiplication, but also matrix-matrix multiplication.

II. System Model

We assume that the workload is matrix multiplication. For now, we just consider a special case of matrix-vector multiplication, and will extend the model to matrix-matrix multiplication in Sec. VI. Hence, we consider a workload of computing $A \cdot x$, and the workload will be computed on n workers in parallel. Each worker computes a task containing a fixed number of subtasks, which are also executed in a fixed order. Once a subtask is complete, its result will be uploaded to another server called *master*. The master keeps receiving results of subtasks from all workers until such results are sufficient for recovering Ax, and then the master may instruct all workers to stop if they still have subtasks incomplete.

To compute $A \cdot x$ in parallel, we split A into n submatrices horizontally, *i.e.*, $A_1, \ldots, A_n$, such that $A \cdot x = \begin{bmatrix} A_1 \cdot x \\ \vdots \\ A_n \cdot x \end{bmatrix}$. Therefore, A_i will be assigned to Worker i, $i = 1, \ldots, n$. Each worker then further splits A_i to create u subtasks ($u \leq s$), called uncoded subtasks, such that $A_i = \begin{bmatrix} A_{i,1} \\ \vdots \\ A_{i,u} \end{bmatrix}$.

Besides uncoded subtasks, Worker i will also have $c = s - u$ additional coded subtasks $\tilde{A}_{i,j} \cdot x$, $j = 1, \ldots, c$, $i = 1, \ldots, n$, in order to mitigate potential stragglers. Conventionally, $\tilde{A}_{i,j}$ can be encoded by a $((u + c)n, un)$ systematic MDS codes as a linear combination of all submatrices of A, *i.e.*, $\{A_{i,j} | i = 1, \ldots, n, j = 1, \ldots, u\}$ [20]. We name this scheme as global MDS codes.

In this paper, we argue that not all subtasks have an equal probability of being incomplete when the completed results are sufficient for decoding, as subtasks are not started concurrently but sequentially. A naive application of this idea is placing all uncoded subtasks at the beginning on each worker. However, all coded subtasks are still considered equally likely to be incomplete. Therefore, we can save the complexities of coded subtasks based on their probability of being incomplete.

We now give an intuition of the coding schemes in this paper. For convenience, we define $C_{i,j} = A_{i,j} \cdot x$ and $\tilde{C}_{i,j} = \tilde{A}_{i,j} \cdot x$. We assume that Worker i executes subtasks in the order of $C_{i,u}, \ldots, C_{i,1}, \tilde{C}_{i,1}, \ldots, \tilde{C}_{i,c}$, $i = 1, \ldots, n$, such that uncoded subtasks are always executed before coded subtasks. Moreover, note that the order of uncoded subtasks is decreasing for convenience only. The completion of all uncoded subtasks is sufficient to obtain $C = Ax$ without decoding. When there is one uncoded subtask incomplete, the only possibility is having the last uncoded subtask on some worker incomplete, *i.e.*, $C_{i,1}$. Moreover, if we anticipate this incomplete subtask can be recovered by one coded subtask, this one must be the first coded subtask on some other worker, *i.e.*, $\tilde{C}_{i',1}$. More generally, an uncoded subtask $C_{i,j}$ with a smaller j is more likely to be incomplete and a coded subtask $\tilde{C}_{i,j}$ with a smaller j is also more likely to be needed. Intuitively, the complexity of $\tilde{A}_{i,j}$ in a coded subtask should grow with the increasing of j, and any $A_{i,j}$ becomes less likely to be encoded into a coded subtask with the increasing of j.

In this paper, our objective is to find the coding scheme such that given the first un completed subtasks, the probability that Ax can be recovered is at least θ, called the *θ-recoverability*, $0 \leq \theta \leq 1$. Obviously, global MDS codes can guarantee $\theta = 1$, indicating strong mitigation of stragglers. When $\theta = 0$, on the other hand, no coded subtask is necessary, and also there is no mitigation of any kind of stragglers. Hence, we can achieve a tradeoff between the mitigation of stragglers and the complexity of the coding scheme.

To construct the coding scheme, we first compute the chance that an $A_{i,j}$ is needed by an $\tilde{A}_{i',j'}$ (in Sec. III). Then we present algorithms to give the code construction for matrix-vector multiplication while achieving the θ-recoverability (in Sec. IV and Sec. V). We also extend the construction to matrix-matrix multiplication (in Sec. VI).

III. Placements of Completed Subtasks

As we aim to recover Ax with a probability of at least θ, when un subtasks — no matter uncoded or coded — have been completed, we need to understand how likely an uncoded subtask needs to be recovered by a coded subtask.

We first consider the number of valid placements of completed subtasks, where a valid placement must have all subtasks completed sequentially. In other words, any incomplete subtasks should appear after all completed subtasks on the same worker. Assume that x_i is the number of completed subtasks on Worker i, and then we have

$$\sum_{i=1}^{n} x_i = un, \quad s.t. \quad 0 \le x_i \le u+c, i = 1, \dots, n. \quad (1)$$

Therefore, the number of valid placements equals the number of integer solutions of (1). This is a counting problem and can be solved by the inclusion-exclusion principle.

We define S as the set of all integer solutions of $\sum_{i=1}^{n} x_i = un$, *s.t.* $0 \le x_i, i = 1, \dots, n$. S corresponds to (1) with all upper bounds *can be* violated. Then we have $|S| = \binom{un+n-1}{n-1}$.

Similarly, we define S_i as the set of all integer solutions in S with the upper bound of only x_i *is* violated, *i.e.*, $x_i \ge u+c+1$ which is equivalent to $x_i - (u+c+1) \ge 0$. Thus we have $|S_i| = \binom{un-(u+c+1)+n-1}{n-1}$.

When we have two variables in (1) violate their upper bounds, *i.e.*, $x_i \ge u+c+1$ and $x_j \ge u+c+1$, $1 \le i \ne j \le n$, we can similarly have $|S_i \cap S_j| = \binom{un-2(u+c+1)+n-1}{n-1}$. Furthermore, if m variables violate their upper bounds, we have the number of such integer solutions as $\binom{un-m(u+c+1)+n-1}{n-1}$. Eventually, there can be at most $\left\lfloor \frac{un}{u+c+1} \right\rfloor$ variables violating their upper bounds as $\sum_{i=1}^{n} x_i$ must be no less than the sum of all violated upper bounds.

By the inclusion-exclusion principle, the number of solutions of (1) equals

$$\begin{aligned}
&S - \sum_{i=1}^{n} |S_i| + \sum_{1 \le i < j \le n} |S_i \cap S_j| + \cdots \\
=&\binom{un+n-1}{n-1} - \binom{n}{1}\binom{un-(u+c+1)+n-1}{n-1} + \cdots \\
&+ (-1)^m \binom{n}{m}\binom{un-m(u+c+1)+(n-1)}{n-1} + \cdots \\
=&\sum_{m=0}^{\lfloor \frac{un}{u+c+1} \rfloor} (-1)^m \binom{n}{m}\binom{un-m(u+c+1)+(n-1)}{n-1} \\
\triangleq&|N(n,u,c)|,
\end{aligned}$$

where we define $N(n,u,c)$ as the set of all valid placements.

Moreover, we are also particularly interested in special cases such that a coded subtask may *not* be needed to recover an uncoded subtask. As shown in Fig. 2, we can see that all uncoded subtasks $C_{i,j}$ with $j > 1$ are completed and all coded subtasks $\tilde{C}_{i',j'}$ with $j' > 1$ are incomplete. Therefore, if an uncoded subtask is incomplete, it can only be recovered from a coded subtask with $j' \le 1$. Reversely, a coded subtask in this case also only needs to recover an uncoded subtask with $j \le 1$. Therefore, coded subtasks with $j' \le 1$ can be encoded as combinations of uncoded subtasks with $j \le 1$ only. This example is just one placement in $N(4,2,2)$. However, how many placements are there with $j \le 1$ and $j' \le 1$? As the first subtasks $C(i,2)$ are always completed and the last subtasks $\tilde{C}(i',2)$ are always incomplete, $i = 1, \dots, n$, we only need to consider the placements of four completed subtasks among all places in-between. Therefore, the number of such placements is $|N(4,1,1)|$.

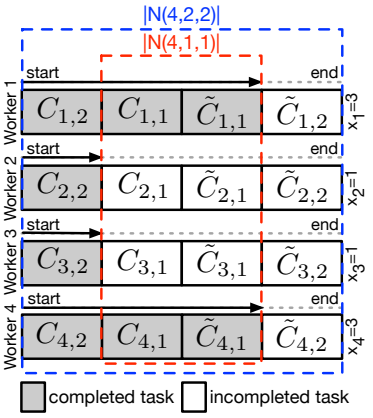


Fig. 2: An example of placements of 8 completed subtasks with $n = 4$ and $u = c = 2$.

Note that a placement in $N(n,u,c)$ does not guarantee that some worker's first (uncoded) subtask is incomplete or some worker's last (coded) subtask is completed, as it does not violate the requirement of $0 \le x_i \le u+c$. Hence, we define $N_c(n,u,c)$ as the set of placements in $N(n,u,c)$ that at least one coded subtask $\tilde{C}_{i',j'}$ with a dedicated $j' = c$ is completed. By this definition, we have

$$|N_c(n,u,c)| = \begin{cases} |N(n,u,c)| & c = 1 \\ |N(n,u,c)| - |N(n,u,c-1)| & c > 1. \end{cases}$$

In Fig. 3, we illustrate values of $|N_c(n=5,u,c)|$ with $c = 1,2,3$ and $u = 1, \dots, 4$. With a larger u, the number of corresponding placements in $N_c(n=5,u,c)$ also grows, as the higher u gives more available locations to place the completed uncoded subtasks. In other words, more incomplete subtasks can be recovered if the coding scheme can recover such placements. Meanwhile, it means that the coding scheme

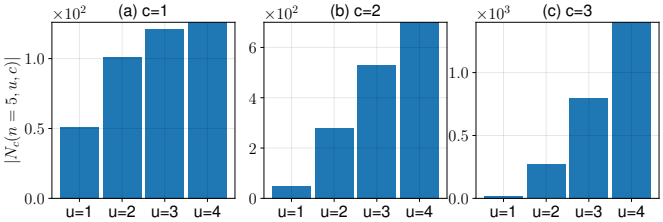


Fig. 3: The numbers of placements in $N(n=5,u,c)$.

becomes more complex. Hence, we will exploit such a tradeoff to achieve the θ-recoverability in Sec. IV. Moreover, with c increased, the ratio between $|N_c(n=5,u,c)|$ and $|N_c(n=5,u=4,c)|$ also decreases, meaning that the corresponding coded subtasks are more likely to have higher complexities. Therefore, if workers are not heavily straggling, we can also expect a lower decoding complexity.

IV. Static Construction

Based on the intuition in Fig. 3 where $|N_c(n,u,c)|$ always increases with u, we first propose a static construction of the sequence-aware coding that supports θ-recoverability. To construct the coding scheme, we first introduce the following theorem.

Theorem 1. *Given valid c and u, $N_c(n,u,c) \subseteq N_c(n,u+1,c)$.*

Proof. Given any one placement in $N_c(n,u,c)$, we can add one uncoded subtask on top of existing subtasks on all workers, and then we have a placement with $u+1$ uncoded subtasks and c coded subtasks in $N_c(n,u+1,c)$. Therefore, $N_c(n,u,c) \subseteq N_c(u,u+1,c)$ □

Compared to $N_c(n,u,c)$, the additional placements in $N_c(n,u+1,c)$ also suggest that more uncoded subtasks are needed in the coded subtasks $\tilde{C}_{i,c}$. As there should be u uncoded subtasks and c coded subtasks on each worker, to achieve the θ-recoverability, we should cover at least $\theta|N_c(n,u,c)|$ placements. Hence, we need to find the minimum u' such that $|N_c(n,u',c)| \geq \theta|N_c(n,u,c)|$ where $u' \leq u$. The coding scheme can thus be constructed as in Alg. 1.

Algorithm 1 The static construction of the sequence-aware coding with θ-recoverability.

1: **for** $j = 1,\ldots,c$ **do**
2: $u_j = \min\left\{u' \middle| \frac{|N_c(n,u',c)|}{|N_c(n,u,c)|} \geq \theta\right\}$
3: encode $\tilde{A}_{i,j}, i = 1,\ldots,n$ with an MDS code from $\{A_{i,j}|j \leq u_j, i = 1,\ldots,n\}$

Intuitively, with an increased θ, we can achieve a higher level of recoverability, as more cases of incomplete uncoded subtasks can be recovered. We hence present the actual recoverability of the static construction in Fig. 4, with $n=5$, $u=4$, and $c=3$. We measure the recoverability by randomly generating 10000 cases of un sequentially completed subtasks in total on all workers, and the actual recoverability is measured as the ratio of the cases that the result of all subtasks can be recovered to all cases. In Fig. 4, we change the value of θ from 0 to 1, and we can see that the actual recoverability is always above θ. In fact, we can see the recoverability increases when it becomes close to θ, and such turning points directly correspond to $\frac{|N_c(n,u',c)|}{|N_c(n,u,c)|}, u' = 1,\ldots,u$.

We also measure the complexity of coded subtasks in terms of the number of uncoded subtasks they are encoded from. When all coded subtasks are encoded from all un uncoded subtasks, it is equivalent to global MDS codes. From Fig. 4 we can see that unless the value of θ is close 1, the average

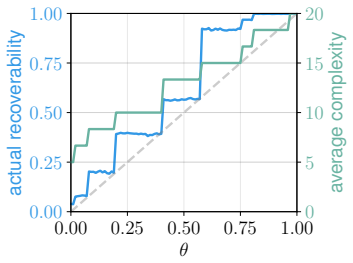


Fig. 4: The actual recoverability and average complexity of the static construction with $n=5$, $u=4$, and $c=3$.

complexity of coded subtasks can be significantly lower, and we can easily achieve a tradeoff between the recoverability and the complexity.

We also observe from Fig. 4 that with most values of θ, the actual recoverability is much more than θ, suggesting that there is still room to further reduce the complexity. The reason is that we only change the coding scheme when θ is larger than some threshold in Line 2 of Alg. 1, making the coding scheme less flexible to the change of θ. Hence, we present another construction below where the coding scheme can be adjusted with a finer granularity.

V. Stochastic Construction

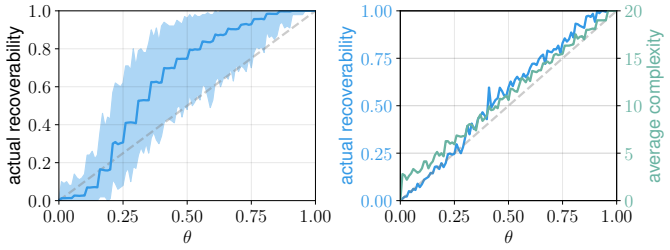
In the static construction, we choose the number of uncoded subtasks encoded into a coded subtask purely based on the rate between $|N_c(n,u',c)|$ and $|N_c(n,u,c)|$, $u' = 1,\ldots,u$. Hence, when θ is not increased to the next rate, the coding scheme remains unchanged. In this section, we propose a different method where a coded subtask is encoded from uncoded subtask more stochastically, which eventually achieves the tradeoff between recoverability and the complexity with a finer granularity.

Algorithm 2 The stochastic construction of the sequence-aware coding with θ-recoverability.

1: $C = \lceil \theta un \rceil$
2: **for** $j = 1,\ldots,c$ **do**
3: **for** $i = 1,\ldots,n$ **do**
4: $P = \emptyset$
5: **while** $|P| \leq C$ **do**
6: ρ is a random number between 0 and 1
7: $u_{i,j} = \min\left\{u' \middle| \frac{|N_c(n,u',c)|}{|N_c(n,u,c)|} \geq \rho\right\}$
8: **for** $i' = 1,\ldots,u_{i,j}$ **do**
9: **if** there exists a random integer j', $1 \leq j' \leq n$, such that $(i',j') \notin P$ **then**
10: $P = P \cup \{(i',j')\}$
11: encode $\tilde{A}_{i,j}$ with an MDS code from $\{A_{i,j}|(i,j) \in P\}$

This stochastic construction is given in Alg. 2, where θ is not just an indicator of the recoverability, but also that of the complexity of the coding scheme. Specifically, each coded subtask should be encoded from $\lceil \theta un \rceil$ uncoded subtasks in

principle. Such $\lceil \theta un \rceil$ uncoded subtasks, however, should not be chosen upwards as in the static construction. Instead, we choose such uncoded subtasks stochastically, according to their chance to be used, *i.e.*, $\frac{|N_c(n,u',c)|}{|N_c(n,u,c)|}$.



(a) before extension (b) after extension

Fig. 5: The actual recoverability and average complexity of the stochastic constructions with $n = 5$, $u = 4$, $c = 3$, $\alpha = 100$, and $\beta = 1000$.

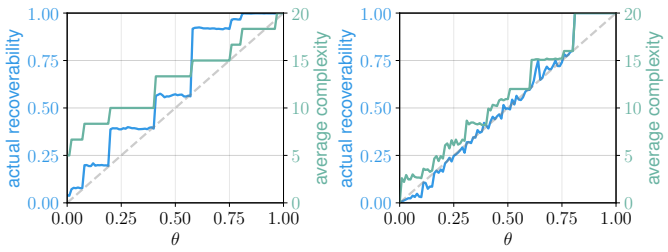
We illustrate the actual recoverability of this construction in Fig. 5a. As the stochastic nature of the construction, the actual recoverability varies one after another, and thus we repeat the construct $\alpha = 100$ times, measure their actual recoverability with $\beta = 1000$ tests, and eventually obtain the average recoverability. Fig. 5a hence illustrates the mean, maximum, and minimum of the actual recoverability with θ between 0 and 1. We can see that the average curve becomes smoother than the static construction. Moreover, the variety of the actual recoverability in such α cases allows us to find the case whose recoverability is even closer to θ. From this observation, we extend the stochastic construction by choosing the best coding scheme among all α cases. In particular, by evaluating the recoverability from β tests, we pick up the one with its recoverability closest to θ. We show the actual recoverability in Fig. 5b which becomes very close to θ. The average distance between the actual recoverability and θ is reduced from 0.116 (in Fig. 4) to 0.058. Similarly, we also measure the average complexity of coded subtasks, whose distance from θun is also reduced from 3.05 to 1.06.

VI. Matrix-Matrix Multiplication

We now extend the sequence-aware coding to matrix-matrix multiplication. Assume that we need to compute $A \cdot B$, and both A and B are two large matrices. The un uncoded subtasks can then be constructed by splitting A into $\begin{bmatrix} A_1 \\ \vdots \\ A_n \end{bmatrix}$ and $\begin{bmatrix} B_1 & \cdots & B_u \end{bmatrix}$, and then we have $C_{i,j} = A_i \cdot B_j$. We now apply the static and stochastic constructions above to matrix-matrix multiplication.

We can directly extend the static construction to matrix-matrix multiplication. Similar to Alg. 1, after getting u_j, we can let $\tilde{C}_{i',j'} = \left(\sum_{i=1}^{n} A_i x_{i',j'}^{i}\right) \cdot \left(\sum_{j=1}^{u_j} B_j x_{i',j'}^{jn}\right) = \sum_{i=1}^{n}\sum_{j=1}^{u_j} A_i B_j x_{i',j'}^{jn+i} = \sum_{i=1}^{n}\sum_{j=1}^{u_j} C_{i,j} x_{i',j'}^{jn+i}$. The value of $x_{i',j'}$ among all coded subtasks should be unique.

On the other hand, uncoded subtasks encoded into a coded subtask are randomly chosen in the stochastic construction. However, in the coded subtask of matrix-matrix multiplication, they cannot be chosen purely randomly. Specifically, if we encode $\tilde{C}_{i,j}$ as $\left(\sum_{i\in P_{i',j'}} A_i x_{i',j'}^{i}\right) \cdot \left(\sum_{j\in Q_{i',j'}} B_j x_{i',j'}^{jn}\right) = \sum_{i\in P_{i',j'}}\sum_{j\in Q_{i',j'}} C_{i,j} x_{i',j'}^{jn+i}$, the number of uncoded subtasks inside must be $|P_{i',j'}| \cdot |Q_{i',j'}|$, which cannot be an arbitrary integer. Therefore, we need to change Line 8 – Line 11 in Alg. 2. After getting $u_{i,j}$, we let $Q_{i',j'} = \{1, \ldots, u_{i,j}\}$ and $P_{i',j'}$ be a random subset of $\{1, \ldots, n\}$ with $\left\lceil \frac{N}{u_{i,j}} \right\rceil$ elements.



(a) static construction (b) stochastic construction

Fig. 6: The actual recoverability and average complexity of the static and stochastic constructions for matrix-matrix multiplication with $n = 5$, $u = 4$, $c = 3$, $\alpha = 100$, and $\beta = 1000$.

From Fig. 6a, we can see that the static construction maintains almost the same performance as in Fig. 4, and the performance of the stochastic construction becomes different, especially when $\theta > 0.8$. It is because when $n = 5$ and $u = 4$, there are 20 uncoded subtasks. With $\theta > 0.8$, there need to be more than 16 subtasks. As $u = 4$, the size of $P_{i',j'}$ must be 5, leaving the construction with no choice but being equivalent as global MDS codes.

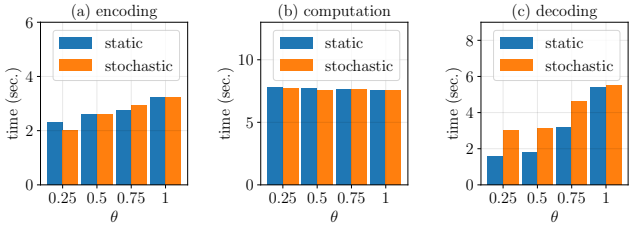


Fig. 7: The time of encoding, computation, and decoding in the matrix-matrix multiplication with $n = 10$, $u = 10$, and $c = 3$.

We also implement matrix-matrix multiplication with the static and stochastic constructions using mpi4py. We multiply two matrices of size 3000×3000 on a local cluster with $n = 10$ workers. Each worker runs $u = 10$ uncoded subtasks and $c = 4$ coded subtasks. To simulate stragglers, we add random delays on each subtask following an exponential distribution $exp(\frac{1}{0.2})$. We measure the time of encoding, computation, and decoding by running the job 50 times, with different values of θ. We choose the values of θ between 0.25 and 1. When $\theta = 1$, the

scheme becomes the same as global MDS codes. We present the results in Fig. 7.

With $\theta = 1$, the time of encoding and decoding is 52.2% of the whole job. We can see from Fig. 7b that the value of θ does not affect the computation time, but encoding time and decoding time significantly. Fig. 7a shows that the static construction can save the encoding time by up to 28.0% and the stochastic construction can save it by up to 37.5%. Again, it shows that the encoding complexity can be more precisely controlled by θ in the stochastic construction. Similarly, we can see in Fig. 7c that the decoding time can be saved by up to 45.0% with the static construction and up to 70.6% with the stochastic construction.

VII. Conclusions

We propose coding schemes for matrix multiplication that leverage the results on stragglers to lower the computation time and meanwhile save the complexities of encoding and decoding. This is achieved by taking into account the order of subtasks in the code construction. The proposed coding schemes offer θ-recoverability, enabling an arbitrary tradeoff between recoverability and complexity. Our evaluation shows that, in comparison to global MDS codes, our coding schemes do not impact computation time, but considerably reduce the time required for encoding and decoding.

References

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[2] M. H. Qasem, A. A. Sarhan, R. Qaddoura, and B. A. Mahafzah, "Matrix multiplication of big data using MapReduce: A review," in *2017 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS)*, Dec. 2017, pp. 1–6.

[3] H. Zhou, J. Dong, J. Cheng, W. Dong, C. Huang, Y. Shen, Q. Zhang, M. Gu, C. Qian, H. Chen, Z. Ruan, and X. Zhang, "Photonic matrix multiplication lights up photonic accelerator and beyond," *Light: Science & Applications*, vol. 11, no. 1, p. 30, Feb. 2022.

[4] I. Palacios, M. Medina, and J. Moreno, "Matrix Multiplication on Digital Signal Processors and Hierarchical Memory Systems," in *Computer Science: Research and Applications*, R. Baeza-Yates and U. Manber, Eds. Boston, MA: Springer US, 1992, pp. 473–483.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[6] J.-C. Bolot, "End-to-end packet delay and loss behavior in the internet," in *Conference Proceedings on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '93. New York, NY, USA: Association for Computing Machinery, Oct. 1993, pp. 289–298.

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

[8] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 715–722, 2016.

[9] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 185–198.

[10] Z. Qiu and J. F. Pérez, "Evaluating replication for parallel jobs: An efficient approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2288–2302, 2016.

[11] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1, pp. 599–600, 2014.

[12] K. Lee, R. Pedarsani, and K. Ramchandran, "On scheduling redundant requests with cancellation overheads," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1279–1290, 2017.

[13] K. Narra, Z. Lin, M. Kiamari, S. Avestimehr, and M. Annavaram, "Distributed Matrix Multiplication Using Speed Adaptive Coding," *arXiv preprint arXiv:1904.07098*, 2019.

[14] N. Ferdinand, B. Gharachorloo, and S. C. Draper, "Anytime exploitation of stragglers in synchronous stochastic gradient descent," in *IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018.

[15] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[16] ——, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.

[17] P. Soto, J. Li, and X. Fan, "Dual entangled polynomial code: Three-dimensional coding for distributed matrix multiplication," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5937–5945.

[18] 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.

[19] X. Fan, P. Soto, X. Zhong, D. Xi, Y. Wang, and J. Li, "Leveraging Stragglers in Coded Computing with Heterogeneous Servers," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. Hang Zhou, China: IEEE, Jun. 2020, pp. 1–10.

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

[21] A. B. Das, L. Tang, and A. Ramamoorthy, "C^{3}LES: Codes for Coded Computation that Leverage Stragglers," in *IEEE Information Theory Workshop (ITW)*. IEEE, 2018, pp. 1–5.

[22] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1620–1624.

[23] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *arXiv preprint arXiv:1805.09934*, 2018.

[24] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 1777–1781.

[25] S. Kiani, N. Ferdinand, and S. C. Draper, "Hierarchical coded matrix multiplication," in *16th Canadian Workshop on Information Theory (CWIT)*. IEEE, 2019, pp. 1–6.

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

[27] Y. Sun, J. Zhao, S. Zhou, and D. Gunduz, "Heterogeneous coded computation across heterogeneous workers," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[28] D. Kim, H. Park, and J. Choi, "Optimal load allocation for coded distributed computation in heterogeneous clusters," *arXiv preprint arXiv:1904.09496*, 2019.

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

[30] K. G. Narra, Z. Lin, M. Kiamari, S. Avestimehr, and M. Annavaram, "Slack squeeze coded computing for adaptive straggler mitigation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–16.

[31] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2729–2733.

[32] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2654–2658.

[33] K. T. Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2020, pp. 237–246.

[34] X. Fan, P. Soto, Y. Zou, X. Su, and J. Li, "Sequence-Aware Coding for Leveraging Stragglers in Coded Matrix Multiplication," in *IEEE International Conference on Communications (ICC)*, 2023.