Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes

Tavor Baharav
UC Berkeley
tavorb@berkeley.edu

Kangwook Lee
KAIST
kw1jjang@kaist.ac.kr

Orhan Ocal
UC Berkeley
ocal@eecs.berkeley.edu

Kannan Ramchandran
UC Berkeley
kannanr@eecs.berkeley.edu

***Abstract*—Distributed computing allows for large-scale computation and machine learning tasks by enabling parallel computing at massive scale. A critical challenge to speeding up distributed computing comes from *stragglers*, a crippling bottleneck to system performance [1]. Recently, coding theory has offered an attractive paradigm dubbed as coded computation [2] for addressing this challenge through the judicious introduction of redundant computing to combat stragglers. However, most existing approaches have limited applicability if the system scales to hundreds or thousands of workers, as is the trend in computing platforms. At these scales, previously proposed algorithms based on Maximum Distance Separable (MDS) codes are too expensive due to their hidden cost, i.e., computing and communication costs associated with the encoding/decoding procedures. Motivated by this limitation, we present a novel coded matrix-matrix multiplication scheme based on d-dimensional product codes. We show that our scheme allows for order-optimal computation/communication costs for the encoding/decoding procedures while achieving near-optimal task runtime.**

I. Introduction

Distributed computing is becoming the main mode of computation for modern large-scale machine learning and data analytics jobs. However, one problem in distributed computing systems is that there are *stragglers*, which are compute nodes that are slowed down due to unpredictable factors [1]. At the same time, distributed computing systems are trending to massive scales (e.g., using AWS Lambda [3]). The straggler problem is even more evident in such large-scale computing systems; for instance, in [3], about 5% to 10% of 3000 distributed workers are observed to be stragglers. Left untreated, these severely impact latency, as the total execution time depends on the slowest worker.

In [2], the authors apply erasure codes to distributed matrix multiplication algorithms, and propose a new framework called *coded computation* to combat stragglers, significantly speeding up distributed computing algorithms. Subsequently, coded computation has been shown to speed up a variety of computing tasks such as large-scale matrix multiplications [4], distributed gradient computing, convolutions [5], Fourier transforms [6], personalized PageRank [7], etc.

The archetypal workflow of coded computation schemes is illustrated in Fig. 1. Here, we assume a standard master-worker model but one may also consider a decentralized (or masterless) setting. First, unless the input data is already stored as encoded across the workers, the master node needs to encode the input data and share the encoded data with the distributed

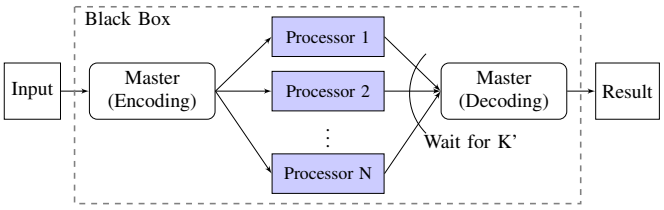

Fig. 1: Distributed Computing System Architecture

workers. Hence, the encoding time consists of 1) the time to encode data (T_{enc}) and 2) the time to transmit the encoded data from the master to the workers ($T_{\text{comm,in}}$). Upon receiving the input data, the distributed workers start working on their tasks. Further, the task results need to be transmitted from each worker to the master. Denote the task runtime (including the time to communicate the task result to the master) of each worker by T_i. Once a *decodable* set of task results are collected, the master starts running the decoding algorithm to obtain the final output, and the time to run this stage is called the decoding time T_{dec}. Hence, one can write the total execution time of a coded computing algorithm as

$$T_{\text{total}} = \underbrace{T_{\text{enc}} + T_{\text{enc,comm}}}_{\text{Encoding}} + \underbrace{\min_{S\in\mathcal{S}} \max_{i\in S} T_i}_{\text{Task runtime}} + \underbrace{T_{\text{dec}}}_{\text{Decoding}}, \qquad (1)$$

where $\mathcal{S}$ is the set of all decodable sets of task indices [2]. We define $T_{\text{task}} = \min_{S\in\mathcal{S}} \max_{i\in S} T_i$.

Most existing works have focused on modeling T_i and reducing the task runtime, while assuming that the encoding/decoding times are negligible [2], [8], [9]. When the system size is relatively small, the encoding/decoding costs can be safely ignored. However, the encoding/decoding costs of coded computation schemes scale with the size of the system, and hence this assumption does not hold anymore for larger systems, e.g., systems with thousands or tens of thousands workers [3].

This forms the precise motivation of our work. Our goal is to design a coded computation scheme that not only achieves low task runtime but also allows for efficient encoding and decoding, hence a lower total execution time.

A. Problem Formulation

We consider the matrix-matrix multiplication task [4], where we compute $A^\top B$ for $A \in \mathbb{R}^{s\times\ell}, B \in \mathbb{R}^{s\times t}$. We focus on the case where s is large, and $s \geq \ell, t$, as otherwise

the resulting matrix is rank-deficient and there may be more efficient ways of computing the product. Further, while our results and algorithm can easily be generalized to the case where $\ell \neq t$, we will focus on the setting where $\ell = t$ for ease of presentation. To equalize the amount of computation, we divide A into m chunks, $A_i \in \mathbb{R}^{s\times \ell/m}$, and B into h chunks, $B_i \in \mathbb{R}^{s\times \ell/h}$, and have worker i perform a single matrix-matrix multiplication $\tilde{A}_i^\top \tilde{B}_i$, where $\tilde{A}_i$ and $\tilde{B}_i$ are input encoded matrices. The goal is to minimize T_{total} in (1) by delicately designing a coding mechanism.

B. Our Contributions

We propose a novel coded computation scheme based on d-dimensional product codes for matrix-matrix multiplication. We form this connection between matrix-matrix multiplication and decoding a high-dimensional product codes by adding carefully designed parity chunks on the input matrices. We show that our proposed scheme can achieve near-optimal task runtime performance as well as order-optimal encoding/decoding costs. This enables applicability to massive scale distributed computing where encoding/decoding costs make MDS based approaches impractical[1].

C. Related Works

In [4], the authors propose a 2D product code based scheme for straggler resilient matrix-matrix multiplication. They apply an $(n, k, d_{\min})$ code to each matrix, generating $n-k$ new chunks for each. Then, worker (i, j) is assigned the task of computing $A_i^\top B_j$. While this scheme has linear encoding and decoding costs, it exhibits suboptimal T_{task}. Another scheme proposed for matrix-matrix multiplication is Polynomial Codes [10], where a Reed-Solomon (RS) code is applied to the resulting matrix chunks. The authors show that T_{task} is optimal under the polynomial codes. However, in practical regimes, the encoding/decoding costs are much higher than those of product codes, and as the number of workers increase encoding/decoding costs can even become prohibitive.

II. d-DIMENSIONAL PRODUCT CODED COMPUTATION

In this section, we propose the d-dimensional product coded matrix multiplication algorithm, generalizing the 2D algorithm proposed in [4]. Under the metric of end-to-end latency, current MDS codes as in [10] exhibit undesirably long encoding and decoding times. On the other side of the spectrum, 2D product codes exhibit an error floor meaning that there are some theoretical limits to their performance, in that they require more processors than optimal. The d-dimensional product codes fall between these two, exhibiting optimal encoding and decoding times while only requiring a fractionally longer compute time.

[1] Indeed, the decoding cost of polynomial codes is prohibitive in large-scale systems. We argue that Reed-Solomon (RS) codes and other dense parity schemes will be less usable in the near future due to the increasing number of workers being used. All the off-the-shelf RS erasure coding libraries we found are capable of operating only up to $N = 256$, in $GF(2^8)$. This is because RS implementations require storing log and exp tables for each element of $GF(2^8)$ to operate quickly.

TABLE I: Glossary of important notation

Notation	Description
$(n, k, r+1)$	component code
d	dimension of product code
$N = n^d$	product code length
$K = k^d$	number of 'data' workers
$\ell \times s$	dimensions of input matrices A, B

Traditionally, product codes are thought of as 2D constructions, where one arranges the data in a $k\times k$ square, and applies a systematic linear (n, k) component code to every row and column, yielding an $n\times n$ square. We use the natural extension of this to higher dimensions, the d-dimensional product code as described more in depth in [11].

We describe the scenario of encoding a distributed matrix multiplication job via a d-dimensional product code. We proceed by chunking $A \in \mathbb{R}^{s\times \ell}$ into $\sqrt{K} = (n-r)^{d/2}$ chunks, of size $A_i \in \mathbb{R}^{s\times \ell/\sqrt{K}}$, and $B \in \mathbb{R}^{s\times \ell}$ into $\sqrt{K} = (n-r)^{d/2}$ chunks, of size $B_i \in \mathbb{R}^{s\times \ell/\sqrt{K}}$, to be computed over $N = n^d$ processors. In the rest of the section, as we describe the different phases of our algorithm, we are going to refer to Fig. 2 as a toy example.

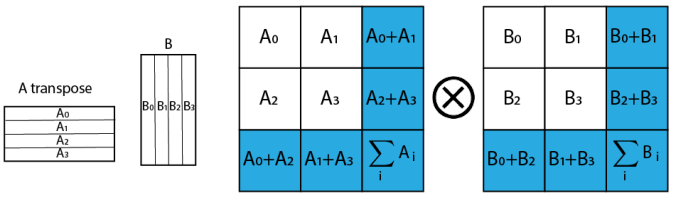

Fig. 2: Toy example for $n = 3$, $d = 4$, $r = 1$, which results in $N = n^d = 3^4$. This 4D product code is equivalent to the two 2D product codes tensored together. Parity chunks are colored in blue.

A. Encoding

Our encoding rests on the inherent tensored structure of a d-dimensional product code, as can be seen through the tensored structure of its encoding matrix. For a d-dimensional product code with an $(n, k, r+1)$ component code, the overall generator matrix is simply the tensor product[2] of d generator matrices for its $(n, k, r+1)$ component code (see Fig. 3). We utilize this inherent tensor structure by grouping the tensors into 2 groups of $d/2$ matrices, and spreading A, B over $d/2$ dimensions each.

$$G = \underbrace{\overbrace{\begin{bmatrix} 1 & \\ & 1 \\ 1 & 1 \end{bmatrix}}^{n-1} \otimes \ldots \otimes \overbrace{\begin{bmatrix} 1 & \\ & 1 \\ 1 & 1 \end{bmatrix}}^{n-1}}_{d}$$

Fig. 3: Generator matrix of d-dimensional product code with $(n, k, r+1) = (3, 2, 2)$

[2] We refer to a Kronecker product on the blocks of the matrices A, B.

We achieve this by computing parities for the matrices A, B independently after we have divided them into chunks. We do this by arranging the $(n-r)^{d/2}$ matrix chunks of each matrix in a $d/2$ dimensional hypercube with side length $k = n - r$. We then encode these matrix chunks using a d-dimensional product code with component code $(n, k, r+1)$, yielding a $d/2$-dimensional hypercube with side length n of encoded chunks of A. For example, in the toy example in Fig. 2 we have $d = 4$, so we divide each matrix along $4/2 = 2$ dimensions. Since each component code is $(n, k) = (3, 1)$, we have 2 data chunks and 1 parity chunk on each dimension. This completes our encoding; we now have all the coded matrix chunks.

B. Communication

As we can see from Fig. 2, there is a regular, easily exploitable structure to the matrix chunks processors require. Due to the tensor operation, we simply need to broadcast the $n^{d/2}$ coded chunks each of A, B to $n^{d/2}$ workers each. In our toy example, the worker indexed by (i, j, k, ℓ) where $i, j, k, l \in \{0, 1, 2\}$ needs to receive the (i, j)th chunk of A and (k, ℓ)th chunk of B. In particular, worker $(0, 0, 0, 0)$ receives A_0 and B_0, and worker $(0, 0, 0, 2)$ receives A_0 and $B_0 + B_1$.

C. Computation

We are able to induce a d-dimensional product code on the workers' results through a tensoring of coded matrix chunks of A, B. This is done by taking the $d/2$ dimensional hypercube of coded chunks of A and taking its tensor product with the $d/2$ dimensional hypercube of coded chunks of B. In our toy example, each worker (i, j, k, ℓ) where $i, j, k, l \in \{0, 1, 2\}$ multiplies the matrices it receives; worker $(0, 0, 0, 0)$ performs A_0B_0, and worker $(0, 0, 0, 2)$ performs $A_0(B_0 + B_1)$. This yields the desired d-dimensional product code structure.

D. Decoding

For decoding, we are able to employ a simple peeling decoder; due to the d-dimensional product code we have induced, we are able to simply arrange the workers' results in a d-dimensional hypercube with side length n, as suggested by the tensor structure (in Fig. 2 this would creating a 4D hypercube). Since each 'row' is a codeword for the $(n, k, r+1)$ component code, we can iterate over all dn^{d-1} rows and resolve the erasures (straggling workers) if there are fewer than r erasures in a given row. Resolving an erasure in one row may lead to another row becoming resolvable, and so if there are few enough erasures (formally characterized in Theorem 1), we will simply be able to iterate over the rows of the d-dimensional product code and resolve $(n, k, r+1)$ component codes until the results of all straggling processors have been recovered.

E. Choosing d

The choice of d makes a significant difference in performance and setting. In practice, one has a fixed number, N, of processors to use, and a targeted level of straggler resiliency, $N - K$. Since $K = (n-r)^d$, once we fix N and $N - K$, one can try and find an operating point characterized by d, r: for any $d \geq 2$, we have $r = n - \sqrt[d]{K}$ (c.f., Fig. 4). From here, one can choose the code parameter (d, r) that requires to wait for the fewest number of additional workers by looking up Table II which summarizes the performance of different choices of (d, r). An example that displays how the choice of code parameters effect performance is shown in Fig. 5.

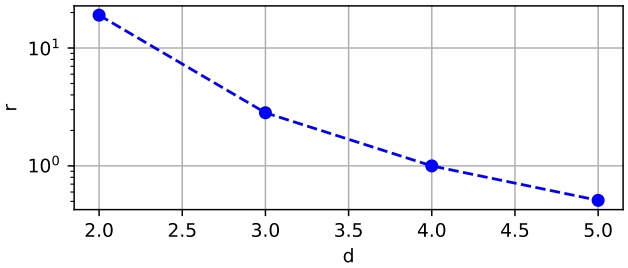

Fig. 4: Operating curve: (d, r) tuples for $N = 10^4$, $k = 9^4$

So far in our examples, we assumed d is even. For odd d, the encoding and communication costs for the two matrices are different, with one matrix being spread over $\lceil d/2 \rceil$ dimensions, and the other one over $\lfloor d/2 \rfloor$. For example, for $d = 3$, one matrix is cut into $(n-r)^2$ chunks and the other into $(n-r)$.

III. Analysis

A. Task Runtime

Our analysis utilizes the recently discovered isomorphism between d-dimensional product codes and a class of d-left regular LDPC codes presented [11]. A high level intuition for this isomorphism is that the randomness of the erasure channel (nature's random choice of straggling processors) allows us to leverage LDPC analysis techniques on the residual graphs. We can then derive the density evolution update equation, and substitute in d, r to numerically compute the maximum number of unknowns the residual graph can peel.

***Theorem** 1:* In d-dimensional product coded matrix multiplication scheme with $(n, k, r+1)$ component codes, a set of tasks S with $|S| \geq \left(N - \frac{N-K}{\eta(d,r)}\right)(1+\epsilon)$ drawn uniformly at random from the set of all N tasks is a *decodable set* w.h.p. for arbitrarily small $\epsilon > 0$ and $\eta(d, r)$ as given in Table II,

Proof: Through density evolution analysis, one can determine the erasure correcting capability of these d-dimensional product codes. This result directly follows from [11]. ■

For convenience, we define $K' = \left(N - \frac{N-K}{\eta(d,r)}\right)$, as the effective number of tasks a d-dimensional product code requires completed before it becomes decodable.

B. Encoding and Decoding costs

While achieving near-optimal task runtime as stated in Theorem 1, our scheme also achieves order-optimal encoding (both T_{enc} and $T_{\text{enc,comm}}$) and decoding complexities.

Lemma 1: T_{enc} for a d-dimensional product coded scheme is $O(\ell s)$, which is order-optimal.

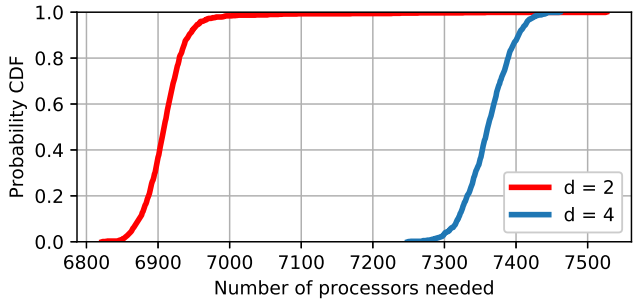

Fig. 5: Simulation results for $N = 10^4 = 100^2$, $k = 9^4 = 81^2$.

Proof: To encode our d-dimensional product code, we simply need to encode the matrix chunks of A, B in $d/2$-dimensional product codes, as explained in Sec. II-A. Since this is a $d/2$-dimensional product code itself, it can be encoded in time linear with respect to the number of overall chunks: $O(n^{d/2})$. Since matrix chunks are of size $\ell \times \frac{s}{(n-r)^{d/2}}$, our total work encoding A is $(n^{d/2}) \cdot \frac{\ell s}{(n-r)^{d/2}} = O(\ell s)$.

In addition, our encoding procedure is very simple and distributable itself. This means it is well suited for a masterless computational framework. Each worker will need to do $O(\frac{\ell^2 s}{K})$ work to multiply its two matrix chunks, and so the encoding cost of $O(\frac{\ell s}{n^{d/2-1}})$ is minimal as $\frac{\ell}{K} \gg \frac{1}{n^{d/2-1}}$, as $\ell \gg n^{d/2+1}$. The lower bound comes from the requirement that any scheme needs to read every element of A and B in order to prepare the chunks for communication. ■

Lemma 2: Communication cost, $T_{\text{enc,comm}}$, is $O(\ell s \gamma(\sqrt{N}))$, where $\gamma(m)$ is the multicast cost of communicating the same message to m workers. This cost is order-optimal.

Proof: Consider an arbitrary systematic coding scheme that divides A, B into m chunks each. It needs at least m^2 workers to return to finish computing, so $N \geq m^2$. We proceed by considering its communication cost for A. Assume that this scheme sends chunks of the same size to each worker, and that there are p, $m \leq p \leq N$, distinct chunks of A. Hence, the cost is $\frac{\ell s}{m} \times (\sum_{i=1}^{p} \gamma(c_i))$, where c_i is the number of times we send chunk i. Since the code is systematic, $c_i \geq m$ for $i = 1, ..., m$. Thus, the cost is lower bounded by $\frac{\ell s}{m} \times (m\gamma(m))$, as $\gamma(\cdot)$ is monotone non-decreasing, and positive. For $m^2 = \theta(N)$, $m = \theta(\sqrt{N})$ in our scheme, and so the cost can be lower bounded by $O(\ell s \times \gamma(\sqrt{N}))$.

Due to the tensored structure of our encoding scheme, we send the same chunk of A to $n^{d/2}$ workers. Our communication cost is thus $O(\frac{\ell s}{(n-r)^{d/2}}(n^{d/2} \cdot \gamma(n^{d/2})))$. For instance, if $\gamma(m) = \log(m)$, then our communication cost is $O(\frac{\ell s}{(n-r)^{d/2}}(n^{d/2} \cdot \log(n^{d/2}))) = O(\ell s \log(n))$. ■

TABLE II: Thresholds: $\eta(d, r)$

d \ r	1	2	3	4	5	6
3	1.2218	1.2880	1.3797	1.4564	1.5202	1.5741
4	1.2949	1.4998	1.6568	1.7781	1.8760	1.9575
5	1.4250	1.7275	1.9409	2.1031	2.2327	2.3406
6	1.5697	1.9577	2.2244	2.4256	2.5864	2.7199
7	1.7189	2.1869	2.5051	2.7446	2.9361	3.0953

Lemma 3: For a d-dimensional product code, decoding cost, T_{dec} is $O(\ell^2)$ which is order-optimal.

Proof: The decoding time of our scheme is linear in the size of the output matrix, ℓ^2. After K' workers have arrived the residual graph will be resolvable with high probability (c.f., Theorem 1). Because each straggling matrix chunk is of size $\frac{\ell^2}{K}$, and there are n chunks per row, $\frac{n}{(n-r)^d}\ell^2$ work is required to resolve a row. Since we have dn^{d-1} rows, our decoding time can be upper bounded by the time it takes to resolve them all, which gives $\frac{dn^d}{(n-r)^d}\ell^2 = O(\ell^2)$. ■

C. Comparisons

We first assume the existence of a *mother runtime distribution* $F(t)$ [2]: we assume that running an algorithm using a single machine takes a random amount of time T, that is a positive-valued, continuous random variable distributed according to F, i.e. $Pr(T \leq t) = F(t)$. Further, when the algorithm is distributed into K data subtasks, the runtime distribution for each subtask is assumed to be a scaled distribution of the mother distribution, i.e., $Pr(T_i \leq t) = F(Kt)$ for $1 \leq i \leq N$. We assume the computing times of these N tasks are independent of one another. Under these assumptions, $T_{\text{task}} = X_{(K')}$, the K' order statistic of N iid $X_i \sim f(x)$.

Empirically, a shifted exponential distribution which is the sum of a constant $c \geq 0$ and an exponential random variable with mean $\mu \geq 0$, i.e., $F(t) = 1 - e^{-\mu(t-c)}, \forall t \geq c$, has been found to be an accurate and tractable model for processor run time [2]. For $\{X_i\}_{i=1}^{N}$ distributed as iid shifted exponentials, $\mathbb{E}X_{(m)} = \frac{1}{K}(c + \frac{1}{\mu}\ln(\frac{N}{N-m}))$.

Specializing to the case of a d-dimensional product code, we have $\mathbb{E}T_{\text{task}} = \mathbb{E}X_{(K')} = \frac{1}{K}(c + \frac{1}{\mu}\ln\left(\frac{N}{(N-K)/\eta(d,r)}\right)) = \frac{1}{K}\left(c + \frac{1}{\mu}\ln\left(\frac{N}{N-K}\right) + \frac{1}{\mu}\ln\eta(d,r)\right)$

1) Reed-Solomon: We assume a systematic RS code is used. For the parities, encoding is done by creating a Vandermonde structure, which entails a weighted linear combination of the chunks of A, of the form $\sum_{i=1}^{\sqrt{K}} \alpha^i A_i$, for a properly chosen α. However, this work is unique for each parity, and involves using the entire matrix A. Thus, the total work encoding will be $\ell s(N - K)$.

For communication, a systematic RS code will be able to broadcast the $\sqrt{K}$ data chunks of each matrix to the data workers; each chunk is $\ell \times \frac{s}{\sqrt{K}}$, and needs to be used by $\sqrt{K}$ workers, yielding a cost of $\frac{\ell s}{\sqrt{K}}(\sqrt{K} \cdot \gamma(\sqrt{K})) = \ell s \cdot \gamma(\sqrt{K})$. For the parity workers, each one uses a uniquely computed chunk of A, meaning that multicast gains cannot be utilized. The cost for sending a parity chunks is $\frac{\ell s}{\sqrt{K}} \cdot (N - K)$, giving a total communication cost of $\ell s(\gamma(\sqrt{K}) + \frac{N-K}{\sqrt{K}})$.

For decoding, a systematic RS code will need to interpolate a polynomial for each one of the ℓ^2 entries in the resultant matrix. In each of these interpolations, one needs to solve a Vandermonde system of size $(N - K) \times (N - K)$, which is usually accepted to cost $O((N - K)^2)$ [12]. This results in a decoding cost of $\ell t(N - K)^2$.

TABLE III: Simplified table (comparisons are orderwise)

	D-dim Prod Code	Systematic RS	Uncoded
T_{enc}	$d\ell s$	$d\ell s n^{d-1}$	ℓs
T_{comm}	$\ell s\gamma(n^{d/2})$	$\ell s\gamma(k^{1/2})+$ $dn^{d/2-1}$	$\ell s\gamma(\sqrt{N})$
T_{task}	$\frac{1}{N}(c+\frac{1}{\mu}\log(n))$	$\frac{1}{N}(c+\frac{1}{\mu}\log(n))$	$\frac{1}{N}(c+\frac{1}{\mu}\log(N))$
T_{dec}	$d\ell^2$	$\ell^2 d^2 n^{2(d-1)}$	ℓ^2

2) Uncoded: Encoding is simply splitting the matrices into $\sqrt{N}$ chunks each, which requires a single pass over the data. Each processor receives $A_i, B_j \in \mathbb{R}^{s\times \ell/\sqrt{N}}$, and thus one broadcasts each A_i (similarly B_i) chunk to $\sqrt{N}$ processors. Thus the overall communication cost is $O(\ell s\gamma(\sqrt{N}))$. The computation latency is $\mathbb{E}X_{(N)}$, and so the expected latency under a shifted exponential model is $O(c+\mu\ln(N))$.

IV. Simulation Results

One way of comparing different schemes is by comparing their synthetic wall clock times; for a given matrix size, and N, K, we can calculate the number of flops required for the encoding and decoding procedures. We can then compute the expected worker latency for a given model (currently shifted exponential), and then express the wall clock time as $\alpha\cdot(T_{\text{enc}}+T_{\text{dec}})+T_{\text{task}}$. We sweep α over a wide range of values. We can see from Fig. 6 that when costs negligible (very low α), RS codes perform better. Above that, $d=4$ product codes perform better due to its superior threshold to $d=2$, but once encoding/decoding costs become dominant, $d=2$ performs better (due its marginally lower encoding/decoding costs). As encoding and decoding costs become more dominant, stragglers become less of an issue, and the uncoded scheme yields the fastest runtime.

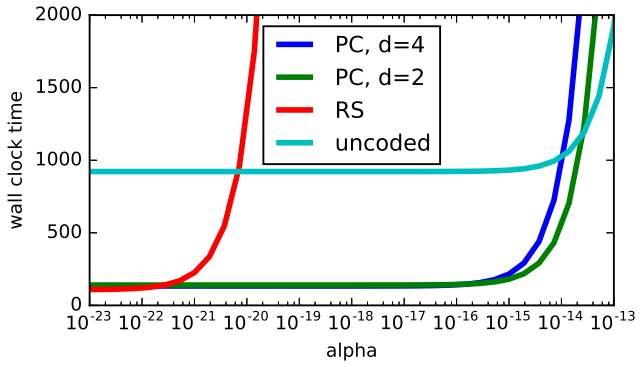

Fig. 6: Synthetic wall clock times for $N=10^4, k=9^4, r=s=10^8$. Time = $\alpha\cdot(T_{\text{enc}}+T_{\text{dec}})+\mathbb{E}T_{\text{task}}$

An alternate expression for wall clock time is $\beta T_{\text{comm}}+T_{\text{task}}$. We plot this in Fig. 7. We can see a similar trend as before. From these numerical comparisons, it is clear that the minimum task runtime does not always imply the minimum execution time. One should consider not only the task runtime, but also the encoding/decoding complexities as well as associated communication costs.

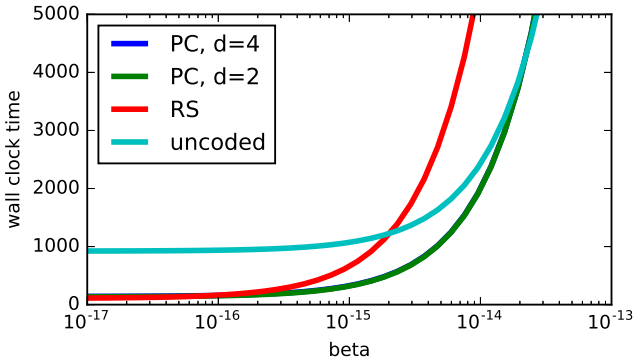

Fig. 7: Synthetic wall clock times for $N=10^4, k=9^4, r=s=10^8$. Time = $\beta\cdot T_{\text{enc,comm}}+\mathbb{E}T_{\text{task}}$

V. Conclusion and Future Work

We presented a novel coded matrix-matrix multiplication scheme that has order-optimal computation/communication costs for the encoding/decoding procedures and achieves near-optimal task runtime. Our scheme is based on d-dimensional product codes, and allows for efficient encoding/decoding algorithms and low communication cost. Because product codes can be decoded by operating on single 'rows' and 'columns' in an iterative way, our algorithm is a suitable candidate to have an impact for master-less computation framework. In particular, in a platform having shared memory with fast input/output where workers write their results, separate workers assigned to the task of decoding can resolve 'rows' and 'columns' in streamlined fashion. Implementation of such a practical master-less algorithm is an ongoing work.

References

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013.

[2] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Int. Symp. on Inf. Theory*, vol. 2016-Augus, pp. 1143–1147, 2016.

[3] E. Jonas, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," *CoRR*, vol. abs/1702.04024, 2017.

[4] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," *IEEE Int. Symp. on Inf. Theory*, pp. 2418–2422, 2017.

[5] S. Dutta, V. R. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," *CoRR*, vol. abs/1705.03875, 2017.

[6] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded fourier transform," *arXiv preprint arXiv:1710.06471*, 2017.

[7] Y. Yang, P. Grover, and S. Kar, "Coding method for parallel iterative linear solver," *CoRR*, vol. abs/1706.00163, 2017.

[8] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *ICML*, 2017, pp. 3368–3376.

[9] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *IEEE Int. Symp. on Inf. Theory*, June 2017, pp. 2413–2417.

[10] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," 2017.

[11] T. Baharav and K. Ramchandran. (2018, Jan.) Leaving randomness to nature: d-dimensional product codes through the lens of Generalized-LDPC codes. [Online]. Available: https://people.eecs.berkeley.edu/~kannanr/assets/high-dim-prod-codes.pdf

[12] F. Didier, "Efficient erasure decoding of reed-solomon codes," *arXiv preprint arXiv:0901.1886*, 2009.