DISTRIBUTED GRADIENT DESCENT WITH
CODED PARTIAL GRADIENT COMPUTATIONS

E. Ozfatura†, S. Ulukus+ and D. Gündüz†

† Department of Electrical and Electronic Engineering, Imperial College London, UK
+ Department of Electrical and Computer Engineering, University of Maryland, MD

ABSTRACT

Coded computation techniques provide robustness against *straggling* servers in distributed computing, with the following limitations: First, they increase decoding complexity. Second, they ignore computations carried out by straggling servers; and they are typically designed to recover the full gradient, and thus, cannot provide a balance between the accuracy of the gradient and per-iteration completion time. Here we introduce a hybrid approach, called *coded partial gradient computation (CPGC)*, that benefits from the advantages of both coded and uncoded computation schemes, and reduces both the computation time and decoding complexity.

Index Terms— Gradient descent, coded computation, maximum distance separable (MDS) codes, LT codes.

1. INTRODUCTION

In many machine learning applications, the principal computational task boils down to a matrix-vector multiplication. Consider, for example, the minimization of the empirical mean squared error in linear regression $L(\boldsymbol{\theta}) \triangleq \frac{1}{2N} \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2$, where $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^L$ are the data points with corresponding labels $y_1, \ldots, y_N \in \mathbb{R}$, and $\boldsymbol{\theta} \in \mathbb{R}^L$ is the parameter vector. The optimal parameter vector can be obtained iteratively by gradient descent (GD): $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$, where η_t is the learning rate and $\boldsymbol{\theta}_t$ is the parameter vector at the tth iteration. We have $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) = \mathbf{X}^T \mathbf{X} \boldsymbol{\theta}_t - \mathbf{X}^T \mathbf{y}$, where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$ and $\mathbf{y} = [y_1, \ldots, y_N]^T$. In the gradient expression, only $\boldsymbol{\theta}_t$ changes over iterations; hence, the key computational task at each iteration is the matrix-vector multiplication $\mathbf{W} \boldsymbol{\theta}_t$, where $\mathbf{W} \triangleq \mathbf{X}^T \mathbf{X} \in \mathbb{R}^{L \times L}$. To speed up GD, execution of this multiplication can be distributed across K *workers*, by simply dividing $\mathbf{W}$ into K equal-size disjoint submatrices. However, the computation time will now be limited by the *straggling* workers [1].

Coded computation has been introduced to tolerate stragglers by introducing redundant computations [2–12]. In [2], $\mathbf{W} \in \mathbb{R}^{L \times L}$ is divided into M disjoint submatrices,

$\mathbf{W}_1, \ldots, \mathbf{W}_M \in \mathbb{R}^{L/M \times L}$, which are then encoded with an (M, K) maximum distance separable (MDS) code, and each coded submatrix is assigned to a different worker. Each worker multiplies $\boldsymbol{\theta}_t$ with its assigned coded submatrix, and sends the result to the master, which can recover $\mathbf{W} \boldsymbol{\theta}_t$ from the results of any M workers. Up to $K - M$ stragglers can be tolerated at the expense of increasing the *computation load* of each worker by $r = K/M$ [2]. Alternatively, uncoded computations can be executed, and the results can be sent as a coded messages [13–15]. However, these approaches discard computations carried out by straggling servers, and hence, the overall computational capacity is underutilized.

Alternatively, workers can be allowed to send multiple messages per-iteration, corresponding to partial computations [3, 6, 9, 16]. In [3] *multi-message communication (MMC)* is applied to MDS-coded computation utilizing the statistics of stragglers. Instead, rateless codes are proposed in [9] as they do not require the knowledge of the straggler statistics, and also reduce the decoding complexity. However, rateless codes come with an overhead, which vanishes only if the number of codewords goes to infinity. This, in turn, would increase the number of read/write operations at the master at each iteration, limiting its benefits in practical applications.

Uncoded distributed computation with MMC (UC-MMC) is introduced in [1, 6, 16, 17], and is shown to outperform coded computation in terms of average completion time, concluding that coded computation can be effective against persistent stragglers, and particularly when full gradient is required. Coded GD strategies are mainly designed for full gradient computation; hence, the master needs to wait until all the gradients can be recovered. UC-MMC, on the other hand, in addition to exploiting partial computations performed by straggling servers, also allows the master to update the parameter vector with only a subset of the gradient computations further reducing the per-iteration completion time.

Here, we introduce a novel hybrid scheme, called *coded partial gradient computation (CPGC)*, bringing together the advantages of uncoded computation, such as low decoding complexity and partial gradient updates, with those of coded computation, such as reduced per-iteration completion time and limited communication load. Next, we will motivate CPGC on a simple example.

Cumulative computation type	MCC $n_m(i)$	UC-MMC $n_u(i)$	CPGC $n_c(i)$
$\mathbf{N}_1 : N_2 = 4, N_1 = 0, N_0 = 0$	1	1	1
$\mathbf{N}_2 : N_2 = 3, N_1 = 1, N_0 = 0$	4	4	4
$\mathbf{N}_3 : N_2 = 3, N_1 = 0, N_0 = 1$	4	4	4
$\mathbf{N}_4 : N_2 = 2, N_1 = 2, N_0 = 0$	6	6	6
$\mathbf{N}_5 : N_2 = 2, N_1 = 1, N_0 = 1$	12	8	12
$\mathbf{N}_6 : N_2 = 2, N_1 = 0, N_0 = 2$	6	2	6
$\mathbf{N}_7 : N_2 = 1, N_1 = 3, N_0 = 0$	0	4	4
$\mathbf{N}_8 : N_2 = 1, N_1 = 2, N_0 = 1$	0	4	8
$\mathbf{N}_9 : N_2 = 0, N_1 = 4, N_0 = 1$	0	1	1

Table 1: Number of score vectors for full gradient computation with $K = 4$ workers with computation load $r = 2$.

2. MOTIVATING EXAMPLE

Consider $M = 4$ computation tasks, represented by submatrices $\mathbf{W}_1, \ldots, \mathbf{W}_4$, which are to be executed across $K = 4$ workers, each with a maximum computation load of $r = 2$; that is, each worker can perform up to 2 computations, due to storage or computation capacity limitations. Let us first consider two known distributed computation schemes, namely UC-MMC [6, 16] and MDS-coded computation (MCC) [2].

For each scheme, the $r \times K$ *computation scheduling matrix*, $\mathbf{A}$, shows the assigned computation tasks to each worker with their execution order. More specifically, $\mathbf{A}(i, j)$ denotes the ith computation task to be executed by the jth worker. In MCC, linearly independent coded computation tasks are distributed to the workers as follows:

$$\mathbf{A}_m = \begin{bmatrix} \mathbf{W}_1 + \mathbf{W}_3 & \mathbf{W}_1 + 2\mathbf{W}_3 & \mathbf{W}_1 + 4\mathbf{W}_3 & \mathbf{W}_1 + 8\mathbf{W}_3 \\ \mathbf{W}_2 + \mathbf{W}_4 & \mathbf{W}_2 + 2\mathbf{W}_4 & \mathbf{W}_2 + 4\mathbf{W}_4 & \mathbf{W}_2 + 8\mathbf{W}_4 \end{bmatrix}.$$

Each worker sends the results of its computations only after all of them are completed, e.g., first worker sends the concatenation $[(\mathbf{W}_1 + \mathbf{W}_3)\boldsymbol{\theta}_t \quad (\mathbf{W}_2 + \mathbf{W}_4)\boldsymbol{\theta}_t]$ after completing both computations; therefore, any permutations of each column vector would result in the same performance. $\mathbf{A}_m$ corresponds to a $(2, 4)$ MDS code; hence, the master can recover the full gradient from the results of any two workers.

In the UC-MMC scheme with a shifted computation schedule [6], computation scheduling matrix is given by

$$\mathbf{A}_u = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 & \mathbf{W}_1 \end{bmatrix},$$

and each worker sends the results of its computations sequentially, as soon as each of them is completed. This helps to reduce the per-iteration completion time with an increase in the communication load [6, 16]. With UC-MMC, full gradient can be recovered even if each worker performs only one computation, which is faster if the workers have similar speeds.

The computation scheduling matrix of CPGC is given by

$$\mathbf{A}_c = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 + \mathbf{W}_4 & \mathbf{W}_1 + \mathbf{W}_3 & \mathbf{W}_2 + \mathbf{W}_4 & \mathbf{W}_1 + \mathbf{W}_2 \end{bmatrix}.$$

Cumulative computation type	MCC $n_m(i)$	UC-MMC $n_u(i)$	CPGC $n_c(i)$
$\mathbf{N}_1 : N_2 = 4, N_1 = 0, N_0 = 0$	1	1	1
$\mathbf{N}_2 : N_2 = 3, N_1 = 1, N_0 = 0$	4	4	4
$\mathbf{N}_3 : N_2 = 3, N_1 = 0, N_0 = 1$	4	4	4
$\mathbf{N}_4 : N_2 = 2, N_1 = 2, N_0 = 0$	6	6	6
$\mathbf{N}_5 : N_2 = 2, N_1 = 1, N_0 = 1$	12	12	12
$\mathbf{N}_6 : N_2 = 2, N_1 = 0, N_0 = 2$	6	6	6
$\mathbf{N}_7 : N_2 = 1, N_1 = 3, N_0 = 0$	0	4	4
$\mathbf{N}_8 : N_2 = 1, N_1 = 2, N_0 = 1$	0	12	12
$\mathbf{N}_9 : N_2 = 1, N_1 = 1, N_0 = 2$	0	8	8
$\mathbf{N}_{10} : N_2 = 0, N_1 = 4, N_0 = 0$	0	1	1
$\mathbf{N}_{11} : N_2 = 0, N_1 = 3, N_0 = 1$	0	4	4

Table 2: Number of score vectors for partial gradient computation (3 out of 4) with $K = 4$ and $r = 2$.

2.1. Full Gradient Performance

Now, let us focus on a particular iteration, and let N_s denote the number of workers that have completed exactly s computations by time t, $s = 0, \ldots, r$. We define $\mathbf{N} \triangleq (N_r, \ldots, N_0)$ as the *cumulative computation type*. Additionally, we introduce the K-dimensional *score vector* $\mathbf{C} = [c_1, \ldots, c_K]$, where c_i denotes the number of computations completed by the ith worker. Let $n_m(i), n_u(i)$ and $n_c(i)$ denote the number of distinct score vectors with the cumulative computation type N_i, which allow the recovery of full gradient, for the MCC, UC-MMC, and CPGC schemes, respectively. These values are listed in Table 1. Particularly striking are the last three rows that correspond to cases with very few computations completed, i.e., when at most one worker completes all its assigned tasks. In these cases, CPGC is much more likely to allow full gradient computation; and hence, the computation deadline can be reduced significantly while still recovering the full gradient.

Next, we analyze the probability of each type under a specific computation time statistics. We adopt the model in [2], where the probability of completing exactly s computations by time t, $P_s(t)$, is given by

$$P_s(t) = \begin{cases} 0, & \text{if } t < s\alpha, \\ 1 - e^{-\mu(\frac{t}{s} - \alpha)}, & s\alpha \le t < (s+1)\alpha, \\ e^{-\mu(\frac{t}{s+1} - \alpha)} - e^{-\mu(\frac{t}{s} - \alpha)} & (s+1)\alpha < t, \end{cases} \quad (1)$$

where α is the minimum required time to finish a computation task, and μ is the average number of computations completed in unit time. The probability of cumulative computation type $\mathbf{N}(t)$ at time t is given by $\Pr(\mathbf{N}(t)) = \prod_{s=0}^{r} P_s(t)^{N_s}$. Let T denote the full gradient recovery time. Accordingly, for the example above, $\Pr(T < t)$ for CPGC is given by $\sum_{i=1}^{9} n_c(i) \cdot \Pr(\mathbf{N}_i(t))$, where the types $\mathbf{N}_i$ and corresponding $n_c(i)$, $i = 1, \ldots, 9$, are listed in Table 1. $\Pr(T < t)$ for MCC and UC-MMC can be written similarly by replacing $n_c(i)$ with $n_m(i)$ and $n_u(i)$, respectively. It is now clear that CPGC has the highest $\Pr(T < t)$ for any t; and hence, the minimum average per-iteration completion time $E[T]$. In the next subsection, we will highlight the partial recoverability property of CPGC.

2.2. Partial Gradient Performance

GD methods can be effective even with less accurate model updates. For instance, stochastic GD can still guarantee convergence even if each iteration is completed with only a subset of the gradient estimations [18,19]. Similarly, in our example, sufficient accuracy may be achieved by updating the parameter vector using three out of four partial computations at each iteration, assuming that the straggling server varies over iterations. The number of score vectors for which a partial gradient (with at least three gradient computations) can be recovered are given in Table 2. Note that when three gradient computations are sufficient to complete an iteration UC-MMC and CPGC have the same average completion time statistics. Hence, CPGC can provide a lower average per-iteration completion time for full gradient computation compared to UC-MMC, while achieving the same performance when partial gradients are allowed.

3. DESIGN PRINCIPLES OF CPGC

In [9], LT codes are proposed for distributed computation in order to exploit MMC with coded computations. However, LT codes come with a trade-off between the overhead and the associated coding/decoding complexity. Moreover, the original design in [9] does not allow partial gradient recovery.

The key design issue in LT codes is the degree distribution $P(d)$. Degree of a codeword, d, chosen randomly from $P(d)$, defines the number of symbols ($\mathbf{W}_i$ submatrices in our setting) that are used in generating a codeword. Then, d symbols are chosen randomly to form a codeword. Degree distribution plays an important role in the performance of an LT code, and the main challenge is to find the optimal degree distribution. Codewords with smaller degrees reduce decoding complexity; however, having many codewords with smaller degrees increases the probability of linear dependence among codewords. We also note that, LT code design is based on the assumption that the erasure probability of different codewords are identical and independent from each other. However, in a coded computing scenario, the computational tasks, each of which corresponds to a distinct codeword, are executed sequentially; thus, erasure probabilities of codewords are neither identical nor independent. Codewords must be designed taking into account their execution orders to prevent overlaps and to minimize the average completion time. This is the main intuition behind the CPGC scheme, and guides the design of the computation scheduling matrix.

3.1. Degree Limitation

To allow partial gradient computation at the master, we limit the degree of all codewords by two; that is, each codeword (i.e., coded submatrix) is the sum of at most two submatrices. Moreover, the first computation task assigned to each worker corresponds to a codeword with degree one (i.e., a $\mathbf{W}_i$ submatrix is assigned to each worker without any coding), while all other tasks correspond to codewords with degree two (coded submatrices). Recall that, due to the straggling behavior, the first task at each worker has the highest completion probability, thus assigning uncoded submatrices as the first computation task at each worker helps to enable partial recovery.

3.2. Coded Data Generation

In an LT code, symbols (submatrices) that are linearly combined to generate a codeword are chosen randomly; however, to enable partial gradient recovery, we carefully design the codewords for each worker.

For a given set of submatrices $\mathcal{W}$, a partition $\mathcal{P}$ is a grouping of its elements into nonempty disjoint subsets. In our example, we have $\mathcal{W} = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4\}$, and $\mathcal{P} = \{\{\mathbf{W}_1, \mathbf{W}_2\}, \{\mathbf{W}_3, \mathbf{W}_4\}\}$ is a partition. Now, consider the following scheme: for each $Q \in \mathcal{P}$, a codeword $c(Q)$ is generated by $\sum_{\mathbf{W}' \in Q} \mathbf{W}'$. Since for any $Q_i, Q_j \in \mathcal{P}$, $i \neq j$, $Q_i \cap Q_j = \emptyset$, codewords $c(Q_i)$ and $c(Q_j)$ share no common submatrix. Accordingly, one can easily observe that if n partitions are used to generate coded submatrices, each submatrix $\mathbf{W}_i$ appears in exactly n different coded submatrices. In order to generate degree-two codewords, we use partitions with subsets of size two; hence, exactly $K/2$ coded submatrices are generated from a single partition. Therefore, for each row of the computation scheduling matrix we need exactly two partitions of $\mathcal{W}$, and in total we require $2(r-1)$ distinct partitions (see [20] for details).

Note that the probability of not receiving the results of computations corresponding to coded submatrices in the same column of the computation scheduling matrix are correlated, as they are executed by the same worker. Hence, in order to minimize the dependence on a single worker, we would like to limit the appearance of a submatrix in any single column of the computation scheduling matrix. In the next section, we provide a heuristic strategy for coded submatrix assignment.

4. NUMERICAL RESULTS AND CONCLUSIONS

We will analyze and compare the performance of three schemes, UC-MMC, MCC, and CPGC, in terms of three performance measures, the *average per-iteration completion time*, *communication load* and the *communication volume*. The communication load, defined in [6, 16], refers to the average number of messages transmitted to the master from the workers per iteration, whereas the communication volume refers to the average total size of the computations sent to the master per iteration. This is normalized with respect to the result of $\mathbf{W}\theta$, which is set as the unit data volume. This is to distinguish between the partial and full computation results sent from the workers in CPGC and MCC schemes, respectively. In CPGC we transmit many messages of smaller

(a) Average per-iteration completion time. (b) Communication volume. (c) Communication load.
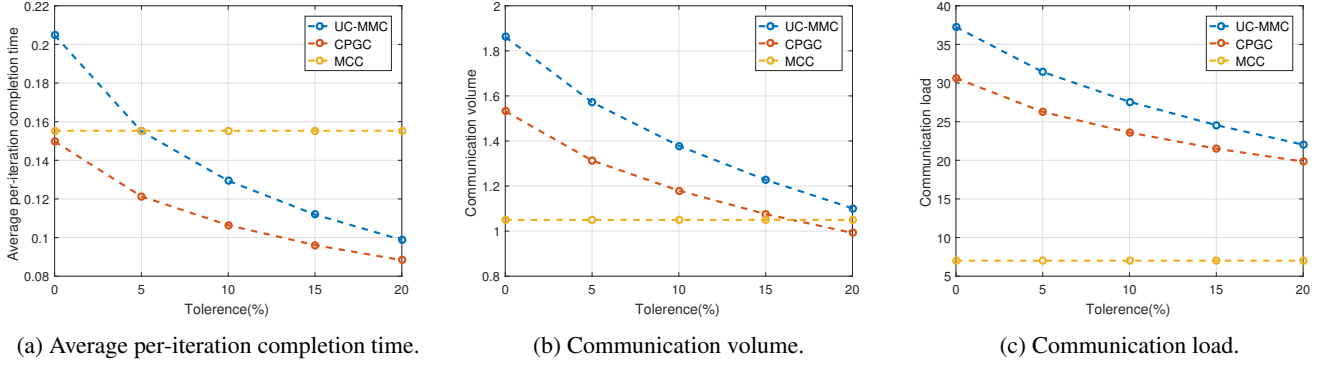
Fig. 1: Performance comparison between UC-MCC, CPGC and MCC schemes for $M = K = 20$ and $r = 3$.

size, while MCC sends a single message consisting of multiple results. Communication volume allows us to compare the amount of redundant computations sent from the workers to the master. A communication volume of 1 implies zero communication overhead, whereas a communication volume larger than 1 implies communication overhead due to transmission of multiple messages.

4.1. Simulation Setup

We consider $K = 20$ workers and $M = 20$ computation tasks (submatrices), and a computation load of $r = 3$. We set $\mu = 10$ and $\alpha = 0.01$ for the statistics of computation speed in (1).

In CPGC, first computations assigned are uncoded submatrices. For the second and third rows of the computation scheduling matrix we use four different partitions with coded submatrices as follows (assuming N is even):

$$\mathbf{v}_1 = [\mathbf{W}_1 + \mathbf{W}_2, \ldots, \mathbf{W}_n + \mathbf{W}_{n+1}, \ldots, \mathbf{W}_{N-1} + \mathbf{W}_N]$$
$$\mathbf{v}_2 = [\mathbf{W}_1 + \mathbf{W}_3, \ldots, \mathbf{W}_n + \mathbf{W}_{n+2}, \ldots, \mathbf{W}_{N-2} + \mathbf{W}_N]$$
$$\mathbf{v}_3 = [\mathbf{W}_1 + \mathbf{W}_N, \ldots, \mathbf{W}_n + \mathbf{W}_{N-n+1}, \ldots, \mathbf{W}_{N/2} + \mathbf{W}_{N/2+1}]$$
$$\mathbf{v}_4 = [\mathbf{W}_1 + \mathbf{W}_{N/2+1}, \ldots, \mathbf{W}_n + \mathbf{W}_{N/2+n}, \ldots, \mathbf{W}_{N/2} + \mathbf{W}_N]$$

These coded submatrices are used to form a computation scheduling matrix in the following way: $\mathbf{A}(2, 1 : K/2) = \text{circshift}(\mathbf{v}_1; -1)$, $\mathbf{A}(2, K/2 + 1 : K) = \text{circshift}(\mathbf{v}_2; -1)$, $\mathbf{A}(3, 1 : K/2) = \text{circshift}(\mathbf{v}_3; 1)$, $\mathbf{A}(3, K/2 + 1 : K) = \text{circshift}(\mathbf{v}_4; -2)$, where circshift is the circular shift operator, i.e., $\text{circshift}(\mathbf{v}; d)$ is the d times right-shifted version of vector $\mathbf{v}$. We use the shifted version of the vectors to prevent multiple appearance of a submatrix in a single column.

4.2. Results

For M submatrices, let M' be the required number of computations, each corresponding to a different submatrix, to terminate an iteration. We define $\frac{M-M'}{M}$ as the *tolerance ratio*, which reflects the gradient accuracy at each iteration (lower tolerance ratio means higher accuracy).

In Fig. 1, we compare the three schemes under the three performance metrics with respect to the tolerance ratio. Since

partial recovery is not possible with MDS-coded computation, its performance remains the same with the tolerance ratio. The performance of UC-MMC and CPGC improve with the tolerance ratio. This comes at the expense of a slight reduction in the accuracy of the resultant gradient computation. We remark that, beyond a certain tolerance ratio UC-MMC scheme achieves a lower average per iteration completion time than MCC thanks to the utilization of non-persistent stragglers [6, 16]. Also, CPGC outperforms both UC-MMC and MCC thanks to coded inputs. It also allows partial gradient computation, and provides approximately 25% reduction in the average per iteration completion time compared to MCC and UC-MMC at a 5% tolerance ratio.

Communication volume of the UC-MMC scheme for with full gradient computation is around 1.8, which means that there is 80% communication overhead. Similarly, the communication volume of CPGC is around 1.5. MCC has the minimum communication volume since the MDS code has zero decoding overhead[1]. We also observe that the communication volume of CPGC decreases with the tolerance ratio, and it is close to that of MCC at a tolerance ratio of around 10%.

We recall that the design goal of CPGC is to provide flexibility in seeking a balance between the per-iteration completion time and accuracy. To this end, different iteration termination strategies can be introduced to reduce the overall convergence time. We show in [20] that a faster overall convergence can be achieved with CPGC by increasing the tolerance ratio at each iteration, as this would reduce the per-iteration completion time. Finally, one can observe from Fig. (1b) and (1c) that the MMC affects the communication load more drastically compared to the communication volume. This may introduce additional delays depending on the computing infrastructure and the communication protocol employed, e.g., dedicated links from the workers to the master compared to a shared communication network as in wireless edge learning [21].

---

[1]Communication volume of the MCC is slightly greater than 1 since K is not divisible by r, and zero padding is used before encoding.

5. REFERENCES

[1] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz, "Revisiting distributed synchronous SGD," *CoRR*, vol. abs/1604.00981, 2016.

[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, March 2018.

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

[4] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with LDPC codes," *SysML Conference*, February 2018.

[5] Songze Li, Seyed Mohammadreza Mousavi Kalan, Qian Yu, Mahdi Soltanolkotabi, and Amir Salman Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *CoRR*, vol. abs/1805.09934, 2018.

[6] Emre Ozfatura, Deniz Gündüz, and Sennur Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," *CoRR*, vol. abs/1808.02240, 2018.

[7] Sanghamitra Dutta, Mohammad Fahim, Farzin Haddadpour, Haewon Jeong, Viveck R. Cadambe, and Pulkit Grover, "On the optimal recovery threshold of coded matrix multiplication," *CoRR*, vol. abs/1801.10292, 2018.

[8] Can Karakus, Yifan Sun, Suhas Diggavi, and Wotao Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems 30*, pp. 5434–5442. 2017.

[9] A. Mallick, M. Chaudhari, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," 2018, available at arXiv:1804.10331.

[10] H. Park, K. Lee, J. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1630–1634.

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

[12] A. B. Das, L. Tang, and A. Ramamoorthy, "C^3LES: Codes for coded computation that leverage stragglers," in *2018 IEEE Information Theory Workshop (ITW)*, Nov 2018, pp. 1–5.

[13] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, International Convention Centre, Sydney, Australia, 06–11 Aug 2017, vol. 70 of *Proceedings of Machine Learning Research*, pp. 3368–3376, PMLR.

[14] Min Ye and Emmanuel Abbe, "Communication-computation efficient gradient coding," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholmsmassan, Stockholm Sweden, 10–15 Jul 2018, vol. 80 of *Proceedings of Machine Learning Research*, pp. 5610–5619, PMLR.

[15] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 2027–2031.

[16] Mohammad Mohammadi Amiri and Deniz Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *CoRR*, vol. abs/1810.09992, 2018.

[17] S. Li, S. M. Mousavi Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018, pp. 857–866.

[18] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in *21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, April 2018.

[19] L. Bottou, F. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[20] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," `https://1drv.ms/f/s!Ag0zbhMUMbtqsQIhJhBQ6pL0QtUA`, 2018.

[21] Mohammad Mohammadi Amiri and Deniz Gündüz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," *CoRR*, vol. abs/1901.00844, 2019.