COMMUNICATION-EFFICIENT DISTRIBUTED TRAINING VIA RING-BASED CODED APPROXIMATE ALL-REDUCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Ring All-Reduce is a widely used protocol within large-scale distributed training for aggregating gradients across workers in each iteration. For a system with N workers, its normalized per-worker communication is $2(N-1)/N$. In this work we present a communication-efficient Ring All-Reduce (CERAR) protocol for "approximate" gradient aggregation. CERAR partitions each local gradient into c components and crucially relies on linear encoding and decoding operations. It performs $L = c + N - 2$ communication rounds over an N-worker ring, yielding normalized communication rate $1 + (N-2)/c$ and normalized storage rate $1 + N/c$. We present an explicit Vandermonde matrix based construction, whose approximation error can be made arbitrarily close to zero with communication and storage rates approaching one with increasing c. However, this limit is achieved through ill-conditioned encoding matrices. Accordingly, we give an alternate construction whose error is $O(\epsilon)$, while the relevant condition numbers are $O(\epsilon^{-r_\star})$, where $r_\star = \lceil c/N \rceil - 1$. This naturally motivates a condition-number-constrained optimization formulation for trading off the competing objectives and obtaining numerically stable practical designs.

Our numerical experiments demonstrate that when compared with the production-standard NCCL on NVIDIA A100 PCIe GPU Clusters with low-bandwidth PCIe interconnect, CERAR reduces the raw All-Reduce phase time by up to 47%, and for training a 1.4-billion-parameter Pythia model it reduces training time by about 17% for the same number of iterations while achieving essentially identical validation loss. On the other hand, for GPU clusters with high-bandwidth NVLink interconnects, CERAR performs worse on the All-Reduce phase time and slightly worse in terms of training. Nevertheless, the gap reduces with experiments that involve systems with larger parameter counts. In this scenario we expect the gains to manifest with more optimized implementations of the CERAR protocol.

1 INTRODUCTION

A typical scenario within model training involves a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\tilde{N}}$ of $\tilde{N}$ data points, where $\mathbf{x}_i$'s and y_i's are the features and labels, respectively. The aim is to minimize a loss function $L \triangleq \frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} l(\mathbf{x}_i, y_i; \mathbf{w})$ with respect to $\mathbf{w}$, where $\mathbf{w} \in \mathbb{R}^d$ is the unknown parameter that needs to be learned. The function l measures the prediction error for each data point. In the distributed setting, portions of the dataset are distributed to different workers who compute the gradient on the data points assigned to them. At the t-th iteration, the system aims at computing the overall gradient $\nabla L = \frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} \nabla l(\mathbf{x}_i, y_i; \mathbf{w}_t)$, following which a parameter-update is performed and the iterations continue thereafter. We emphasize that $\mathcal{D}$ can also represent a relevant subset of the entire dataset, e.g., when the cluster wants to run mini-batch stochastic gradient descent Bottou et al. (2018) on an appropriate batch; this is in fact the most common mode of usage. Several distributed learning architectures have been investigated in the literature Langer et al. (2020), that differ based on how the dataset is divided into chunks and how the workers are connected to each other.

Background and Motivation: For N workers, the basic Ring All-Reduce training algorithm Patarasuk & Yuan (2009); Sergeev & Del Balso (2018) operates by partitioning $\mathcal{D}$ into N equal-sized dis-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

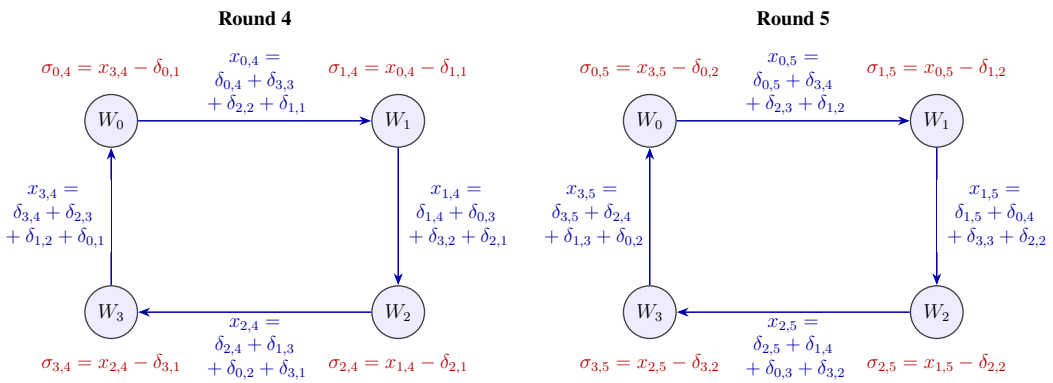


Figure 1: CERAR protocol with $N = 4$, $c = 5$, and $L = 7$: rounds 4 and 5. Arrows show transmitted vectors; equations beside workers show the rolling-state updates after reception in the indicated round. The update equations appear in (2).

joint chunks $\mathcal{D}_j, j = 0, \ldots, N-1$, such that worker W_j is assigned $\mathcal{D}_j$. W_j computes the sum of the gradients on its assigned data points, i.e., it computes $g_{\mathcal{D}_j} = \frac{1}{N} \sum_{i \in \mathcal{D}_j} \nabla l(\mathbf{x}_i, y_i; \mathbf{w}_t)$ (a vector of length-d). Once the workers have computed their assigned gradients, they send messages in parallel around a clockwise ring such that each worker can compute the overall gradient ∇L. For a ring with N workers, each worker transmits and receives a total of $2(N-1)d/N$ real values. For large N, the load normalized by d approaches 2, i.e., it does not grow with the number of workers. In contrast, in older architectures such as parameter server (PS) + workers Li et al. (2014), the bandwidth load on the PS grows with N.

The basic Ring All-Reduce algorithm is nominally considered to be optimal from the point of view of recovering the sum $g_{\mathcal{D}_0} + \cdots + g_{\mathcal{D}_{N-1}}$ exactly Patarasuk & Yuan (2009). From an information-theoretic perspective, one can consider the problem of recovering the sum with arbitrarily small (but nonzero) error; the lower bound of Patarasuk & Yuan (2009) does not apply in this case. We note that approximate recovery often suffices in scenarios such as distributed parameter training, i.e., instead of true gradients, we often settle for approximate gradients which do not affect the quality of training much. We consider this approximate recovery setting in this paper and investigate the fundamental question of how small the communication load on a worker can be in this setting.

Problem Formulation: For a given number of workers N, suppose that (for $i = 0, \ldots, N-1$) W_i divides its gradient $g_{\mathcal{D}_i}$ into c chunks. A round consists of clockwise communication over the ring where W_i sends W_{i+1} (indices reduced modulo-N) a real-valued vector of length $\tilde{c}$. Suppose that a given protocol communicates over L rounds. The requirement is that worker W_i should be able to reconstruct an approximate gradient $\hat{\mathbf{g}}_i$ at the end of these rounds such that

$$\|\hat{\mathbf{g}}_i - \nabla L\|_2 \leq \tilde{\delta}, \text{ for all } i = 0, \ldots, N-1, \tag{1}$$

where $\tilde{\delta} > 0$ is a small constant. Thus, the total number of real values communicated in the protocol per worker is $L\tilde{c}$. We define the normalized rate of the protocol as $\frac{L\tilde{c}}{d}$, e.g., the basic Ring All-Reduce operates with $L = 2(N-1), c = N, \tilde{c} = d/c$ and with $\tilde{\delta} = 0$ (exact reconstruction).

Related Work: Gradient quantization Alistarh et al. (2017); Gandikota et al. (2022); Yu et al. (2018), and sparsification Alistarh et al. (2018); Sahu et al. (2021); Wangni et al. (2018) aim at reducing the communication load of distributed training by "compressing" or "sparsifying" the gradients and then communicating the compressed representations. These approaches address communication reduction along a direction orthogonal to ours. Specifically, it may be possible to apply our techniques in conjunction with them.

Contributions in the area of gradient coding Tandon et al. (2017); Raviv et al. (2018); Ye & Abbe (2018); Ramamoorthy et al. (2024); Munim & Ramamoorthy (2026) aim at making distributed training robust to worker failures/slowdowns; however, these are in the PS + workers model and moreover require redundancy in the assignment of chunks to workers. Coded distributed computing Li et al. (2018); Konstantinidis & Ramamoorthy (2020); Huang et al. (2026) is similar in the sense

that it trades off computational redundancy for reduced communication for exact computation. Our work is related to approximate gradient coding since we rely critically on the judicious usage of linear combinations of gradient chunks and aim at recovering approximate gradients. However, crucially, our setting remains the same as the basic Ring All-Reduce; we do not need redundant chunk assignments.

Main Contributions: In this work, we present a **C**ommunication-**E**fficient, Approximate **R**ing **A**ll-**R**educe protocol named CERAR. (i) We demonstrate that CERAR can achieve arbitrarily small error for a so-called Vandermonde construction, while operating with normalized communication and storage rates close to 1. The caveat is that this is achieved via increasingly ill-conditioned encoding and decoding matrices. (ii) We characterize the tradeoff between the error and numerical stability of the scheme by explicitly deriving a scheme where the error is $O(\epsilon)$ and the condition number of the matrices is $O(\epsilon^{-r_\star})$ where $r_\star = \lceil c/N \rceil - 1$. (iii) We then propose a constrained optimization problem that trades off error for numerical stability in our scheme. Numerical experiments demonstrate the effectiveness of our approach.

2 Communication Efficient Ring All Reduce (CERAR) Protocol

The gradients are subdivided into c chunks, i.e., $g_{\mathcal{D}_i} = [g_{\mathcal{D}_i}[0]^T \ \cdots \ g_{\mathcal{D}_i}[c-1]^T]^T$. In each of the $L = c + N - 2$ rounds (numbered 1 through L), the transmitted vector is of length $p = d/c$ (we assume $c \mid d$, which can be enforced by zero-padding). Let $\mathbf{G}_i := [g_{\mathcal{D}_i}[0] \quad \cdots \quad g_{\mathcal{D}_i}[c-1]] \in \mathbb{R}^{p\times c}$ and let $\mathcal{G}^{(i)} = [\mathbf{G}_i \mid \ldots \mid \mathbf{G}_{i+N-1}]$ (indices reduced modulo-N).

Initialization: Worker W_i has a $c \times c$ matrix $\mathbf{U}_i = [u_{i,1} \quad \cdots \quad u_{i,c}]$. After computing $\mathbf{G}_i$, it creates encoded vectors

$$\Delta_i := [\delta_{i,1} \quad \cdots \quad \delta_{i,c}] = \mathbf{G}_i\mathbf{U}_i.$$

When $\mathbf{U}_i$ is nonsingular, the map $\mathbf{G}_i \mapsto \Delta_i$ is an invertible change of basis. Consequently, the same d-length memory region that initially contains $g_{\mathcal{D}_i}$ can be transformed in place so that it contains Δ_i. We denote the corresponding p-length storage slots by $D_{i,1}, \ldots, D_{i,c}$, such that $D_{i,j}$ stores $\delta_{i,j}$. In addition, we allocate $N-1$ additional p-length empty dedicated buffers in W_i denoted $B_{i,1}, \ldots, B_{i,N-1}$, and one rolling p-length buffer denoted C_i. C_i holds the current state used in the next transmission and can also serve as the scratch buffer for the in-place transform $\mathbf{G}_i \mapsto \Delta_i$ before communication begins. The buffer C_i is initialized to the "rolling state" $\sigma_{i,0} = 0$.

The formal definition of the protocol appears in Algorithm 1, which describes the communication and storage updates at each worker. For round k, where $1 \le k \le c$, worker W_i adds the local encoded vector $\delta_{i,k}$ to its rolling state $\sigma_{i,k-1}$ and transmits the result $x_{i,k}$ to W_{i+1}, and for $c+1 \le k \le L$ it simply transmits $x_{i,k} = \sigma_{i,k-1}$. The first $N-1$ received vectors (in rounds 1 through $N-1$) are stored in dedicated buffers ($B_{i,j}$'s for W_i). For round $k = j + N - 1$ (where $j \ge 1$), W_i subtracts its own returning local contribution, namely $\delta_{i,j}$ from the received vector to obtain the next rolling state $\sigma_{i,k}$, while overwriting the corresponding local buffer $D_{i,j}$ with the received vector for final decoding. After $L = c + N - 2$ rounds, the stored observations, together with the final unmodified local encoded vector $\delta_{i,c}$, form Y_i, from which the approximate gradient is reconstructed using R_i. The update equations at W_i are (worker indices reduced modulo-N)

$$x_{i,k} = \begin{cases} \delta_{i,k} + \sigma_{i,k-1}, & 1 \le k \le c, \\ \sigma_{i,k-1}, & c+1 \le k \le L \end{cases} \ \& \ \sigma_{i,k} = \begin{cases} x_{i-1,k}, & 1 \le k \le N-1, \\ x_{i-1,k} - \delta_{i,k-N+1}, & N \le k \le L. \end{cases} \tag{2}$$

2.1 Illustrative Example with $N = 4, c = 5, L = 7$

We explain the basic protocol for the case of four workers and seven rounds, i.e., $N = 4, c = 5, L = 7$. In round 1, worker W_i receives $x_{i-1,1} = \delta_{i-1,1}$ (since $\sigma_{i,0} = 0$, for all W_i) from its preceding worker, W_{i-1} and stores it in $B_{i,1}$. At the end of round 1, W_i updates its storage to $\sigma_{i,1} = x_{i-1,1} = \delta_{i-1,1}$. In round 4 (see Fig. 1), W_i receives the following vector from W_{i-1}.

$$x_{i-1,4} = \delta_{i-1,4} + \sigma_{i-1,3} = \delta_{i-1,4} + \delta_{i-2,3} + \delta_{i-3,2} + \delta_{i-4,1},$$

where the second equality follows by (2). Note that $i - 4 \equiv i \pmod 4$, i.e., $\delta_{i,1}$ needs to be canceled at W_i. Therefore,

$$\sigma_{i,4} = x_{i-1,4} - \delta_{i,1} = \delta_{i-1,4} + \delta_{i-2,3} + \delta_{i-3,2},$$

Algorithm 1 Communication and storage updates at worker W_i

Input: $N \geq 2$, $L = c + N - 2$, and encoded vectors $\delta_{i,1}, \ldots, \delta_{i,c}$
Output: Stored observations Y_i for final decoding

Worker indices are reduced modulo N. All workers execute each round concurrently.

for $j = 1, \ldots, c$ **do**
  $D_{i,j} \leftarrow \delta_{i,j}$
end for
$C_i \leftarrow 0$ ▷ Initial state $\sigma_{i,0}$
for $k = 1, \ldots, L$ **do**
Step 1: Generate, accumulate, and transmit
  if $k \leq c$ **then**
    $C_i \leftarrow C_i + D_{i,k}$
  end if
  Transmit C_i to W_{i+1} as $x_{i,k}$
  Complete the outgoing send before overwriting C_i.
Step 2: Receive, store, and update
  Receive $x_{i-1,k}$ from W_{i-1} into C_i
  if $k \leq N - 1$ **then**
    $B_{i,k} \leftarrow C_i$
  else
    $j \leftarrow k - N + 1$
    $C_i \leftarrow C_i - D_{i,j}$ ▷ Cancel the returning local contribution
    $D_{i,j} \leftarrow D_{i,j} + C_i$ ▷ Store $x_{i-1,k}$ for decoding
  end if
  The buffer C_i now contains $\sigma_{i,k}$.
end for
$Y_i \leftarrow [\, B_{i,1} \mid \cdots \mid B_{i,N-1} \mid D_{i,1} \mid \cdots \mid D_{i,c} \,]$ ▷ Logical ordering of the existing buffers

and $D_{i,1} = x_{i-1,4}$. This process continues until round 5 (note that $c = 5$ here), following which the workers simply forward their stored values. Thus, in round 6, W_i simply receives $\sigma_{i-1,5}$. For example, W_0 receives $\sigma_{3,5} = \delta_{2,5} + \delta_{1,4} + \delta_{0,3}$ and it calculates

$$\sigma_{0,6} = \delta_{2,5} + \delta_{1,4}$$

by canceling $\delta_{0,3}$. It can be observed that the received vectors at W_i at each round can be compactly represented as matrix $\mathbf{M}_i$ of dimension $Nc \times (L+1)$. For example, $\mathbf{M}_0$ and $\mathbf{M}_1$ are given as

$$\mathbf{M}_0 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} & u_{0,5} \\ \mathbf{0} & \mathbf{0} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,5} & \mathbf{0} \\ \mathbf{0} & u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,5} & \mathbf{0} & \mathbf{0} \\ u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \text{and} \tag{3}$$

$$\mathbf{M}_1 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,5} \\ \mathbf{0} & \mathbf{0} & u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,5} & \mathbf{0} \\ \mathbf{0} & u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} & \mathbf{0} & \mathbf{0} \\ u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} & u_{0,5} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4}$$

$\mathbf{M}_2$ and $\mathbf{M}_3$ can be obtained in a similar manner. For instance, the interpretation of the fourth column of $\mathbf{M}_0$ is that it receives the p-length vector

$$y_{0,4} = \sum_{\ell=0}^{c-1} u_{0,1}[\ell] g_{\mathcal{D}_0}[\ell] + \sum_{\ell=0}^{c-1} u_{1,2}[\ell] g_{\mathcal{D}_1}[\ell] + \sum_{\ell=0}^{c-1} u_{2,3}[\ell] g_{\mathcal{D}_2}[\ell] + \sum_{\ell=0}^{c-1} u_{3,4}[\ell] g_{\mathcal{D}_3}[\ell]$$

from W_3 in round 4. $y_{0,4}$ is stored in $D_{0,1}$ after overwriting $\delta_{0,1}$. The last column corresponds to $y_{0,8} = \delta_{0,5}$ that is contained in $D_{0,5}$. It is not overwritten as there are only $L = 7$ rounds.

The worker W_i wants to obtain $\hat{\mathbf{g}}_i$ (see (1)). Thus, W_0, e.g., wants to find a decoding vector $\mathbf{r}_1$ of length-$(L+1)$ such that it can approximately recover $\sum_{i=0}^{N-1} g_{\mathcal{D}_i}[0]$, i.e.,

$$\sum_{i=1}^{L+1} \mathbf{r}_{1i} y_{0,i} \approx \sum_{i=0}^{N-1} g_{\mathcal{D}_i}[0].$$

This in turn implies that we need $\mathbf{M}_0\mathbf{r}_1 \approx [e_0^T\ e_0^T\ e_0^T\ e_0^T]^T = \mathbb{1}_4 \otimes e_0$ ($\otimes$ - Kronecker product), where $e_i, i = 0, \ldots, c-1$ denotes the canonical basis vector in $\mathbb{R}^c$. In a similar manner it can be observed that we need the existence of other decoding vectors such that $\mathbb{1}_N \otimes e_i$ for $i = 1, \ldots, c-1$ can be decoded from each of them. This can be compactly represented as the existence of a $(L+1) \times c$ matrix $\mathbf{R}_i$ such that $\mathbf{M}_i\mathbf{R}_i \approx \mathbb{1}_N \otimes I_c$ ($\mathbb{1}_N$ - all-ones vector of length-N, I_c - $c \times c$ identity matrix). Suppose that the protocol identifies suitable $\mathbf{M}_i$ and $\mathbf{R}_i$ such that $\|\mathbf{M}_i\mathbf{R}_i - \mathbb{1}_N \otimes I_c\|_F \leq \delta_{\rm dec}$ for $i = 0, \ldots, N-1$, where $\|\cdot\|_F$ denotes the Frobenius norm. Then, we have

$$\|\hat{\mathbf{g}}_i - \nabla L\|_2 = \left\|\mathcal{G}^{(i)}(\mathbf{M}_i\mathbf{R}_i - \mathbb{1}_N \otimes I_c)\right\|_F \leq \left\|\mathcal{G}^{(i)}\right\|_2 \delta_{\rm dec} = \left\|\mathcal{G}^{(0)}\right\|_2 \delta_{\rm dec}, \tag{5}$$

where $\delta_{\rm dec}$ is the decoding error, and $\|\cdot\|_2$ denotes the spectral norm. The inequality is standard, and the second equality holds since the $\mathcal{G}^{(i)}$'s are related by permutations. Thus, assuming that $\left\|\mathcal{G}^{(0)}\right\|_2 \leq C_G$ uniformly over the gradients under consideration, the overall error in the gradient computation (*cf.* (1)) can be made small enough by choosing $\delta_{\rm dec}$ small enough. Appendix §5.1 shows that the normalized rate of any linear protocol satisfying (1) for sufficiently small $\tilde{\delta}$ (under the bounded spectral norm assumption) is at least 1.

2.2 Discussion of the CERAR protocol

The total storage per worker includes $D_{i,j}, j = 1, \ldots, c$, the buffers $B_{i,j}, j = 1, \ldots, N-1$ and the buffer C_i that contains the rolling state. Thus, the normalized storage rate and communication rate of CERAR are given by

$$\mathcal{S} = 1 + \frac{N}{c}, \qquad \text{and} \qquad \mathcal{R} = \frac{L}{c} = 1 + \frac{N-2}{c}.$$

We note that the storage rate of the basic Ring All-Reduce is exactly 1. Let $\mathbf{U}_k, k = 0, \ldots, N-1$ denote the collection of the $u_{k,j}, 1 \leq j \leq c$ variables, i.e., $\mathbf{U}_k = [u_{k,1}\ u_{k,2}\ \ldots\ u_{k,c}]$ and let $\mathbf{U} = (\mathbf{U}_0, \ldots, \mathbf{U}_{N-1})$; these are the encoding matrices. Each $\mathbf{M}_i, 0 \leq i \leq N-1$ denotes a $Nc \times (L+1)$ matrix that is obtained as a result of the CERAR protocol (examples in (3), (4)); henceforth we denote it as $\mathbf{M}_i(\mathbf{U})$ to show the dependence explicitly. Furthermore, note that the r-th block-row of $\mathbf{M}_i(\mathbf{U})$ of dimension $c \times (L+1)$ can be expressed as

$$\mathbf{M}_i(\mathbf{U})[r,:] = [\mathbf{0}_{c\times(N-1-r)}\ \mathbf{U}_{i+r}\ \mathbf{0}_{c\times r}] \text{ for } r = 0, \ldots, N-1. \tag{6}$$

Here the subscript $i + r$ is reduced modulo-N. Let $\mathbf{R}_i, 0 \leq i \leq N-1$ denote $(L+1) \times c$ decoding matrices; let $\mathbf{R} = (\mathbf{R}_0, \ldots, \mathbf{R}_{N-1})$, and the target $\mathbf{T} = \mathbb{1}_N \otimes I_c$. For $\mathbf{R}_i$, we denote the c consecutive rows starting at row s by $\mathcal{W}_s(\mathbf{R}_i) = \mathbf{R}_i[s : s+c-1, :]$ for $s = 0, \ldots, N-1$, where the rows are numbered $0, \ldots, L$. The least-squares fit in our scenario is measured by

$$\Phi(\mathbf{U}, \mathbf{R}) = \sum_{i=0}^{N-1} \|\mathbf{M}_i(\mathbf{U})\mathbf{R}_i - \mathbf{T}\|_F^2 . \tag{7}$$

We note here that both $\mathcal{R}$ and $\mathcal{S}$ are decreasing in c, and the basic Ring All-Reduce already provides a zero-error solution for the case when $c = N$. For the remainder of the paper we only consider the regime $c > N$. Lemma 1 is proved in Appendix, § 5.2.

Lemma 1*:* Suppose that $c > N$. There is no finite collection of encoding and decoding matrices $(\mathbf{U}, \mathbf{R})$ for which $\Phi(\mathbf{U}, \mathbf{R}) = 0$.

2.2.1 Multiplicative Perturbation Construction (MPC)

We now present a specific construction of the $\mathbf{U}$ and $\mathbf{R}$ matrices that are obtained by (i) a $N \times c$ base matrix $\mathbf{B}$ with rows $b_0, \ldots, b_{N-1}$, each of length-c, and (ii) a $c \times c$ nonsingular matrix $\mathbf{D}$ that perturbs $\mathbf{B}$ by right-multiplication. Define for $r = 0, \ldots, N-1$ and an integer m, the row-vector

$$a_{r+mN} = b_r\mathbf{D}^m. \tag{8}$$

Thus, the above vector is defined for all integer indices for the LHS of (8). Note that we have $a_{\ell+N} = a_\ell\mathbf{D}$. For every integer ℓ, define the $c \times c$ consecutive-row window

$$A_\ell = \begin{bmatrix} a_\ell \\ a_{\ell+1} \\ \vdots \\ a_{\ell+c-1} \end{bmatrix}. \tag{9}$$

It follows that $A_{\ell+N} = A_\ell \mathbf{D}$. We assume that $A_0, \ldots, A_{N-1}$ are nonsingular. Define

$$\mathbf{V}_{N-1-k} := A_k, \qquad \mathbf{U}_k := \mathbf{V}_k^{-1}, \qquad k = 0, \ldots, N-1. \tag{10}$$

For worker W_i, define the explicit decoder $\mathbf{R}_i \in \mathbb{R}^{(L+1)\times c}$ by taking the $L+1$ consecutive rows:

$$\mathbf{R}_i[\ell, :] = a_{\ell-i}, \qquad \ell = 0, \ldots, L. \tag{11}$$

Equations (10) and (11) specify the construction of the scheme; the s-th c-row window of $\mathbf{R}_i$ is

$$\mathcal{W}_s(\mathbf{R}_i) = A_{s-i}, \qquad s = 0, \ldots, N-1. \tag{12}$$

Proposition 1*:* Let worker indices be reduced modulo-N. For the construction above,

$$\mathbf{U}_{i-1-s}\mathcal{W}_s(\mathbf{R}_i) = \begin{cases} I_c, & s \geq i, \\ \mathbf{D}^{-1}, & s < i. \end{cases}$$

Proof. If $s \geq i$, then $s - i \in \{0, \ldots, N-1\}$. By (10),

$$A_{s-i} = \mathbf{V}_{i-1-s}, \text{(reducing the index of RHS modulo-}N\text{)}$$

Therefore, $\mathbf{U}_{i-1-s}\mathcal{W}_s(\mathbf{R}_i) = \mathbf{V}_{i-1-s}^{-1} A_{s-i} = I_c$.

If $s < i$, we can assert $A_{s-i} = A_{s-i+N}\mathbf{D}^{-1}$ where $s - i + N \geq 0$ since $s - i \geq -(N-1)$. Again,

$$A_{s-i+N} = \mathbf{V}_{i-1-s}, \text{ and hence}$$

$$\mathbf{U}_{i-1-s}\mathcal{W}_s(\mathbf{R}_i) = \mathbf{V}_{i-1-s}^{-1} A_{s-i+N}\mathbf{D}^{-1} = \mathbf{D}^{-1}. \qquad \square$$

Proposition 2*:* Let $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{R}}$ denote the collections of matrices constructed as in (10) and (11). Then,

$$\inf_{\mathbf{U},\mathbf{R}} \Phi(\mathbf{U}, \mathbf{R}) \leq \Phi(\tilde{\mathbf{U}}, \tilde{\mathbf{R}}) = \frac{N(N-1)}{2} \|\mathbf{D}^{-1} - I_c\|_F^2. \tag{13}$$

Proof. Worker W_i has exactly i indices $s \in \{0, \ldots, N-1\}$ for which $s < i$. By Proposition 1, these are precisely the nonzero residual blocks, and each is equal to $\mathbf{D}^{-1} - I_c$. Therefore

$$\Phi(\tilde{\mathbf{U}}, \tilde{\mathbf{R}}) = \sum_{i=0}^{N-1} i \, \|\mathbf{D}^{-1} - I_c\|_F^2 = \frac{N(N-1)}{2} \|\mathbf{D}^{-1} - I_c\|_F^2.$$

The matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{R}}$ specified in (10) and (11) are feasible choices in the least-squares minimization in (7), so the optimized least-squares residual cannot exceed their residual. $\square$

2.2.2 A VANDERMONDE CONSTRUCTION THAT ACHIEVES ARBITRARILY SMALL $\Phi(\mathbf{U}, \mathbf{R})$

Let $\xi_1, \ldots, \xi_c$ be distinct reals, and for $\tau > 0$, set $z_j(\tau) := e^{\tau \xi_j}$ for $j = 1, \ldots, c$. Define

$$b_\ell(\tau) := \begin{bmatrix} z_1(\tau)^\ell & \cdots & z_c(\tau)^\ell \end{bmatrix}, \text{ for } \ell = 0, \ldots, N-1, \text{ and}$$

$$\mathbf{D}_\tau := \operatorname{diag}\big(z_1(\tau)^N, \ldots, z_c(\tau)^N\big) = \operatorname{diag}\big(e^{N\tau\xi_1}, \ldots, e^{N\tau\xi_c}\big).$$

This defines the sequence specified in (8), and therefore A_ℓ. Henceforth, we refer to it as $A_\ell(\tau)$ to emphasize the dependence on τ; it is a shifted Vandermonde matrix with nodes $z_1(\tau), \ldots z_c(\tau)$ and hence nonsingular Horn & Johnson (2012). In particular, this choice satisfies the requirement that $A_0(\tau), \ldots, A_{N-1}(\tau)$ are nonsingular. We call this the Vandermonde construction. The following theorem follows from an application of Proposition 2 (details in Appendix, §5.3).

Theorem 1*:* Let $\tilde{\mathbf{U}}_\tau$ and $\tilde{\mathbf{R}}_\tau$ be the matrices obtained by applying the Vandermonde construction. Then, $\Phi(\tilde{\mathbf{U}}_\tau, \tilde{\mathbf{R}}_\tau) = \frac{N^3(N-1)}{2}\tau^2 \sum_{j=1}^c \xi_j^2 + O(\tau^3) \to 0$ as $\tau \downarrow 0$ so that $\inf_{\mathbf{U},\mathbf{R}} \Phi(\mathbf{U}, \mathbf{R}) = 0$. If $c > N$, the infimum is not attained.

Remark 1*:* Choosing a sequence τ_c (dependent on c) such that $\tau_c^2 \sum_{j=1}^c \xi_j^2 \to 0$, this shows a rather striking fact that one can simultaneously achieve a normalized rate and storage of 1 as $c \to \infty$[1] while driving the approximation error in the reconstruction arbitrarily close to zero. To our best knowledge, this is the first work that shows such a result for the well-known All-Reduce setting.

Remark 2*:* As $\tau \downarrow 0$, all Vandermonde nodes satisfy $z_j(\tau) \longrightarrow 1$, so that $\mathbf{V}_k(\tau)\mathbf{V}_k(\tau)^T \to c\mathbb{1}\mathbb{1}^T$. Thus, the condition number (ratio of maximum to minimum singular values) $\kappa_2(\mathbf{V}_k(\tau)) = \kappa_2(\mathbf{U}_k(\tau)) \longrightarrow \infty$, i.e., this construction is numerically unstable for small τ. Thus, in practice, one needs to adapt the approach appropriately.

[1] We need $d \to \infty$ as well. In practice d is many orders of magnitude larger than c, so this is not a limitation.

2.3 TRADE-OFF BETWEEN LS ERROR AND CONDITIONING

We now examine the role of the multiplicative perturbation $\mathbf{D}$ on the conditioning of the $\mathbf{M}_i$ matrices. Let $\mathbf{D}_\epsilon = I_c + \epsilon\mathbf{H}$ with $\epsilon \|\mathbf{H}\|_2 < 1$ for a fixed matrix $\mathbf{H}$. This implies that $\left\|\mathbf{D}_\epsilon^{-1}\right\|_2 = \frac{1}{\sigma_{\min}(\mathbf{D}_\epsilon)} \leq \frac{1}{1-\epsilon\|\mathbf{H}\|_2}$. Moreover, $\mathbf{D}_\epsilon^{-1} - I_c = -\epsilon\mathbf{D}_\epsilon^{-1}\mathbf{H}$, so that $\left\|\mathbf{D}_\epsilon^{-1} - I_c\right\|_F \leq \epsilon \left\|\mathbf{D}_\epsilon^{-1}\right\|_2 \|\mathbf{H}\|_F$.

Applying the result of Proposition 2 (nonsingularity of the corresponding $A_\ell(\epsilon)$ follows from the forthcoming assumptions and discussion), we can conclude

$$\Phi(\mathbf{U}, \mathbf{R}) \leq \frac{N(N-1)}{2} \frac{\epsilon^2 \|\mathbf{H}\|_F^2}{(1 - \epsilon \|\mathbf{H}\|_2)^2} = O(\epsilon^2). \tag{14}$$

Consider the $c \times c$ consecutive-row window $A_\ell(\epsilon)$ where $\ell \in \{0, \ldots, N-1\}$ that is formed by using $\mathbf{B}$ and $\mathbf{D}_\epsilon$ (*cf.* (9)). These are the kind of window types that arise in our construction of the decoding matrices $\mathbf{R}_i$'s, except that they are sometimes right-multiplied by powers of $\mathbf{D}_\epsilon$. However, $\kappa_2(\mathbf{D}_\epsilon) = 1 + O(\epsilon)$ (condition number - ratio of maximum to minimum singular value) so it does not affect the scaling of $\kappa_2(A_\ell(\epsilon))$.

Let $c = \beta N + s$, where $0 \leq s < N$ and define $r_\star = \left\lceil \frac{c}{N} \right\rceil - 1$. Let $E_\ell = \{\ell, \ell+1, \ldots, \ell+s-1\} \pmod N$. For $j \in \{\ell, \ell+1, \ldots, \ell+c-1\}$, consider its remainder upon division by N. If $s > 0$, then a given remainder $\alpha \in \{0, \ldots, N-1\}$ occurs $n_\alpha(\ell)$ times where

$$n_\alpha(\ell) = \begin{cases} \beta + 1, & \text{if } \alpha \in E_\ell \\ \beta, & \text{if } \alpha \notin E_\ell. \end{cases}$$

If $s = 0$, then every remainder occurs exactly β times. For $\ell \in \{0, \ldots, N-1\}$, the smallest index $j \geq \ell$ satisfying $j \equiv \alpha \pmod N$ is $j = \alpha + N m_\alpha(\ell)$, where

$$m_\alpha(\ell) = \begin{cases} 0, & \alpha \geq \ell, \\ 1, & \alpha < \ell. \end{cases}$$

Thus, the first row associated with remainder α is $b_\alpha \mathbf{D}_\epsilon^{m_\alpha(\ell)}$, followed by $b_\alpha \mathbf{D}_\epsilon^{m_\alpha(\ell)+1}, \ldots, b_\alpha \mathbf{D}_\epsilon^{m_\alpha(\ell)+n_\alpha(\ell)-1}$.

Let F_n denote the $n \times n$ lower-triangular finite-difference matrix (Appendix, §5.5), and Π_ℓ be a permutation matrix that groups the rows of A_ℓ by remainder class, and define $\tilde{F}_\ell = \text{diag}(F_{n_0(\ell)}, F_{n_1(\ell)}, \ldots, F_{n_{N-1}(\ell)})$. Then, using Lemma 2 (Appendix, §5.5) we have

$$\tilde{F}_\ell \Pi_\ell A_\ell(\epsilon) = \Lambda_\ell(\epsilon) K_\ell(\epsilon) \tag{15}$$

where $\Lambda_\ell(\epsilon)$ is a diagonal matrix with entries $\{\epsilon^j : j = 0, \ldots, n_r(\ell) - 1, r = 0, \ldots, N-1\}$ and the corresponding rows of $K_\ell(\epsilon)$ are $b_r \mathbf{D}_\epsilon^{m_r(\ell)} \mathbf{H}^j$. We note that as $\epsilon \downarrow 0$, we have $\mathbf{D}_\epsilon^{m_r(\ell)} \to I_c$. This implies that up to a row-permutation, $K_\ell(\epsilon)$ approaches a matrix denoted $K^{(\ell)}(\mathbf{B}, \mathbf{H})$ where

$$K^{(\ell)}(\mathbf{B}, \mathbf{H}) = \begin{bmatrix} \mathbf{B} \\ \mathbf{BH} \\ \vdots \\ \mathbf{BH}^{\beta-1} \end{bmatrix} \text{ if } s = 0, \quad \text{and} \quad K^{(\ell)}(\mathbf{B}, \mathbf{H}) = \begin{bmatrix} \mathbf{B} \\ \mathbf{BH} \\ \vdots \\ \mathbf{BH}^{\beta-1} \\ (\mathbf{BH}^\beta)_{E_\ell} \end{bmatrix} \text{ if } s > 0.$$

Here $(\mathbf{BH}^\beta)_{E_\ell}$ denotes the s rows indexed by the set E_ℓ. An example showing the calculation in (15) appears in Appendix §5.5.

It is not too hard to see that the maximum singular value of $K^{(\ell)}(\mathbf{B}, \mathbf{H})$ is upper bounded by $C_1 \sigma_{\max}(\mathbf{B})$ where C_1 is a constant that depends on $r_\star$ and $\sigma_{\max}(\mathbf{H})$ (see Appendix, §5.6).

Assumption: We assume that $K^{(\ell)}(\mathbf{B}, \mathbf{H})$ is nonsingular and further that $\min_{\ell=0,\ldots,N-1} \sigma_{\min}(K^{(\ell)}(\mathbf{B}, \mathbf{H})) = \gamma > 0$. This can be justified through a Schwartz-Zippel type argument, and through multiple trials for selecting $\mathbf{H}$. Constructions of $\mathbf{H}$ that depend on $\mathbf{B}$ can also be considered.

In the following theorem, $N, c, \mathbf{B}$, and $\mathbf{H}$ are fixed, and all asymptotic notation refers to $\epsilon \to 0$. The proof appears in the Appendix, §5.4.

Table 1: Raw reduction time in milliseconds; lower is better. M := 10^6, B := 10^9.

System / interconnect	Tensor size	NCCL	Our algorithm	Gain
A100 PCIe, 8 GPUs / PCIe	100M	1,775.9	940.1	47.1%
A100 PCIe, 8 GPUs / PCIe	500M	8,611.2	4,651.0	46.0%
A100 PCIe, 8 GPUs / PCIe	1B	17,954.4	9,558.6	46.8%
H200 SXM, 4 GPUs / NVLink	9B	147.250	197.097	−33.9%
H200 SXM, 4 GPUs / NVLink	15B	244.974	327.733	−33.8%
H200 SXM, 4 GPUs / NVLink	24B	392.049	522.366	−33.2%

Theorem 2 (Singular value scaling)*:* The singular values of $A_\ell(\epsilon)$ have the same powers of ϵ, up to constant factors, as the diagonal entries of $\Lambda_\ell(\epsilon)$. Specifically, (i) If $s = 0$, then for each $j = 0, \ldots, \beta - 1$, exactly N singular values are $\Theta(\epsilon^j)$. (ii) If $s > 0$, then for each $j = 0, \ldots, \beta - 1$, exactly N singular values are $\Theta(\epsilon^j)$ and the remaining s singular values are $\Theta(\epsilon^\beta)$.

Corollary 1 (Exact smallest-singular-value exponent)*:* For $r_\star = \lceil \frac{c}{N} \rceil - 1$ and every $\ell = 0, \ldots, N-1$,

$$\sigma_{\min}(A_\ell(\epsilon)) = \Theta(\epsilon^{r_\star}), \qquad c_0 \leq \sigma_{\max}(A_\ell(\epsilon)) \leq \tilde{C}_1 \|B\|_2 ,$$

where c_0 is a constant independent of ϵ. Thus, $\sigma_{\max}(A_\ell(\epsilon)) = \Theta(1)$. Thus, $\kappa_2(A_\ell(\epsilon)) = \Theta(\epsilon^{-r_\star})$. Furthermore, since the $\mathbf{U}_i$'s are formed as inverses of matrices of the form $A_\ell(\epsilon)$, we also have that $\kappa_2(\mathbf{U}_i) = \Theta(\epsilon^{-r_\star})$. From the discussion in Appendix, §5.7 we can conclude that $\kappa_2(\mathbf{M}_i) = O(\sqrt{N}\epsilon^{-r_\star})$ for all $i = 0, \ldots, N-1$.

Remark 3*:* Corollary 1 does not apply to the construction in Section 2.2.2 since the Vandermonde construction does not work with a fixed $\mathbf{B}$ and $\mathbf{H}$. In particular, the corresponding definitions of $\mathbf{B}$ and $\mathbf{H}$ change with the parameter τ that goes to zero.

2.4 Constrained Optimization Problem Formulation

The construction in Section 2.2.1 guarantees LS squared error $O(\epsilon^2)$, and $O(\sqrt{N}\epsilon^{-r_\star})$ conditioning of the $\mathbf{M}_i$ matrices. Using this solution as an initialization for the following constrained optimization, it is possible to trade off these competing objectives. Consider

$$\begin{aligned} &\text{minimize } \Phi(\mathbf{U}, \mathbf{R}) \\ &\text{subject to } \kappa_2(\mathbf{M}_i(\mathbf{U})) \leq C, \text{ for } i = 0, \ldots, N-1. \end{aligned} \tag{16}$$

Although (16) is nonconvex, we use an alternating least-squares heuristic in which the $\mathbf{U}_i$'s are projected onto a stricter spectral constraint set that is sufficient to guarantee the condition-number constraint. In particular, in one iteration, we minimize $\Phi(\mathbf{U}, \mathbf{R})$ with respect to $\mathbf{U}$ while keeping $\mathbf{R}$ fixed, and vice versa. After this, we project the computed singular values of the $\mathbf{U}_i$'s into a common interval $a_0 \leq \sigma_{\min}(\mathbf{U}_i) \leq \sigma_{\max}(\mathbf{U}_i) \leq b_0$. Note that this implies that $\kappa_2(\mathbf{M}_i) \leq \sqrt{N} b_0/a_0$. Therefore, the constant C in (16) needs to be chosen so that $\sqrt{N} b_0/a_0 \leq C$.

3 Numerical Experiments

Our numerical experiments compared the performance of the CERAR protocol to the production-level NCCL All-Reduce by NVIDIA. We considered GPU clusters under two different settings - a low-bandwidth (PCIe, 31.5 GB/s) and a high-bandwidth (NVLink, $\geq$ 300 GB/s) interconnect. All code for recreating these experiments can be found in an anonymous repository. [2] The CERAR scheme was fit using the methods in Section 2.4 (details in Appendix, §5.8).

All-Reduce Phase time: We ran the evaluations on single-node systems with eight A100 GPUs and four H200 GPUs; each GPU started with a FP32 local tensor. CERAR used $c = 20, L = 26$ for the A100 experiments and $c = 12, L = 14$ for the H200 experiments. We measured the elapsed time for each GPU to complete the reduction, including communication and local computation. For our algorithm, this includes (i) encoding (preparing $\delta_{i,j}$'s), (ii) all communication rounds and their local updates, and (iii) decoding using the stored values and the $\mathbf{R}_i$ matrices to reconstruct

[2] https://anonymous.4open.science/r/CERAR--43C6/

Table 2: Training results in W_0 for Pythia-1.4B on two four-GPU configurations.

Method	Training time (hours)	Throughput (tokens/s)	Final val. loss	Peak memory (GB)
$4\times$ A100 PCIe 40GB (PCIe interconnect): 1,800 steps				
NCCL All-Reduce	3.79	8,644	1.4218	36.2
Our algorithm	3.14	10,451	1.4213	38.6
$4\times$ A100 SXM 80GB (NVLink interconnect): 7,630 steps				
NCCL All-Reduce	4.578	60,682	1.1644904	49.55
Our algorithm	4.745	58,543	1.1644902	51.91

Table 3: Relative ℓ_2 gradient error on four A100 SXM 80GB GPUs, measured against a separate NCCL reference reduction and maximized across workers.

Method	Step 1	Step 3,815	Step 7,630
NCCL All-Reduce	6.772×10^{-9}	5.532×10^{-9}	6.437×10^{-9}
Our algorithm	0.016078	0.028653	0.027179

the approximate sum. As shown in Table 1, CERAR outperforms the NCCL implementation on the A100 PCIe system, reducing the reduction time by 46.0–47.1%. However, on the H200 NVLink system, CERAR is 33.2%–33.9% slower than NCCL. Appendix §5.8 has more details and more fine-grained information about the time taken for the different parts of the All-Reduce phase.

We emphasize that NCCL is a highly optimized (open-source) NVIDIA implementation. In particular, it establishes multiple communication channels and incorporates overlapping computation and communication; these features greatly improve its performance. Our implementation of CERAR is meant more as a proof of concept and the implementation has not been optimized anywhere close to the level of NCCL. In addition, we emphasize that the gap between the methods will typically manifest in the regime when the parameter length is very high. Owing to access limitations, we were unable to benchmark reduction tensors $>$ 24 billion FP32 entries. However, we believe that an optimized implementation can yield tangible benefits even in the high-bandwidth NVLink systems.

Training time: We trained a Pythia-1.4B model Biderman et al. (2023), containing ≈ 1.41 billion parameters, on the TinyStories dataset Eldan & Li (2023). Both A100 systems used four GPUs with CERAR using $c = 12, L = 14$. For the PCIe experiment, we trained for 1800 iterations ($\approx 117.96 \times 10^6$ tokens), and for the SXM experiment, we trained for 7630 iterations ($\approx 1 \times 10^9$ tokens); more details in Appendix §5.8.

As shown in Table 2, on the PCIe configuration, NCCL requires 3.79 hours, whereas CERAR requires 3.14 hours; the corresponding validation losses at the end of the 1800 iterations are 1.4218 and 1.4213 respectively. Thus, our algorithm reduces the overall training time by 17.3%, saving approximately 39 minutes. On the SXM configuration, NCCL and our algorithm require 4.578 and 4.745 hours, respectively. The validation losses of both schemes are approximately 1.16449. However, our algorithm takes 3.65% longer.

Gradient Error: In the SXM experiment, we measured the relative ℓ_2 error of the reduced gradient at every training step. Table 3 reports the maximum error across workers at the first, middle, and final steps. For our algorithm, these errors are approximately 1.61%, 2.87%, and 2.72%, respectively, while those for NCCL are below 7×10^{-9} (consistent with floating-point roundoff). Nevertheless, these errors do not impact the training in an appreciable manner.

4 Conclusions and Future Work

This work demonstrates that Ring All-Reduce can be made much more communication-efficient when approximate recovery of the sum is sufficient and presents theoretical results guaranteeing asymptotic optimality amongst the class of linear protocols, and practical constructions that guarantee performance metrics (error and numerical stability). Future work will involve optimized implementations of our protocol that allow for practical benefits even in high-bandwidth GPU clusters.

AI USE STATEMENT

In this work, we used AI tools such as ChatGPT and Claude AI for the following tasks (required disclosure).

- *Proofreading and polishing the manuscript:* We used these tools to proofread the draft for typos, grammar and also to check for repeated statements and inconsistent writing.
- *Research Execution:* We used the tools for generating Python code for the optimization described in Section 2.4. We also used them to generate software code for deploying the actual algorithm on GPU clusters.
- *Proofreading and polishing mathematical claims:* We used the tools to check for mathematical consistency. In certain cases, the tools identified typos or suggested ways in which the proofs could be presented better. We accepted the suggestions in some instances and reworded the proofs accordingly. However, we emphasize that the basic ideas and mathematical claims were *not* generated by the AI tools.

We have *not* used AI tools for developing theoretical models/frameworks or formulating or providing critical ingredients for mathematical claims (see discussion above). The rest of the tasks with required disclosure are not applicable in our work.

Additionally, we used AI tools for the following tasks (recommended disclosure).

- *Creating and editing software code:* We have already covered this point under "Research Execution" in the required disclosure.
- *Identifying relevant literature and formatting references:* We used AI tools to do a more exhaustive search of the related literature and also used them to help prepare BibTeX files.

The rest of the tasks with recommended disclosure are not applicable in our work.

We have reviewed all the AI-assisted work: (i) the output of the LLM-generated optimization code is a collection of matrices. This collection was checked for accuracy. Likewise, the numerical experiments were performed on standard datasets and models and the results were cross-checked. (ii) All the polished proofs were rigorously proofread for correctness by the authors.

We take full responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Software code associated with this paper includes the following.

- Code for constrained optimization: For a given set of parameters N and c, we used Python code for implementing the constrained optimization described in §2.4 . This code outputs the corresponding encoding and decoding matrices, along with the value of the least-squares objective function and the condition number of all the relevant $\mathbf{M}_i, i = 0, \ldots, N-1$ matrices.
- Code for implementation on the GPUs: The numerical experiments section, §3 describes the results of our experiments on actual GPU clusters. Code for this was also generated and used in this work.

All of this code is well-commented and ready to use. These codes are available as part of the downloadable source code along with the submission and also in the anonymous repository linked in Section §3.

REFERENCES

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Adv. in Neural Inf. Process. Syst. (NeurIPS)*, 30, 2017.

Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. *Adv. in Neural Inf. Process. Syst. (NeurIPS)*, 31, 2018.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, May 2018.

Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent English?, 2023. URL https://arxiv.org/abs/2305.07759.

Venkata Gandikota, Daniel Kane, Raj Kumar Maity, and Arya Mazumdar. VQSGD: Vector quantized stochastic gradient descent. *IEEE Trans. on Info. Th.*, 68(7):4573–4587, 2022.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

Zhenhao Huang, Minquan Cheng, Kai Wan, Qifu Tyler Sun, and Youlong Wu. On the optimality of coded distributed computing for ring networks, 2026. URL https://arxiv.org/abs/2507.00091.

K. Konstantinidis and A. Ramamoorthy. Resolvable designs for speeding up distributed computing. *IEEE/ACM Trans. on Networking*, 28(4):1657–1670, 2020.

Matthias Langer, Zhen He, Wenny Rahayu, and Yanbo Xue. Distributed training of deep learning models: A taxonomic perspective. *IEEE Trans. on Parall. Distr. Sys.*, 31(12):2802–2818, 2020.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*, pp. 583–598, 2014.

S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Trans. on Info. Th.*, 64(1):109–128, January 2018.

Sifat Munim and Aditya Ramamoorthy. Communication-efficient approximate gradient coding. 2026. URL https://arxiv.org/abs/2603.22514.

NVIDIA. NVIDIA Collective Communication Library (NCCL) Documentation. URL https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html. Accessed: September 21, 2026.

Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009. doi: 10.1016/j.jpdc.2008.09.002.

Aditya Ramamoorthy, Ruoyu Meng, and Vrinda S. Girimaji. Leveraging partial stragglers within gradient coding. In *Adv. in Neural Inf. Process. Syst. (NeurIPS)*, December 2024.

Netanel Raviv, Rashish Tandon, Alex Dimakis, and Itzhak Tamo. Gradient coding from cyclic MDS codes and expander graphs. In *Intl. Conf. Mach. Learn. (ICML)*, pp. 4302–4310, July 2018.

Atal Sahu, Aritra Dutta, Ahmed M Abdelmoniem, Trambak Banerjee, Marco Canini, and Panos Kalnis. Rethinking gradient sparsification as total error minimization. *Adv. in Neural Inf. Process. Syst. (NeurIPS)*, 34:8133–8146, 2021.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *[Online] Available: https://arxiv.org/abs/1802.05799*, 2018.

Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *Intl. Conf. Mach. Learn. (ICML)*, pp. 3368–3376, August 2017.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. *Adv. in Neural Inf. Process. Syst. (NeurIPS)*, 31, 2018.

Min Ye and Emmanuel Abbe. Communication-computation efficient gradient coding. In *Intl. Conf. Mach. Learn. (ICML)*, pp. 5610–5619, 2018.

Mingchao Yu, Zhifeng Lin, Krishna Narra, Songze Li, Youjie Li, Nam Sung Kim, Alexander Schwing, Murali Annavaram, and Salman Avestimehr. Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. *Adv. in Neural Inf. Process. Syst. (NeurIPS)*, 31, 2018.

5 APPENDIX

5.1 PROOF THAT ANY LINEAR PROTOCOL NEEDS TO HAVE A NORMALIZED RATE OF AT LEAST 1

We say that a protocol is linear if it operates by linear encoding and decoding at the workers. Here, we prove that any linear protocol that satisfies the requirement in (1) has to have rate at least 1. We proceed by contradiction; assume that there exists a linear protocol satisfying (1) for all admissible gradients having $L\tilde{c} < d$. In what follows, we will construct two different gradient vectors such that the input to W_i is the same, but the actual values of the sum of the gradients are very different.

We note that at the end of the rounds worker W_i reconstructs an approximation to ∇L from the incoming vectors obtained from W_{i-1}. Suppose that we stack all the vectors received by W_i in a long vector of length $L\tilde{c}$, denoted β_i and suppose that $L\tilde{c} < d$. Now consider the linear map from the gradients outside W_i (namely $g_{\mathcal{D}_j}, j \neq i$) to β_i, and denote it Γ_i. This map Γ_i is from $\mathbb{R}^{(N-1)d} \to \mathbb{R}^{L\tilde{c}}$.

Now, we restrict our attention to gradients where every other worker has the same gradient $v \in \mathbb{R}^d$ and $g_{\mathcal{D}_i} = 0$. This provides a restricted linear map from $\mathbb{R}^d$ to $\mathbb{R}^{L\tilde{c}}$ denoted $\tilde{\Gamma}_i$.

Since $L\tilde{c} < d$, there exists a nonzero vector $h \in \mathbb{R}^d$ such that $\tilde{\Gamma}_i(h) = 0$. Let $t > 0$. Now, consider two different sets of gradient vectors: (i) $g_{\mathcal{D}_j} = th$ for $j \neq i$, and (ii) $g_{\mathcal{D}_j} = -th$ for $j \neq i$. Since $\tilde{\Gamma}_i(h) = 0$, worker W_i sees the same set of input vectors in both cases. Set

$$t = \frac{C_G}{\sqrt{N-1}\,\|h\|_2}.$$

For both sets of inputs $\left\|\mathcal{G}^{(0)}\right\|_2 \leq \left\|\mathcal{G}^{(0)}\right\|_F = \sqrt{N-1}t\,\|h\|_2 = C_G$. This means that both sets of gradient vectors satisfy the bounded spectral norm assumption in (5). However, their corresponding sums differ in norm by $2C_G\sqrt{N-1}$. Thus, W_i cannot recover the gradient within ℓ_2-error norm $\tilde{\delta}$ (see (1)) that is small enough for both sets of gradient vectors. This gives the required contradiction.

5.2 PROOF OF LEMMA 1

Assume, for contradiction, that $\Phi(\mathbf{U}, \mathbf{R}) = 0$. Then, we have

$$\mathbf{M}_i(\mathbf{U})\mathbf{R}_i = \mathbf{T},$$

for all W_i. We note that the r-th block of $\mathbf{M}_i(\mathbf{U})\mathbf{R}_i$ is given by $\mathbf{U}_{i+r}\mathcal{W}_{N-1-r}(\mathbf{R}_i)$ for $r = 0, \ldots, N-1$ (worker indices reduced modulo-N). This can also be expressed as $\mathbf{U}_{i-1-s}\mathcal{W}_s(\mathbf{R}_i)$ by setting $s = N-1-r$. This requires

$$\mathbf{U}_{i-1-s}\mathcal{W}_s(\mathbf{R}_i) = I_c, \qquad s = 0, \ldots, N-1. \tag{17}$$

Consequently, all $\mathbf{U}_i, i = 0, \ldots, N-1$ must be nonsingular. Defining $\mathbf{V}_i := \mathbf{U}_i^{-1}$, eq. (17) becomes

$$\mathcal{W}_s(\mathbf{R}_i) = \mathbf{V}_{i-1-s}. \tag{18}$$

Consecutive windows of the matrices $\mathbf{R}_i$ for $i = 0, \ldots, N-1$ overlap in $c-1$ rows. Taking consecutive values of s in (18) for all workers, therefore gives, for every k,

$$\mathbf{V}_k[2:c,:] = \mathbf{V}_{k-1}[1:c-1,:],$$

where the row indices of $\mathbf{V}_k$ are one-based.

Equivalently, row by row,

$$\mathbf{V}_k[t+1,:] = \mathbf{V}_{k-1}[t,:], \qquad t = 1, \ldots, c-1.$$

For any $j = 1, \ldots, c-N$, applying this identity N times yields

$$\begin{aligned}\mathbf{V}_k[j+N,:] &= \mathbf{V}_{k-1}[j+N-1,:] \\ &= \cdots = \mathbf{V}_{k-N}[j,:] = \mathbf{V}_k[j,:],\end{aligned}$$

where the last equality holds because $k \equiv (k-N) \pmod{N}$. Consequently, every row after the first N repeats an earlier row with the same index modulo N. Hence

$$\operatorname{rank}(\mathbf{V}_k) \leq N < c,$$

contradicting the nonsingularity of $\mathbf{V}_k$.

5.3 Error bound of Vandermonde Construction

Let $\tilde{\mathbf{U}}_\tau$ and $\tilde{\mathbf{R}}_\tau$ be the matrices obtained by applying the Vandermonde construction. Applying the result of Proposition 2, we have

$$0 \le \Phi(\tilde{\mathbf{U}}_\tau, \tilde{\mathbf{R}}_\tau) = \frac{N(N-1)}{2} \sum_{j=1}^{c} \left(e^{-N\tau\xi_j} - 1\right)^2$$
$$= \frac{N^3(N-1)}{2}\,\tau^2 \sum_{j=1}^{c} \xi_j^2 + O(\tau^3) \text{ (when } \tau \text{ is small).} \tag{19}$$

For fixed ξ_j, the right-hand side converges to zero quadratically as $\tau \downarrow 0$. Hence $\inf_{\mathbf{U},\mathbf{R}} \Phi(\mathbf{U}, \mathbf{R}) = 0$. When $c > N$, Lemma 1 shows that no finite $(\mathbf{U}, \mathbf{R})$ attains this value.

5.4 Proof of Theorem 2

We first note that since $K_\ell(\epsilon) \to P_\ell K^{(\ell)}(\mathbf{B}, \mathbf{H})$ for an appropriate row-permutation P_ℓ, we have by the continuity of singular values and the assumption that $\min_\ell \sigma_{\min}(K^{(\ell)}(\mathbf{B}, \mathbf{H})) = \gamma > 0$, there exist ϵ small enough so that $\sigma_{\min}(K_\ell(\epsilon)) \ge \gamma/2$ for every ℓ.

From (15),

$$A_\ell(\epsilon) = \Pi_\ell^T \tilde{F}_\ell^{-1} \Lambda_\ell K_\ell(\epsilon).$$

The matrix Π_ℓ is orthogonal, and $\tilde{F}_\ell$ is fixed and invertible. For every singular-value index i, standard product inequalities Horn & Johnson (2012) give

$$\sigma_{\min}(K_\ell(\epsilon))\sigma_i(\Lambda_\ell(\epsilon)) \le \sigma_i(\Lambda_\ell(\epsilon)K_\ell(\epsilon)) \le \sigma_{\max}(K_\ell(\epsilon))\,\sigma_i(\Lambda_\ell(\epsilon)).$$

Similarly, multiplication on the left by the fixed matrix $\tilde{F}_\ell^{-1}$ changes every singular value by at most fixed positive multiplicative constants:

$$\sigma_{\min}(\tilde{F}_\ell^{-1})\,\sigma_i(\Lambda_\ell(\epsilon))\sigma_{\min}(K_\ell(\epsilon)) \le \sigma_i(\tilde{F}_\ell^{-1}\Lambda_\ell K_\ell(\epsilon)) \le \sigma_{\max}(\tilde{F}_\ell^{-1})\,\sigma_i(\Lambda_\ell K_\ell).$$

Every singular value of A_ℓ is therefore bounded above and below by positive constants multiplied by the corresponding singular value of Λ_ℓ. The multiplicities follow directly from the characterization of Λ_ℓ.

5.5 Finite-difference factorization

For $n \ge 1$, define the $n \times n$ lower-triangular finite-difference matrix F_n

$$(F_n)_{j+1,k+1} = \begin{cases} (-1)^{j-k}\binom{j}{k}, & 0 \le k \le j, \\ 0, & k > j, \end{cases}$$

where $j, k \in \{0, \dots, n-1\}$ and the rows and columns of F_n are numbered $1, \dots, n$. The matrix F_n is invertible and

$$\det F_n = 1.$$

Lemma 2 (Exact finite-difference identity)*:* For every row vector b, every integer $m \ge 0$, and every $j \ge 0$,

$$\sum_{k=0}^{j} (-1)^{j-k} \binom{j}{k} b\mathbf{D}_\epsilon^{m+k} = \epsilon^j b\mathbf{D}_\epsilon^m H^j.$$

Proof. Factor $b\mathbf{D}_\epsilon^m$ on the left and use the binomial identity:

$$\sum_{k=0}^{j} (-1)^{j-k} \binom{j}{k} \mathbf{D}_\epsilon^k = (\mathbf{D}_\epsilon - I_c)^j = \epsilon^j H^j.$$

□

We show an example of how the finite difference matrix acts below.

Example 1*:* For $n = 3$, we have

$$F_3 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix}.$$

Thus, we have

$$\begin{aligned} F_3 \begin{bmatrix} a_0 \\ a_0\mathbf{D}_\epsilon \\ a_0\mathbf{D}_\epsilon^2 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_0\mathbf{D}_\epsilon \\ a_0\mathbf{D}_\epsilon^2 \end{bmatrix} \\ &= \begin{bmatrix} a_0 \\ \epsilon a_0\mathbf{H} \\ \epsilon^2 a_0\mathbf{H}^2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_0\mathbf{H} \\ a_0\mathbf{H}^2 \end{bmatrix}. \end{aligned}$$

5.6 Bounds on the singular values of $K^{(\ell)}(\mathbf{B}, \mathbf{H})$

Let $\mathbf{1}_{s>0}$ denote the indicator function of the positivity of s and x be a vector with $\|x\|_2 = 1$. We have

$$\left\| K^{(\ell)}(\mathbf{B}, \mathbf{H})x \right\|_2^2 = \sum_{j=0}^{\beta-1} \left\| \mathbf{B}\mathbf{H}^j x \right\|_2^2 + \mathbf{1}_{s>0} \left\| (\mathbf{B}\mathbf{H}^\beta)_{E_\ell} x \right\|_2^2 \leq \sigma_{\max}^2(\mathbf{B}) \left(\sum_{j=0}^{r_\star} \|\mathbf{H}\|^{2j} \right) \leq C_1^2 \sigma_{\max}^2(\mathbf{B}),$$

where the constant $C_1 = \sqrt{r_\star + 1}\max(1, \|\mathbf{H}\|_2^{r_\star})$.

The constant $\tilde{C}_1$ in the upper bound in Corollary 1 is greater than or equal to C_1 owing to the multiplicative factor of the inverse of the finite-difference matrix. Likewise, the constant c_0 appears because of the finite-difference matrix and the limiting $K^{(\ell)}(\mathbf{B}, \mathbf{H})$.

5.7 Relation between condition number of the $\mathbf{U}_i$'s and $\mathbf{M}_j$ matrices

In what follows $\mathbf{M}_j[i, :]$ denotes the i-th block-row of $\mathbf{M}_j$ (with c rows) for $i = 0, \ldots, N-1$. Based on the structure of $\mathbf{M}_j$, we note that for $\|x\|_2 = 1$

$$\begin{aligned} \|\mathbf{M}_j x\|_2^2 &= \sum_{i=0}^{N-1} \|\mathbf{M}_j[i, :]x\|_2^2 \\ &\leq N \max_{i=0,\ldots,N-1} \|\mathbf{U}_i\|_2^2 \\ &= N \max_{i=0,\ldots,N-1} \sigma_{\max}^2(\mathbf{U}_i). \end{aligned}$$

This means that $\sigma_{\max}(\mathbf{M}_j) \leq \sqrt{N}\sqrt{\max_{i=0,\ldots,N-1} \sigma_{\max}^2(\mathbf{U}_i)}$. Moreover, we also have $\sigma_{\max}(\mathbf{M}_j) \geq \max_k \sigma_{\max}(\mathbf{U}_k)$. This is because every $\mathbf{U}_k$ appears as a zero-padded block row within $\mathbf{M}_j$. From Corollary 1, we have that $\sigma_{\max}(\mathbf{U}_i) = \Theta(\epsilon^{-r_\star})$. Therefore, for fixed $\mathbf{B}$ and $\mathbf{H}$, we have $\sigma_{\max}(\mathbf{M}_j) = \Theta(\epsilon^{-r_\star})$.

Furthermore, for $\|x\|_2 = 1$, we have

$$\begin{aligned} \|\mathbf{M}_j x\|_2^2 &= \sum_{i=0}^{N-1} \|\mathbf{M}_j[i, :]x\|_2^2 \\ &= \sum_{i=0}^{N-1} \|\mathbf{U}_{j+i} x[N-1-i : N+c-2-i]\|_2^2 \\ &\geq \left(\sum_{i=0}^{N-1} \|x[N-1-i : N+c-2-i]\|_2^2 \right) \times \left(\min_{k=0,\ldots,N-1} \sigma_{\min}^2(\mathbf{U}_k) \right) \end{aligned}$$

$$\geq \|x\|_2^2 \times \left(\min_{k=0,\ldots,N-1} \sigma_{\min}^2(\mathbf{U}_k) \right)$$
$$= \min_{k=0,\ldots,N-1} \sigma_{\min}^2(\mathbf{U}_k).$$

Thus, we have $\sigma_{\min}(\mathbf{M}_j) \geq \sqrt{(\min_{k=0,\ldots,N-1} \sigma_{\min}^2(\mathbf{U}_k))}$

From Corollary 1, we have $\sigma_{\min}(\mathbf{U}_i) = \Theta(1)$. Thus, we can conclude that $\kappa_2(\mathbf{M}_j) = O(\sqrt{N}\epsilon^{-r_\star})$.

5.8 Details about numerical experiments

Details about fitting $(\mathbf{U}, \mathbf{R})$ parameters for CERAR: The code for running the optimization to determine $\mathbf{U}$ and $\mathbf{R}$ is available at this repository.[3] The code picks $\mathbf{B}$ to be a $N \times c$ matrix of i.i.d. $N(0,1)$ random variables. Following this, the matrix $\mathbf{H}$ is either chosen as a diagonal matrix whose diagonal entries are close to zero, or chosen in a deterministic manner using the constructed $\mathbf{B}$.

Details about All-Reduce Phase time measurement experiments: In our experiments, each GPU is controlled by one worker process. Before each measured reduction, the worker processes synchronize at a barrier: each waits until all workers have reached this point. After synchronization, each worker starts its timer with its GPU idle, executes the reduction, and stops the timer once the reduction has finished on its GPU. Before collecting these measurements, we perform 10 untimed warmup runs to initialize communication resources and, for our algorithm, compile the GPU functions. The measured time therefore includes communication and local computation, but excludes warmup runs, input generation, buffer initialization, synchronization before the timer starts, and result reporting. We then perform 100 runs measuring only the total reduction time (unprofiled runs) and 100 additional runs with additional timers measuring the communication and computation phases separately (profiled runs). For each unprofiled run, we take the maximum elapsed time across all the GPUs; Table 1 reports the median of these maximum times across runs.

In the profiled runs, we measure the time spent in our algorithm's communication and computation phases. For each GPU, we measure the elapsed time for sending and receiving vectors in each round, including waiting for these operations to complete. Overlapping sends and receives are timed together, rather than adding their individual durations. The communication time is the sum of these measurements across all rounds. Computation time includes encoding, updating received vectors and stored data, preparing outgoing vectors, and decoding, together with the memory accesses required by these operations. For the profiling results reported in Table 4, we use 1B FP32 entries per GPU on the A100 PCIe system and 24B FP32 entries per GPU on the H200 NVLink system. The table reports the mean times across the profiled runs, separately for each GPU. For NCCL, we report only the total profiled reduction time. Communication and computation times may not sum exactly to the total because the total also includes overhead outside the measured phases.

Using the same profiled runs and tensor sizes as Table 4, Table 5 further divides our algorithm's computation into encoding, outgoing-vector preparation, updates after reception, and decoding. Thus, the A100 PCIe breakdown corresponds to 1B FP32 entries per GPU, whereas the H200 breakdown corresponds to 24B FP32 entries per GPU. For the A100 PCIe configuration, outgoing-vector preparation covers the additions after reception in rounds 1–7 that prepare the sends for rounds 2–8. Updates after reception cover rounds 8–26: storing received vectors for decoding and, except in the final round, removing the returning local contribution and preparing the next send. For the H200 configuration, the corresponding round ranges are 1–3 for outgoing-vector preparation (preparing sends for rounds 2–4) and 4–14 for updates after reception. The four components sum to the computation time in Table 4, up to rounding.

The profiling results help explain the different outcomes. On A100 PCIe at 1B entries per GPU, our algorithm spends approximately 9.29–9.52 seconds per GPU on communication, while its computation takes only 29.1–33.6 ms. On H200 at 24B entries per GPU, our communication phase takes approximately 306.5–306.6 ms, which is less than NCCL's complete reduction time of approximately 392.0 ms. However, the additional computation takes approximately 215.7–215.9 ms, bringing our total reduction time to approximately 522.5–522.6 ms. These results demonstrate the benefit of our algorithm on the tested low-bandwidth PCIe system, while showing that its computation cost outweighs the communication savings on the tested high-bandwidth H200 system for our

[3] `https://anonymous.4open.science/r/CERAR--43C6/`

Table 4: Profiled reduction times in milliseconds for A100 PCIe (PCIe interconnect; $N = 8$, $c = 20$, $L = 26$, 1B FP32 entries per GPU) and H200 SXM (NVLink interconnect; $N = 4$, $c = 12$, $L = 14$, 24B FP32 entries per GPU). Each entry is the mean across profiled runs for the indicated GPU. Total is the elapsed time of the complete profiled reduction.

		NCCL	Our algorithm		
System / interconnect	GPU	Total	Communication	Computation	Total
A100 PCIe, 8 GPUs / PCIe	0	17,927.334	9,509.018	33.613	9,549.329
	1	17,924.756	9,517.594	32.108	9,553.995
	2	17,919.882	9,520.844	31.662	9,556.541
	3	17,923.538	9,519.077	32.479	9,556.751
	4	17,927.527	9,519.617	32.251	9,556.561
	5	17,927.654	9,357.713	29.141	9,387.553
	6	17,927.335	9,289.659	31.173	9,323.855
	7	17,927.416	9,478.487	29.150	9,508.829
H200 SXM, 4 GPUs / NVLink	0	392.158	306.509	215.889	522.592
	1	392.030	306.503	215.867	522.568
	2	391.943	306.459	215.835	522.496
	3	391.948	306.591	215.729	522.512

Table 5: Computation breakdown for our algorithm in milliseconds, using the same profiled runs as Table 4. The A100 PCIe results use a tensor of 1B FP32 entries per GPU, whereas the H200 SXM results use a tensor of 24B FP32 entries per GPU. Each entry is the mean for the indicated GPU.

System / interconnect	GPU	Encoding	Outgoing-vector preparation	Updates after reception	Decoding
A100 PCIe, 8 GPUs / PCIe	0	6.245	4.248	16.197	6.923
	1	6.186	4.120	14.907	6.896
	2	6.206	3.990	14.594	6.872
	3	6.255	4.394	14.955	6.876
	4	6.186	4.351	14.842	6.873
	5	6.229	3.283	12.752	6.877
	6	6.305	3.849	14.087	6.932
	7	6.222	3.508	12.544	6.876
H200 SXM, 4 GPUs / NVLink	0	51.390	16.441	93.414	54.645
	1	51.382	16.437	93.403	54.645
	2	51.366	16.438	93.405	54.626
	3	51.386	16.422	93.276	54.645

current implementation. As discussed in Section 3, optimizing our implementation and performing experiments on larger tensors could further improve performance.

Details about training experiments: Within each configuration, both methods (CERAR and NCCL) use the same model initialization, data ordering, and training hyperparameters. We use sequences of length 512, gradient checkpointing, and the AdamW optimizer. The model computation uses a mixture of BF16 and FP32, while model parameters, stored gradients, and gradient reduction use FP32. The learning rate is linearly increased to 3×10^{-4} and subsequently decayed using a cosine schedule.

For the PCIe experiment, the training set contains 120 million tokens, each GPU processes a batch of 32 sequences, and the learning-rate warmup lasts 200 steps. We train for 1,800 steps, processing 117.96 million tokens. For the SXM experiment, the training set contains approximately 474.89 million tokens, the per-GPU batch size is 64, and the warmup lasts 100 steps. We train for 7,630 steps, processing approximately 1 billion tokens.