

UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Optimizing Ring AllReduce for Sparse Data

Permalink

<https://escholarship.org/uc/item/84n223d4>

ISBN

9798270248345

Author

Arunachalam, Anshul

Publication Date

2026-12-24

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Optimizing Ring AllReduce for Sparse Data

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTERS OF SCIENCE

in Computer Science

by

Anshul Arunachalam

Thesis Committee:
Assistant Professor Sangeetha Abdu Jyothi, Chair
Professor Marco Levorato
Professor Nalini Venkatasubramanian

2025

TABLE OF CONTENTS

	Page
LIST OF FIGURES	iii
LIST OF TABLES	iv
ACKNOWLEDGMENTS	v
ABSTRACT OF THE THESIS	vi
1 Introduction	1
2 Background	3
2.1 What is AllReduce?	3
2.2 Cost Model	4
2.3 Common AllReduce Algorithms	5
2.4 Sparsity	6
2.5 Other AllReduce Algorithms for Sparse Data	7
3 Ring AllReduce for Sparse Data	9
3.1 Description	9
3.2 Correctness	13
3.3 Theoretical Performance	15
4 Evaluation	18
4.1 Experimental Set-Up	18
4.2 Results and Discussion	19
5 Conclusion	21
Bibliography	22

LIST OF FIGURES

	Page
2.1 An example AllReduce operation.	4
3.1 An example of skipping ahead.	10
3.2 An example of a correction message.	10
3.3 Actions at one node of the algorithm during the reduction phase.	11
4.1 The architecture of the ns-3 application.	19
4.2 The results of the ns-3 simulation.	20

LIST OF TABLES

	Page
2.1 Costs for well-known AllReduce algorithms.	6
4.1 Parameters used for the ns-3 simulation	18

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Jyothi, for all of her mentorship and advice. I have learned a lot.

I would also like to thank Professor Levorato and Professor Venkatasubramanian for being on my committee. I appreciated their feedback.

Finally, I would like to thank my family for all of their support.

ABSTRACT OF THE THESIS

Optimizing Ring AllReduce for Sparse Data

By

Anshul Arunachalam

Masters of Science in Computer Science

University of California, Irvine, 2025

Assistant Professor Sangeetha Abdu Jyothi, Chair

The distributed training of machine learning models via gradient descent is generally conducted by iteratively computing the local gradients of a loss function and aggregating them across all processors. Communicating these gradients during aggregation is often a major cost but sparsification techniques can greatly improve efficiency. One such technique, Top- k gradient compression, ensures that only the k largest components of each local gradient are sent. However, effectively scaling this method can be challenging. The standard ring AllReduce algorithm, which is frequently used to aggregate dense gradients, lacks a counterpart that is optimized for sparse data. Notably, ring algorithms are contention-free, which generally make them easier to scale than other collective communication algorithms. Thus, in practice, the ring AllGather algorithm, which can be trivially adapted for sparse data, may be used instead, even though its bandwidth costs are proportional to the number of utilized processors (unlike ring AllReduce). To provide a more scalable contention-free alternative, we present a variant of ring AllReduce that has been better optimized for sparse data. We compare it to the standard dense ring AllReduce and ring AllGather algorithms, and we evaluate it empirically using gradients sampled from fine-tuning Llama 2 7b.

Chapter 1

Introduction

Recent transformer-based models utilize a much greater number of parameters than previous generations of machine learning models. For example, Meta’s largest Llama 3 model has 405 billion parameters [4]. A vast number of techniques are used to efficiently train these models, including using distributed systems. When conducting distributed training, the dataset is partitioned and each compute node (e.g., a GPU) trains on a partition for a period of time. Then, the nodes’ computed gradient updates are aggregated and redistributed. Each node will eventually receive the same final gradient update and, when it does, it can continue training.

There are two commonly used methods used to aggregate gradient updates, parameter servers [7] and AllReduce. Parameter servers receive updates from all compute nodes, aggregate them locally, and broadcast the aggregated result back to the compute nodes. One issue with this is that it can be difficult to scale well. AllReduce algorithms, which are generally more uniform and balanced, tend to scale better.

Since communication costs are a large proportion of overall training costs, a variety of training techniques have been developed to optimize for lower communication volumes. One of

these techniques is sparsification. Sparsification reduces the communication volume and, in many cases, does so without meaningfully reducing inference accuracy. (There can also be other benefits to sparsification, like the tendency of sparse models to generalize better [5].)

However, maintaining sparsity levels can be challenging during gradient aggregation. In order to address this in a scalable manner, we optimize the standard ring AllReduce algorithm for sparse data.

Chapter 2

Background

2.1 What is AllReduce?

In many distributed settings, nodes (e.g., servers with CPUs or GPUs) need to coordinate computation. This can require communicating data between nodes in a highly-choreographed, optimized manner. Collective communication operations [2] fulfill this need. Common operations include broadcast (where one node sends data to all other nodes) and AllGather (where all nodes send data to all other nodes).

Reduction collective communication operations, such as AllReduce, are also important. Given an associative (and, usually, commutative) operator, AllReduce reduces (applies the operator on) data from all nodes and sends the result to every node [9]. (See Figure 2.1 for an example.) One example of this collective communication operation is in Horovod [11], which utilizes AllReduce to improve the scaling of model training. In this case, the reduction operator is tensor addition, which aggregates gradient updates. Note that collective communication operations are used beyond machine learning applications (including in, e.g., scientific computing).

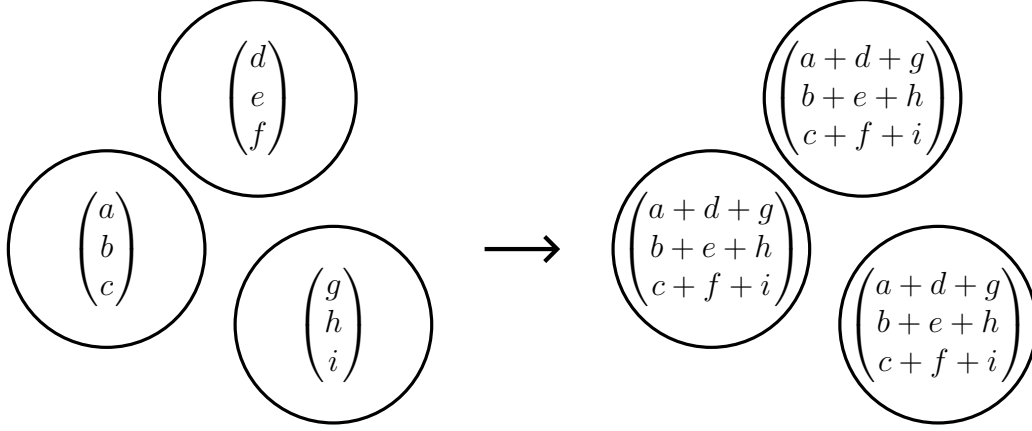


Figure 2.1: An example AllReduce operation.

2.2 Cost Model

The standard linear cost model (the Hockney model) [2] for collective communication operations is: $\mathcal{C}_p(n) = \alpha + \beta n$, where p is the number of processors, n is the size of the data, α is the per-message cost (the latency) and β is the per-byte cost (the inverse of the bandwidth). (Note that we'll refer to it as the “bandwidth cost” from here.)

There may also be a factor δ to account for congestion: $\mathcal{C}_p(n) = \alpha + \delta\beta n$. We'll generally omit it since the ring algorithms we'll discuss are contention-free. (Of course, it's generally nontrivial to account for congestion, especially at scale, so any benefits of a contention-free algorithm may be difficult to quantify sufficiently precisely in this model.) Additionally, we assume that wormhole routing and (full duplex) bidirectional links are used. (This also allows us to avoid additional communication cost terms [3, 10] in our model.)

Finally, for operations involving reductions, there is a per-byte computation cost term: γn . Thus, our final cost model is $\mathcal{C}_p(n) = \alpha + \beta n + \gamma n$.

2.2.1 Lower Bounds

Informally, the theoretical lower bounds on *total* messages and bandwidth for AllReduce in *single-ported* indirect networks are $2(p - 1)$ and $2n\frac{p-1}{p}$, respectively [9]. The ring and recursive-doubling algorithms are optimal in the bandwidth term, as illustrated in Table 2.1.

2.3 Common AllReduce Algorithms

2.3.1 Binary Trees

The binary tree algorithm [10] is a well-known method of implementing AllReduce. In each round, data is exchanged between pairs of nodes and is reduced. After any given round, only half the processes are used in future rounds of reduction (which makes this algorithm suboptimal). Once the final reduction occurs at the root of the tree, the result is broadcast to all other nodes.

2.3.2 Recursive Doubling (Rabenseifner’s Algorithm)

Recursive doubling (also known as Rabenseifner’s algorithm) [10] utilizes an exchange of data between nodes, similar to the binary tree algorithm. However, in the first round, only half the data is exchanged (with different halves going to each node) and reduced. In each subsequent round, the data exchanged is further halved. In addition, to further improve performance, nodes closer together exchange the largest amounts of data.

One issue with recursive doubling is that it can be sensitive to the number of processes used. The lowest cost version of the algorithm is only utilized if the number of processes is a power of 2. (However, utilization of binary blocks, where processes are partitioned into groups of

Algorithm	Latency	Bandwidth	Computation	Contention-Free?
Binary Tree [10]	$2 \log p$	$2n \log p$	$n \log p$	No
Recursive Doubling ¹ [10]	$2 \log p$	$2n \frac{p-1}{p}$	$n \frac{p-1}{p}$	No
Ring [9]	$2(p-1)$	$2n \frac{p-1}{p}$	$n \frac{p-1}{p}$	Yes

¹For p is a power of 2.

Table 2.1: Costs for well-known AllReduce algorithms.

different powers of 2, can help improve performance when the total number of nodes is not a power of 2.)

2.3.3 Rings

Ring algorithms [3, 9] tend to scale better than other collective communication algorithms, such as recursive-doubling algorithms, since they are contention-free (and, hence, have no congestion). In ring AllReduce, also known as bucket AllReduce, each node’s data is partitioned into blocks and, in each round, one block (of size $\frac{n}{p}$) is sent from each node to the next node in the ring in a synchronous manner. One drawback to this, however, is that latencies tend to be higher, particularly if Hamiltonian rings are used.

2.4 Sparsity

Sparsity is an important aspect of both model training and inference [5]. One specific benefit of sparsity is that utilizing sparsity can enable efficiently training large models for little cost to inference.

2.4.1 Top- k Sparsification

Top- k sparsification [1] utilizes the fact that communication accounts for a substantial share of costs when training large models. The idea here is that instead of sending all gradient updates, we just send the k largest components of the update and keep the rest (to be added back locally in the next round). Note that a variant of this method, Random- k , does not converge well in practice [12].

2.4.2 Sparse Tensors

Sparse tensors can be characterized in several ways but the method we use here is to consider the fraction of nonzero values. One important aspect of this is that the sum of two κ -sparse matrices may be 2κ -sparse (which would happen if there was no overlap of components). This means that efficiency gains from sparse tensors can be easily lost if suboptimal methods (e.g., suboptimal aggregation methods) are used.

2.5 Other AllReduce Algorithms for Sparse Data

2.5.1 $\mathcal{O}(k)$ -Top k

$\mathcal{O}(k)$ -Top k [8] is a recursive doubling-based aggregation technique for sparse data. The algorithm utilizes an approximation of the global Top- k components (across multiple nodes). It also uses load balancing to ensure that nonzero values are more evenly partitioned among workers. Its costs are linear in terms of k so it is asymptotically optimal in the bandwidth component. (In particular, bandwidth costs are between $2k\frac{p-1}{p}\beta$ and $6k\frac{p-1}{p}\beta$, where p is the number of compute nodes.) However, since it optimizes recursive doubling, congestion may

hinder scalability. Furthermore, the utilization of the approximation could reduce broader applicability, especially because it utilizes periodic recalculations (the scheduling of these recalculations is a hyperparameter that may need to be tuned).

2.5.2 AllGather

Some existing frameworks that implement sparse collective communication tend to use AllGather algorithms to implement sparse AllReduce. One reason is that it is more broadly applicable than $\mathcal{O}(k)$ -Top k , which is a fairly heavy-duty, opinionated algorithm.

Ring AllGather can ensure that the sparsity of the data transmission is maintained throughout the duration of the operation. Horovod [11], for example, utilizes ring AllGather. However, although it is contention-free, it does not scale as well since the bandwidth cost increases with the number of compute nodes (unlike dense ring AllReduce).

Chapter 3

Ring AllReduce for Sparse Data

3.1 Description

The main idea is to turn AllReduce into a sort of greedy algorithm. We start with the regular ring AllReduce set up, except each node starts with a set of sparse data (the particular format is not too relevant as long as it is efficient to track ordering). In essence, in any particular round, each node i can skip ahead if it does not have enough values to fill up its send buffer. Figure 3.1 illustrates this step.

If another node j sends indices to node i that node i already sent (when it skipped ahead), then node i simply passes along the new values without reducing it. Figure 3.2 illustrates this correction message. (Node i would have already sent its own message so this ensures the accuracy of the final output.) Note that the loss in efficiency from this correction message will, in the worst case, only set the algorithm back to where the node would have been in the standard AllReduce.

No matter what, if a node receives a message in its “final zone,” then the node reduces and

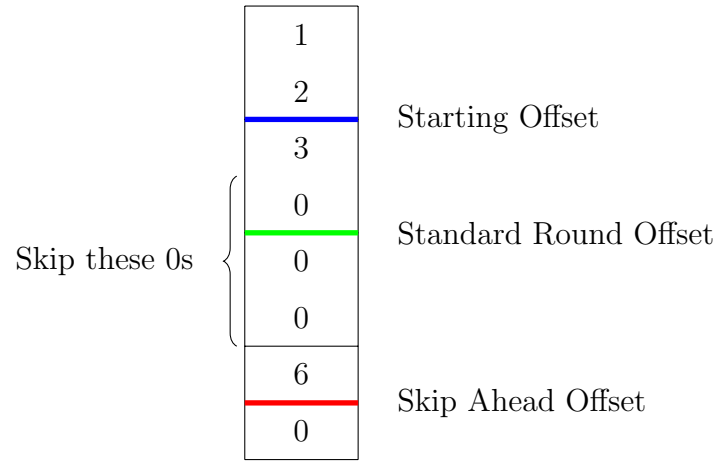


Figure 3.1: An example of skipping ahead.

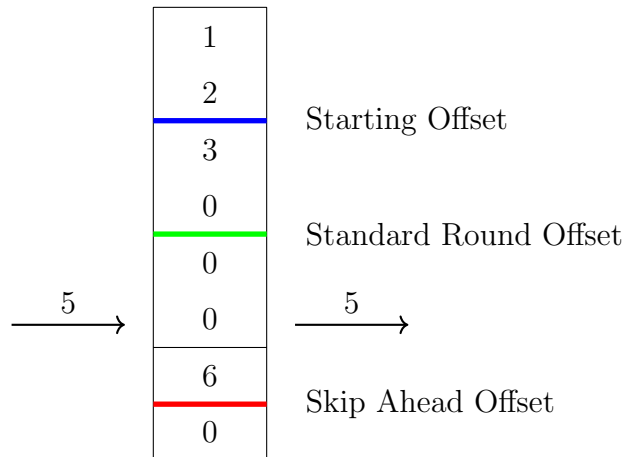


Figure 3.2: An example of a correction message.

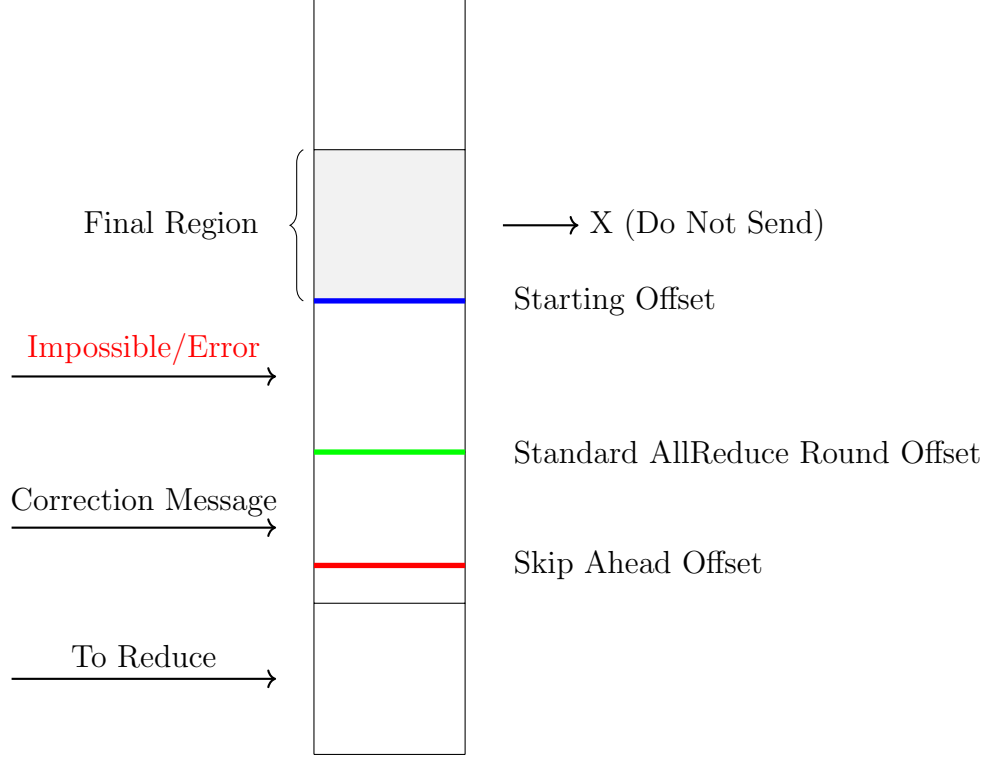


Figure 3.3: Actions at one node of the algorithm during the reduction phase.

keeps it. The final zone is essentially the region of the data that will eventually be broadcast. Figure 3.3 illustrates and summarizes several aspects of the reduction phase of the algorithm. Note that the orientation of the torus is down.

After the reduce-scatter phase, a broadcast (AllGather) phase is done just like the one for the standard ring AllReduce. (This is done in $p - 1$ rounds as well.) The only difference is that the sparse broadcast phase only broadcasts nonzero values. The more precise pseudocode of the algorithm is presented in Algorithm 1.

Algorithm 1: SPARSERINGALLREDUCE

```
// Let p be the number of compute nodes.
1 function SparseRingAllReduce(rank, dataBuffer, p)
  // totalData denotes all data (including zeros)
2  chunkLength  $\leftarrow \lceil |totalData| / p \rceil$ 
  // This enables the calculation of the final region (no values are
  // sent from the region)
3  noGo  $\leftarrow ((rank - 1) \bmod p)$ 
  // Get a chunk's worth of non-zero values (until the final region
  // is reached)
4  initialSendBuffer, nextStartOffset
    $\leftarrow$  getInitialSendData(chunkLength, noGo)
5  sendData(initialSendBuffer,  $(rank + 1) \bmod p$ )
6  for (p - 1 rounds) do
7    newDataBuffer  $\leftarrow$  receiveData( $(rank - 1) \bmod p$ )
    // correctionBuffer and/or toReduceBuffer may be empty
8    correctionBuffer, toReduceBuffer
      $\leftarrow$  processBuffer(newDataBuffer, nextStartOffset)
9    reduce(dataBuffer, toReduceBuffer)
10   remainingChunkLength  $\leftarrow$  chunkLength - |correctionBuffer|
11   toSendBuffer, nextStartOffset  $\leftarrow$ 
     getNextDataValues(remainingChunkLength, nextStartOffset, noGo)
12   if (round  $\neq$  p - 1) then
13     sendBuffer  $\leftarrow$  concat(correctionBuffer, toSendBuffer)
14     sendData(sendBuffer,  $(rank + 1) \bmod p$ )
   // The broadcast (i.e. AllGather) phase is the same as the one in
   // the standard ring AllReduce algorithm (except it only broadcasts
   // nonzero values)
15 StandardBroadcastPhaseWithNonzero()
```

3.2 Correctness

To show the correctness of the algorithm, we'll prove the following theorem.

Theorem 1. *Let p compute nodes $\mathcal{P} = \{P_0, \dots, P_{p-1}\}$ be connected in a ring such that $P_i \rightarrow P_{(i+1) \bmod p}$. Let there also be data $\mathcal{D} = (d_0, \dots, d_{p-1})$, where $|d_i| = n$, such that each node P_i has partition d_i . Given a commutative (and associative) reduction operator $\oplus$, and an identity element $\mathbf{0}$, $\text{SPARSERINGALLREDUCE}_{\oplus}(\mathcal{P}, \mathcal{D})$ correctly outputs the final data $\mathcal{D}^* = (d_0^*, \dots, d_{p-1}^*)$, where $d_i^* = \bigoplus_{j=0}^{p-1} d_j$ for all i .*

Proof. We reduce the correctness of $\text{SPARSERINGALLREDUCE}$ to the correctness of $\text{STANDARDRINGALLREDUCE}$.

The standard algorithm has two distinct phases: the reduction phase and the broadcast phase. Let $\phi(i) : \mathbb{Z}_p \rightarrow \mathcal{P}(\{1, 2, \dots, n\})$ be a function that assigns a unique part of a partition of indices to each node i . Let $d_{j, \phi(i)}$ denote a region of node j 's initial data, as selected by $\phi(i)$. At the start of the broadcast phase, each node P_i has a region of data $d_{i, \phi(i)}^*$ such that $d_{i, \phi(i)}^* = \bigoplus_{j=0}^{p-1} d_{j, \phi(i)}$ (i.e., each region is uniquely fully reduced). Note that the state transitions of the broadcast phase of $\text{STANDARDRINGALLREDUCE}$ are independent of the distribution of the data. Then, since $d_i^* = \bigoplus_{i=0}^{p-1} d_i = \bigcup_{j=0}^{p-1} d_{j, \phi(j)}^*$, the regions $d_{i, \phi(i)}^*$ are the only inputs to the broadcast phase that have any impact on the output.

$\text{SPARSERINGALLREDUCE}$ also has reduction and broadcast phases, and it has a very similar broadcast phase to $\text{STANDARDRINGALLREDUCE}$. For each partition $\phi(i)$, let $\phi_{nz}(i)$ be the set of the nonzero values of the partition. Consider an invertible mapping $g : \bigcup_i (\{i\} \times \phi_{nz}(i)) \rightarrow \{1, 2, \dots, n\}$, where each index of a nonzero value in each partition is assigned a unique virtual index. In other words, the only indices being considered are nonzero indices and, at the start of the broadcast phase, each node has a region $d_{i, g(i, \phi_{nz}(i))}^*$. After mapping, the broadcast phase is exactly the same (we apply g^{-1} to get the original indices at the end).

Thus, if the reduction phase of SPARSERINGALLREDUCE correctly outputs the regions $d_{i,g(i,\phi_{nz}(i))}^*$, the whole algorithm is correct.

We'll show that this is the case. Without loss of generality, assume all partitions are the same size b . First, note that the current offset counter cannot repeat an offset. Furthermore, after the first r rounds, the current offset counter must have been incremented by at least rb . To see this, consider a hypothetical dense version of the data. Note that by default, the counter increments by b . For every zero index encountered, it increments by one additional position (until the final zone is reached). (This also applies to zeros that appear in the new additional positions.) Every correction index received corresponds to a unique additional increment caused by a zero. For example, if two correction messages with the same index need to be sent by a node, each correction message corresponds to two different unique increments. Thus, at the end of $p - 1$ rounds, the current offset counter will be at the final zone (so all local indices would be sent by the end of the reduction phase).

Consider an arbitrary message $m = (i_1, i_2, \dots, i_m)$ (where i denotes an index) initiated by an arbitrary node P_a . Consider an arbitrary index in the message i_* . We will show that it must reach its final destination. Consider some non-final node P_b that receives the message. Its current offset counter has either passed i_* or it hasn't. If it has, we know that i_* will be sent in the next message (since it is a correction index). (Clearly, all the current correction indices will fit into the next message since they must have arrived in one message in the first place.) If it hasn't, we know it will eventually be sent since the current offset must eventually pass i_* on its way to its final zone. In the worst case, i_* will be sent as a correction index for every other node after the current node. However, i_* will still reach its final node since we know that there must be enough round left. (Note that otherwise, the current offset would have passed i_* .)

Finally, note that each node's index will only be reduced with all other data once so the actual value is also correct. □

3.3 Theoretical Performance

3.3.1 Latency, Computation, and Bandwidth Costs

Latency and Computation Costs

There are $p - 1$ rounds for the reduce-scatter (i.e., the reduction phase) and $p - 1$ rounds for the AllGather (i.e., the broadcast phase). Hence, just like with the standard algorithm, the latency cost of SPARSERINGALLREDUCE is $2(p - 1)$.

Let κ be the fraction of the data that is nonzero. The computation cost of the algorithm is $\frac{\kappa n}{p}(p - 1)$ since we only reduce each node's nonzero components of data once. (Note that, when a correction is sent, we only pass along the message so we do not incur any additional computation costs.)

Bandwidth Cost

The bandwidth cost is dependent on the distribution of the data. However, we can compare it with the alternative contention-free options.

3.3.2 Comparison to Standard Ring AllReduce

Consider the timeline of both the regular ring AllReduce and the sparse ring AllReduce. Recall the proof of correctness of the sparse ring AllReduce algorithm. In each round of the sparse AllReduce, when a node skips ahead, it gains extra credits as compared to the regular AllReduce. Thus, when it has to stall due to corrections, it simply loses some credits. It can never lose more than it gained since the messages it receives are bounded by the regular AllReduce's schedule. (This can also be seen in the distance it takes for a message to reach

a node.) Hence, the amount of data sent by sparse ring AllReduce can be no worse than the standard ring AllReduce.

However, no correction messages are sent for indices for which there are no nodes that have nonzero values. Thus, the amount of data sent by sparse ring AllReduce can be no worse than the standard ring AllReduce on simply the final aggregated indices. (Note that these indices would not be known ahead of time.)

(Also, note that the latency costs are the same and the computation costs are better. Thus, the total cost of sparse AllReduce can be no worse than that of dense AllReduce.)

3.3.3 Comparison to Standard Ring AllGather

Let κ be the fraction of the data that is nonzero. Note that standard AllReduce on sparse data (where the data is stored in a dense format) outperforms AllGather as the number of compute nodes increases:

$$\kappa > \frac{2}{p} \implies p(p-1)(\kappa n) > p \left(2(p-1) \frac{n}{p} \right)$$

Since we know that the sparse ring AllReduce is no worse than the standard ring AllReduce, we know that when $\kappa > \frac{2}{p}$, sparse ring AllReduce is better than the standard ring AllGather.

(The latency cost of the standard ring AllGather algorithm is $p-1$ since there's just $p-1$ rounds of broadcasting. This is half the latency cost of SPARSERINGALLREDUCE. The computation cost is $\kappa n(p-1)$ since, in each round, data of size κn is sent and reduced.)

3.3.4 Additional Remarks

The number of rounds utilized for the reduce-scatter phase of the sparse algorithm is simply the length of the longest path taken by any message. For example, suppose the orientation of the ring is $i \rightarrow (i+1) \bmod p$. If no node i needs to send data to node $(i-1) \bmod p$, then the last round is unnecessary. However, this is unlikely to be the case for most workloads (including machine learning tasks), so allowing for the possibility of exiting the algorithm early would not be worth the coordination costs.

Even for workloads that are sparse and *highly* asymmetrical, no node in the ring would know if they are finished early since, at any time, they may receive a message that was initiated at a faraway node. A simple way to resolve this would be to do a preliminary AllReduce. Each node would compute the length of the longest path required by its local data. Then, the reduction operator utilized over the lengths would be $\oplus := \max(\cdot)$. (Since the messages are small, it would almost certainly be more efficient to utilize a latency-optimal AllReduce algorithm like, e.g., recursive doubling.)

Chapter 4

Evaluation

We evaluate our algorithm along with the primary contention-free alternatives for AllReduce on sparse data, which are ring AllGather and standard (dense) ring AllReduce.

4.1 Experimental Set-Up

To evaluate the algorithms, we use a packet-level simulator, ns-3. We use parameters that approximately match the NDR Infiniband’s parameters, as can be seen in Table 4.1.

The architecture of the ns-3 application used to simulate the communications is simple. Note that sending a notification packet backward is a common design pattern that is used

Parameter	Value
Data Rate	400 GB/s
Delay	1 μ s
MTU	4096

Table 4.1: Parameters used for the ns-3 simulation

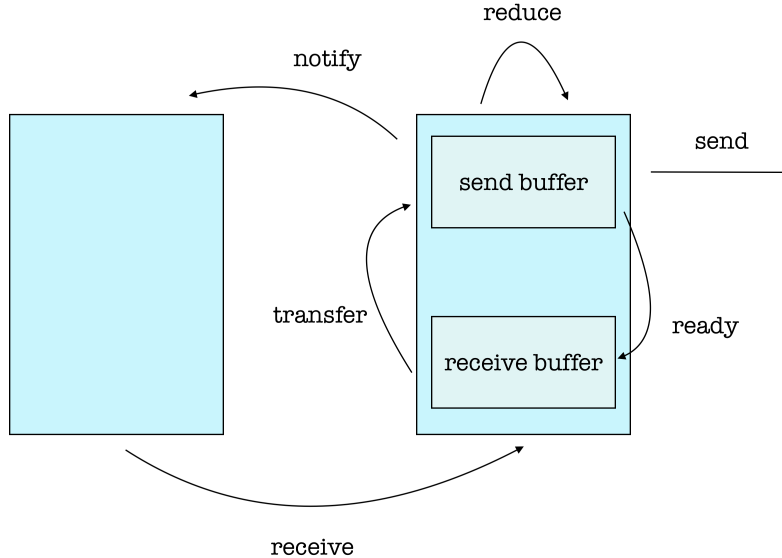


Figure 4.1: The architecture of the ns-3 application.

in production libraries, such as gloo.

Also, note that ns-3 does not model computation costs. In practice, we precompute the reductions at each stage to improve the simulation’s efficiency.

In order to accurately model large gradient updates, we sample gradient updates, with enforced layer-level sparsity of 65% via Top- k , from fine-tuning Llama 2 7b [13] on the OpenAssistant Conversations dataset [6], which was released after Llama 2 7b.

4.2 Results and Discussion

We simulate both a 5 node ring and a 10 node ring, and compare how both ring AllGather and sparse ring AllReduce scale. Figure 4.2 shows the results. We see that as the number of nodes increase, the performance of the sparse ring AllReduce algorithm improves (as compared to ring AllGather). This is likely due to the higher overhead of sparse ring AllReduce (when utilizing a smaller number of nodes).

Sparse Ring AllReduce vs Ring AllGather

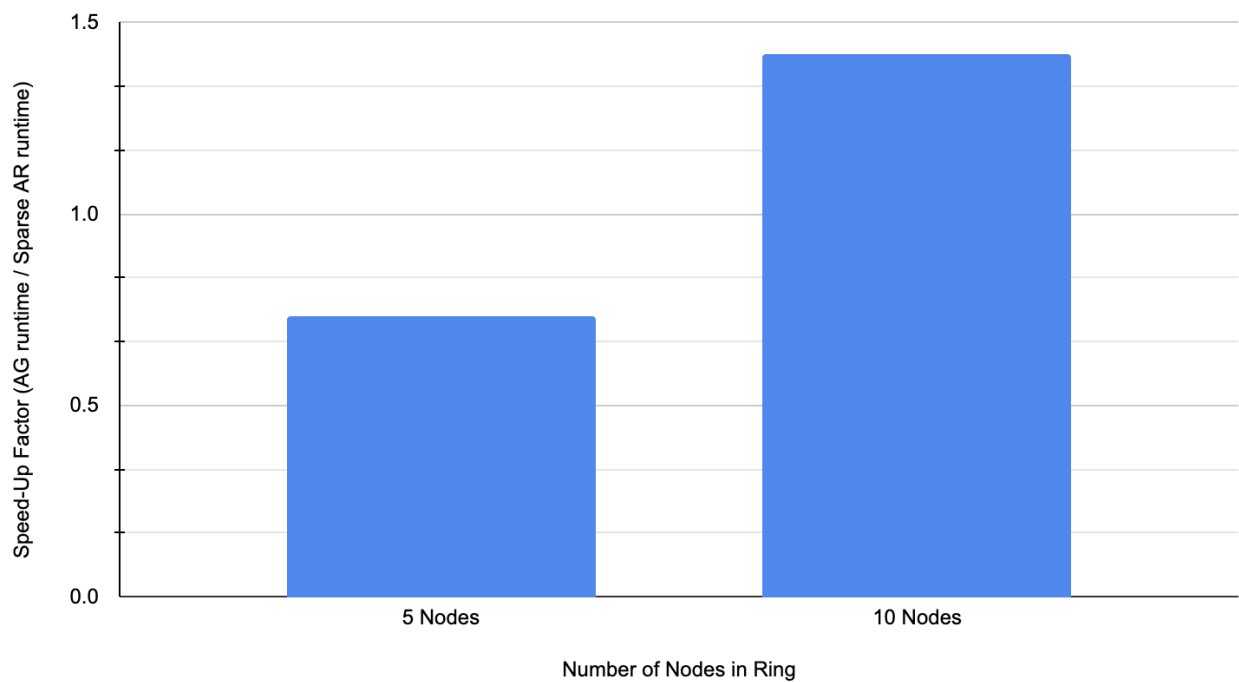


Figure 4.2: The results of the ns-3 simulation.

Chapter 5

Conclusion

Here, we have presented a variant of the standard ring AllReduce algorithm that has been optimized for sparse data. The algorithm shows both theoretical and empirical improvements over the primary contention-free alternative for sparse data, ring AllGather, particularly as the number of compute nodes increases.

Future potential research directions could include developing a variant of the sparse AllReduce algorithm that computes a global approximate Top- k . This may improve efficiency even more as it could reduce the bandwidth costs during the broadcast phase of the algorithm.

Bibliography

- [1] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.
- [2] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.
- [3] A. Faraj, P. Patarasuk, and X. Yuan. Bandwidth efficient all-to-all broadcast on switched clusters. *International Journal of Parallel Programming*, 36(4):426–453, 2008.
- [4] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yeary, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen,

S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Col-
lot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou,
T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta,
V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic,
W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan,
X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song,
Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh,
A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand,
A. Menon, A. Sharma, A. Boesenberg, A. Baeviski, A. Feinstein, A. Kallet, A. Sangani,
A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan,
A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel,
A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang,
B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti,
B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia,
C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichten-
hofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine,
D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling,
E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Ar-
caute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel,
F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee,
G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan,
H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren,
H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche,
I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus,
J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cum-
mings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H.
U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena,
K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva,
L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani,
M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov,
M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vy-
atskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat,
M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal,
N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz,
O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner,
P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang,
R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy,
R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J.
Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan,
S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C.
Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe,
S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta,
S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez,
T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou,
T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla,

- V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma. The llama 3 herd of models, 2024.
- [5] T. Hoefer, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.
- [6] A. Köpf, Y. Kilcher, D. Von Rütte, S. Anagnostidis, Z. R. Tam, K. Stevens, A. Barhoum, D. Nguyen, O. Stanley, R. Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in neural information processing systems*, 36:47669–47681, 2023.
- [7] M. Li, D. G. Andersen, A. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. *Advances in neural information processing systems*, 27, 2014.
- [8] S. Li and T. Hoefer. Near-optimal sparse allreduce for distributed deep learning. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 135–149, 2022.
- [9] P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.
- [10] R. Rabenseifner. Optimization of collective reduction operations. In *International Conference on Computational Science*, pages 1–9. Springer, 2004.
- [11] A. Sergeev and M. Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.
- [12] S. Shi, X. Chu, K. C. Cheung, and S. See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.
- [13] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.