All-Reducible signSGD

Naman Jain, Basava Kolagani

August 10, 2022

1 Introduction

Distributed Data Parallel training[1] is a well known solution to the problem of large scale Machine Learning training. A significant portion of training time is spent in communication between machines for the purpose of gradient synchronization. Current research hints at various ways in which we can optimize communication between machines - Gradient compression[2], topK [3], reduced precision[4], ternGrad [5] - to name a few. In this paper, we propose an All-Reducible implementation for signSGD gradient synchronization and show that it achieves 2-4x speed up when compared to the State of the Art 1-bit Adam [6] implementation.

2 Background

2.1 Distributed data parallelism

Using distributed training on multiple GPU/CPU, we can reduce the training time manifold. Distributed data parallelism(DDP) is one of the most common distributed training methods. In DDP, all the model parameters are replicated across all the nodes, and all the nodes access a different partition of the data. Each node computes the local gradient, and all local gradients are aggregated to update the model parameters across all the nodes. Gradient aggregation can be achieved by parameter server or all reduce topology.

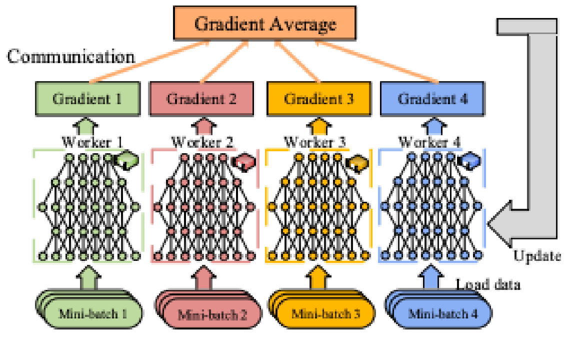


Figure 1: Distributed data parallelism across 4 nodes

2.2 Gradient aggregation techniques

As mentioned above, we can do gradient aggregation with parameter server or all reduce topology. The issue of using a parameter server for gradient compression is that the communication cost increases linearly with the number of GPU/CPU and the parameter server also becomes the bottleneck.

To solve this problem, nowadays, all reduce topology is generally used. In all reduce, the communication cost remains constant irrespective of the number of CPU/GPU. All reduce approach is bottleneck-ed by the slowest link between GPU/CPU. There are multiple ways of using all reduce topology like ring all reduce, tree all reduce, recursive doubling all reduce, etc.

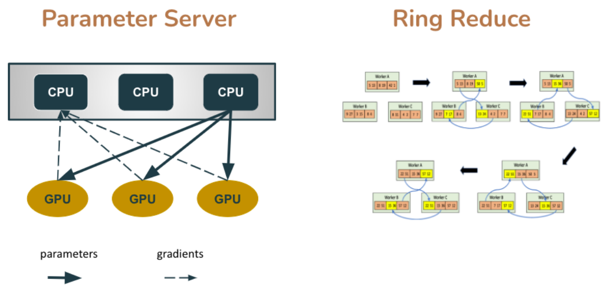


Figure 2: Parameter server and Ring all reduce for gradient aggregation

2.3 signSGD

In DDP, since we need to aggregate all the local gradients from the nodes, communication cost becomes a bottleneck. To reduce the communication cost, gradient compression is a way that has been proven to be very effective. signSGD[7] is a 1-bit gradient compression technique in which we compress the gradient based on the sign of the gradient. If the gradient is negative, it is converted to *-1*, and if the gradient is positive, it is converted to *+1*. After the compression, all the nodes send their 1-bit compressed gradient to a parameter server. Parameter server calculates the majority of all the gradients and sends back the result as 1-bit back to all the nodes, and then all the nodes update their model parameter accordingly.

One major issue with signSGD is that it is not all reducible when being performed without introducing additional data i.e using just 1 bit at each phase. As shown in the Figure:3, if we have a two-node cluster with four *+1* local gradients in the first node and three *-1* one *+1* in the second node. Then if we take the majority, the result will be 0, but it should have been *+1*. Hence we can say that the majority operator is not all reducible. Since the parameter server's communication cost increases with the number of GPU/CPU, signSGD is not scalable. To scale signSGD, we need to have an all-reducible compatible signSGD.

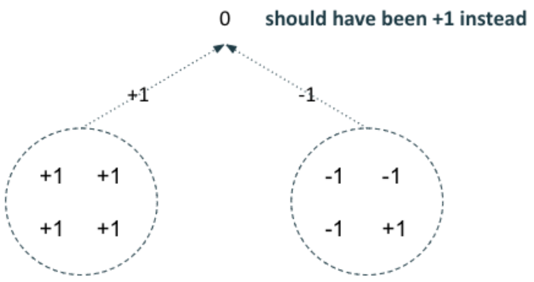


Figure 3: All reduce with majority operator

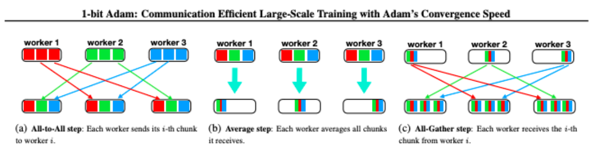
2.4 1-bit Adam

1-bit Adam[6] introduces an efficient and straightforward all-reducible method for 1-bit compressed gradients, which is helpful in large-scale training. It's a three-step process:

- All to All step: In this step, each worker sends its ith chunk to worker i.
- Average step: All the chunks that each worker has recieved are averaged.
- All Gather step: In this step, each worker receives ith chunk from worker i.

3 Design

We make use of the existing Ring reduce algorithm to compute the majority. In particular, each node reads information from its predecessor in the ring, updates it by adding the local information and proceeds forward with the next iteration. We maintain a cumulative count


1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed

(a) **All-to-All step**: Each worker sends its i-th chunk to worker i.
(b) **Average step**: Each worker averages all chunks it receives.
(c) **All-Gather step**: Each worker receives the i-th chunk from worker i.

Figure 4: all reducible implementation by 1-bit adam on 3 workers

of the *+1s* seen up until that point in the ring reduction. Once we have this cumulative count over all worker nodes, we can determine the majority locally by a simple boolean comparison. If the scalar value is greater than half of the participating workers, then the gradient takes the positive direction, else it takes the negative direction. For a given setup of N worker nodes, the cumulative count of *+1s* for any co-ordinate in the gradient tensor is capped by N. As a result, we need at most *log(N)* bits in each iteration to retain the cumulative count information. As a further optimization, we use only $\lceil log(i+1) \rceil$ bits in the ith iteration. This works because the value being forwarded in the ith iteration is the count across i machines and is capped by i. To represent this information, we need only $\lceil log(i+1) \rceil$ bits.
In case of N workers in a ring reduction process, $\lceil log(i+1) \rceil$ bits are forwarded for each scalar in the first N-1 iterations. Post this, every worker has the total count of *+1s* across the ring for its corresponding chunk. It determines the majority information by comparing the count with N/2. Once the majority is determined, the second half of ring reduction process kicks in, where a server receives the processed information for the other chunks. As a optimization, we stream back only the majority bit (1 for +1, 0 for -1) instead of streaming back the cumulative count.
If each worker has a tensor of size *O(K)* at the beginning of the ring reduce process, the total data transferred by each worker node throughout the process is of the order

$$(N-1) * \frac{K}{N} * \sum_{i=1}^{N-1} \lceil log(i+1) \rceil + (N-1)$$

Hence the total data transferred at each node is of the order *O(K * log(N))*, where K is the size of the tensor on each machine and N is the number of worker nodes in the ring.
In order to transfer $\lceil log(i+1) \rceil$ bits in the ith iteration instead of transferring the whole integer, we make use of bitwise AND operations. Given a tensor of integers, we perform the bitwise AND operation as shown below.

To extract j bits, the matrix B should have j elements with incremental powers of 2. The above operations extracts 4 bits from the integers and should have been invoked as part of 8-15 iterations ($4 = \lceil log(8+1) \rceil =$

$$\begin{bmatrix}3\\6\\10\\13\end{bmatrix} \;\&\; \overset{B}{\begin{bmatrix}8&4&2&1\end{bmatrix}} = \begin{bmatrix}0&0&1&1\\0&1&1&0\\1&0&1&0\\1&1&0&1\end{bmatrix} \xrightarrow{\text{flatten}} \begin{bmatrix}0&0&1&1&0\ldots&0&1&1&0&1\end{bmatrix}$$

Figure 5: Extracting required # bits from integers using bitwise AND operator

$\lceil log(15+1) \rceil$)
On receiving this tensor, a worker node converts it back to actual integers by using matrix multiplication. All this is possible because the number of bits being sent per integer in each iteration is deterministic and hence the receiver is able to map back to the correct information. The following matrix multiplication yields the actual information that should have been forwarded from the predecessor.

4 Evaluation

We evaluated both, our algorithm and the 1-bit Adam implementation on different network topologies. We ran all the experiments on ClouldLab CPU machines - c220g2. Each machine had 40 cores and 160GB RAM available. Since the existing DeepLearning frameworks like Pytorch provide support for byte addressable memory and not bit addressable, we make use of boolean data primitive to evaluate out implementation. In specific, we maintain one boolean per bit though a boolean actually consists of 8 bits i.e 1 byte. To keep the comparison with 1-bit Adam meaningful, we implement it using boolean data type, where we again maintain one boolean variable for each signed bit.
Since the Ring reduce implementation has extra bitwise AND and matrix multiplication operations, there is some additional overhead introduced. Evaluating the setup on GPU workers would have probably yielded greater speedups since GPUs are highly optimized for such tensor operations. To get an estimate of the probable speedup when evaluated on GPU workers, we voided out the additional compute and recorded the time taken by the Ring reduce implementation in each experimental setup. This is recorded under the Ring-void column in the below experimental results. The Probable-speedup corresponds to the Ring-void column.

4.1 Chain Topology

We configured a Chain like topology with one slow link in the middle. We benchmarked the time taken by our implementation as well the 1-bit Adam implementation. The experiment was repeated with 16 and 32 workers in the chain. We varied the tensor size from 100MB to 1GB. All links except one have a bandwidth capacity of 500MBps, while the center link has a bandwidth of 100MBps.

4.1.1 1-bit Adam

Each node in the first half on the chain transfers data of order *O(K/2)* to the other half, where K is the size of the tensor on each machine. So the net data transferred across the slow link is of the order *O(N * K)*, where N is the total number of nodes in the chain. The main problem with an all-to-all communication setup is that it does not exploit the underlying network topology and hence leads to worse communication times.

4.1.2 Ring Reduce

Each node in the ring communicates only with its neighbors. The total data transferred across the slow link is nothing but the data sent from 2 nodes alone, one for communication between the node vertices sharing the slow edge and the other for communication between the last vertex in the chain with the first vertex. Hence the total data transferred across the slow link here is of the order *O(K * log(N))*. This is where our implementation gains the speedup from.

As observed in Table 1, the speedup increases as we scale the number of machines involved. This behavior can be explained by taking into consideration the difference in the order of data transferred in both implementations. The 1-bit Adam implementation has a factor of *N*, whereas the Ring reduce approach has a factor of *log(N)*.

4.2 Cluster Topology

We configured a Cluster like topology where nodes within the cluster have faster interconnect (1GBps), while nodes across the clusters have slower connection (100MBps). Again, we used ClouldLab c220g2 CPU workers for this experiment. Here too, we varied the

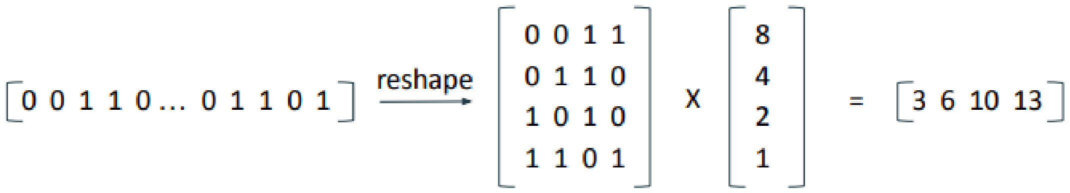
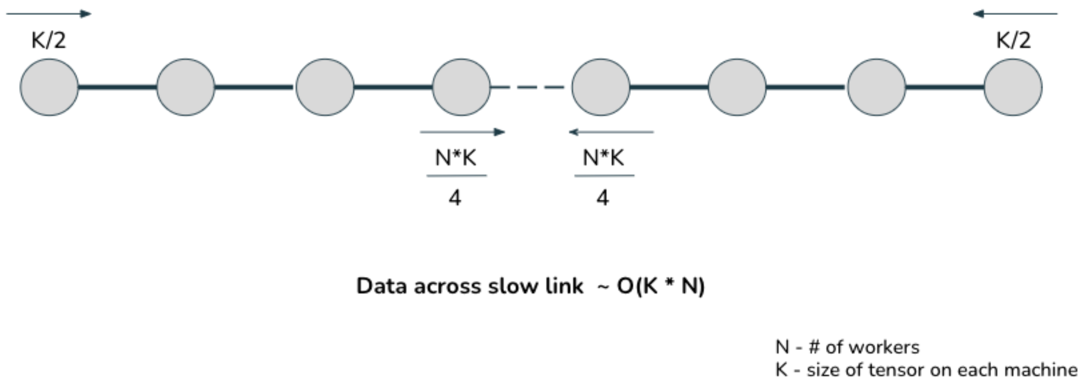
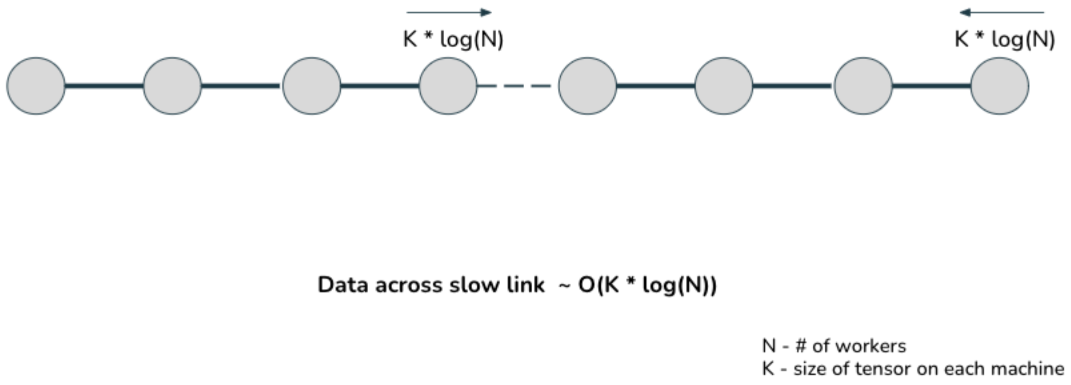


Figure 6: Converting custom bits tensor back to tensor of ints



K/2
K/2
N*K/4
N*K/4
Data across slow link ~ O(K * N)
N - # of workers
K - size of tensor on each machine

Figure 7: Data transferred across the slow link for 1-bit Adam, in a Chain like topology



K * log(N)
K * log(N)
Data across slow link ~ O(K * log(N))
N - # of workers
K - size of tensor on each machine

Figure 8: Data transferred across the slow link for Ring Reduction, in a Chain like topology

Table 1: Time taken on chain topology with one slow link in the middle (fast link - 500MBps, slow link - 100MBps)

Topology	100MB					1GB				
	ring-void	ring	1-bit Adam	speedup	prob. speedup	ring-void	ring	1-bit Adam	speedup	prob. speedup
16 - chain	4.8	6.1	11.0	**1.8**	2.3	45.8	53.6	130.7	**2.4**	2.9
32 - chain	6.1	6.9	21.9	**3.2**	3.6	57.9	67.1	238.3	**3.6**	4.1

number of machines in the cluster from 16 to 32, and the tensor size from 100MB to 1GB.

4.2.1 1-bit Adam

Here too, similar to the Chain topology, the total data transferred across the slow link is of the order *O(N * K)*.

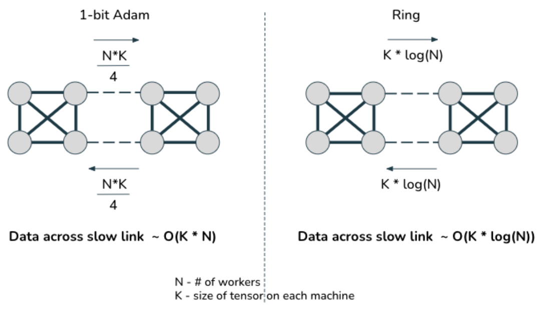

1-bit Adam
N*K / 4
N*K / 4
Data across slow link ~ O(K * N)
Ring
K * log(N)
K * log(N)
Data across slow link ~ O(K * log(N))
N - # of workers
K - size of tensor on each machine

Figure 9: Data transferred across the slow link for 1-bit Adam and Ring Reduction, in a Cluster like topology

4.2.2 Ring Reduce

Similar to the Chain topology, the data transferred across the slow link in case of a Cluster like topology is of the order *O(K * log(N))*

The Speedup in case of Cluster topology doesn't increase with the increase in the number of worker nodes. This is probably due to the fact that every node in the cluster has multiple possible routes to communicate with another node in a different cluster, which is favourable in case of an all-to-all communication model. The results are tabulated in Table 2.

4.3 Fully Heterogeneous Network

To further demonstrate the speedup achieved by the proposed Ring reduce setup, we created a fully heterogeneous chain topology, where the bandwidth available between machines varied greatly.

On a 16 node chain topology with link speeds varying from 100-250 MBps, we could observe a speedup of $\tilde{3}.5$. On a 32 node chain with link speeds varying from 20-150 MBps, we noticed an even higher speedup of $\tilde{4}.1$. The results are tabulated in Table 3.

5 Related Work

Apart from gradient compression, there are various techniques that reduce the communication cost in distributed training. Gradient sparsification is an example of such a method. In gradient sparsification, we select only a few gradients from each worker, thus resulting in sparse gradients. We take a bit vector(bv) of the same length as the gradient(gd) vector for gradient sparsification. Bit vector comprises only zeroes and ones. Zero in the bit vector represents that gradient corresponding to that point will be truncated. Therefore, in the sparsed gradient, we will have gradients that have ones in the bit vector. A Sparsed gradient vector is a dot product between the bit vector and gradient vector. Sparsification can be deterministic or random.

5.1 Random-k

Random-k [8] is a deterministic gradient sparsification method. If the size of the bit vector is l, then k indices out of l indices are chosen randomly and set as ones. Therefore the length of the sparsed gradient vector will be k.

5.2 Top-k

Top-k [3] is also a deterministic gradient sparsification method. In top-k we set the indices of the bit vector to 1 for the top k values in the gradient vector. Figure 10 is an example of 20 percent top-k gradient sparsification.

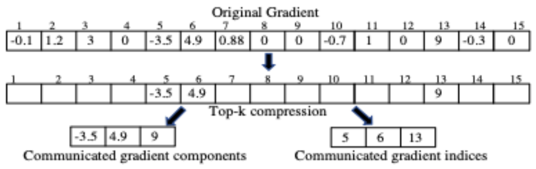


Figure 10: 20 percent top-k gradient sparsification

6 Conclusion

Majority operator is not all-reduce compatible when we try to compute the Majority without introducing additional data at each step. Existing solutions for Majority computation are bottle-necked by the data movement to centralized CPU workers which calculate the Majority and stream back the results to the CPU/GPU worker nodes.

1-bit Adam puts forward an innovative solution to calculate Majority without a set of centralized CPU workers, but uses an all-to-all communication model. An all-to-all communication model does not fully leverage the underlying network topology and could perform worse in heterogeneous network architectures.

In this paper, we build upon the existing Ring Reduce algorithm to compute the Majority by introducing additional information in each iteration. We implement a few optimizations to reduce the data being transferred across the ring. Finally, we show experiments where the proposed implementation outperforms the State of the Art implementation by achieving a speed-up of 2-4x.

Table 2: Time taken on cluster like topology with slow links across clusters (fast link - 1GBps, slow link - 100MBps)

Topology	100MB					1GB				
	ring-void	ring	1-bit Adam	speedup	prob. speedup	ring-void	ring	1-bit Adam	speedup	prob. speedup
16 - cluster	4.9	5.4	10.8	**2.0**	2.2	46.3	53.7	103.4	**1.9**	2.2
32 - cluster	5.7	6.6	10.6	**1.6**	1.9	58.4	66.8	105.6	**1.6**	1.8

Table 3: Time taken on a fully heterogeneous chain topology

Topology	100MB					1GB				
	ring-void	ring	1-bit Adam	speedup	prob. speedup	ring-void	ring	1-bit Adam	speedup	prob. speedup
16 - chain	13.4	18.6	64.9	**3.5**	4.8	130.0	182.6	664.0	**3.6**	5.1
32 - chain	42.0	55.6	225.3	**4.1**	5.4	398.0	511.2	2322.8	**4.5**	5.8

7 Future work

Gradient optimization algorithms would be more performent when they try to leverage the underlying network topology. In this paper, we tried to do that to some extent but limited ourselves to neighbor-to-neighbor communication. Future Research could lead in a direction where the synchronization algorithm gets determined dynamically based on some kind of profiling. This would lead to much optimized gradient synchronization times in distributed training.

References

[1] Tang Z, Shi S, Chu X, Wang W, Li B. Communication-Efficient Distributed Deep Learning: A Comprehensive Survey. arXiv; 2020. Available from: https://arxiv.org/abs/2003.06307.

[2] Li S, Zhao Y, Varma R, Salpekar O, Noordhuis P, Li T, et al. Pytorch distributed: Experiences on accelerating data parallel training. arXiv preprint arXiv:200615704. 2020.

[3] Aji AF, Heafield K. Sparse Communication for Distributed Gradient Descent. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics; 2017. Available from: https://doi.org/10.18653

[4] Jia X, Song S, He W, Wang Y, Rong H, Zhou F, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. arXiv preprint arXiv:180711205. 2018.

[5] Wen W, Xu C, Yan F, Wu C, Wang Y, Chen Y, et al. Terngrad: Ternary gradients to reduce communication in distributed deep learning. Advances in neural information processing systems. 2017;30.

[6] Tang H, Gan S, Awan AA, Rajbhandari S, Li C, Lian X, et al.. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed. arXiv; 2021. Available from: https://arxiv.org/abs/2102.02888.

[7] Bernstein J, Wang YX, Azizzadenesheli K, Anandkumar A. signSGD: Compressed Optimisation for Non-Convex Problems. arXiv; 2018. Available from: https://arxiv.org/abs/1802.04434.

[8] Stich SU, Cordonnier JB, Jaggi M. Sparsified SGD with Memory. arXiv; 2018. Available from: https://arxiv.org/abs/1809.07599.