Near-Optimal Sparse Allreduce for Distributed Deep Learning

Shigang Li, Torsten Hoefler
ETH Zurich

PPoPP’22
April 2-6, 2022
Seoul, Republic of Korea
Model size growing exponentially

Even if the model can be fit in a single GPU (for example, by swapping parameters between host and device memory), the high number of compute operations required can result in unrealistically long training times without parallelization. For example, training a GPT-3 model with 175 billion parameters would take 36 years on eight V100 GPUs, or seven months with 512 V100 GPUs.
Parallel and distributed training

**Data parallelism**

- **Pros:**
  - Easy to realize
- **Cons:**
  - Using DP alone may NOT work for large models, but works with others (OP, PP)
  - High allreduce overhead

This work (Ok-Topk) aims to solve

**Operator parallelism**

- **Pros:**
  - Make large model training feasible
- **Cons:**
  - Communication for each operator (or each layer)

**Pipeline parallelism**

- **Pros:**
  - Make large model training feasible
  - No collective, only P2P
- **Cons:**
  - Bubbles in pipeline
  - Removing bubbles leads to stale weights

We have Chimera (SC’21) to solve the above issues for pipeline parallelism.
Data parallelism

\[ f(w) = \mathbb{E}_{\xi \sim D} F(w; \xi) \]

- **w** denotes the model parameters.
- **F** is the loss function.
- **\( \xi \)** is a data batch sampled from a distribution **D**.

**Training:** update **w** to minimize **f** (e.g., SGD).

\[ g_t = \frac{1}{b} \sum_{i=0}^{b} \nabla F(w_t, \xi_i) \]
\[ w_{t+1} = w_t - \eta t g_t \]
Revisit Dense Allreduce (Rabenseifner’s algorithm)

Latency-bandwidth model:
- $\alpha$ is the latency;
- $\beta$ is the transfer time per word;
- $p$ is the number of processes;
- $n$ is the message size.

Comm. Cost = $2(\log p)\alpha + 2((p - 1)/p)n\beta$

Bandwidth optimal and scalable, but w.r.t. $n$
Gradient sparsification (Top$k$ SGD)

**Top$k$ SGD**: each process only selects the largest (absolute value) $k$ of $n$ components from the gradients, and usually the density $k/n$ is around 1% or less.

How to **Allreduce** these **sparse** gradients?
Algorithm 1: Sparse Allreduce based on Allgather (TopkA)

Comm. Cost = (\log p)\alpha + 2k(p - 1)\beta

Not scalable!
Algorithm 2: Dynamic Sparse Allreduce (TopkDSA, SC’19)

Inspired by dense Rabenseifner’s algorithm

Comm. Cost = \((p + 2\lg p)\alpha + \left\lfloor \frac{4k(p-1)}{p} \right\rfloor \beta, \left\lfloor \frac{(2k+n)(p-1)}{p} \right\rfloor \beta\)
Algorithm 3: Global Topk (gTopk, ICDCS’19)

Comm. Cost = \((2 \log p) \alpha + 4k (\log p) \beta\)

Solve the fill-in issue, but
(1) high top-\(k\) selection cost, and
(2) suboptimal bandwidth cost.
O(k) sparse Allreduce

The cost of balanced \( \text{split\_and\_reduce} = (P - 1)\alpha + 2k \left(\frac{(P - 1)}{P}\right) \beta \)
O(k) sparse Allreduce

(2) Global top-k selection & balanced Allgatter

\[ \text{Total Cost} \leq (2p + 2\log p)\alpha + 6k((p-1)/p)\beta \]

Less than 6k, asymptotically optimal
Scalability analysis for dense/sparse Allreduce algorithms

| Algorithms           | Bandwidth                                      | Scalability |
|----------------------|------------------------------------------------|-------------|
| Dense [12]           | $2n \frac{P-1}{p} \beta$                      |             |
| TopkA [36, 47]       | $2k(P-1)\beta$                                 |             |
| TopkDSA [36]         | $[4k \frac{P-1}{p} \beta, (2k + n) \frac{P-1}{p} \beta]$ |             |
| gTopk [42]           | $4k(\log P)\beta$                              |             |
| Gaussiank [41]       | $2k(P-1)\beta$                                 |             |
| Ok-Topk (ours)       | $[2k \frac{P-1}{p} \beta, 6k \frac{P-1}{p} \beta]$ |             |

Existing sparse Allreduce algorithms suffer from scalability issue. Ok-Topk solves the issue!
Efficient local and global top-\(k\) selection

In Ok-Topk (local and global top-\(k\)):
1. We calculate the thresholds once and reuse them in \(\tau'\) iterations (temporal locality)
2. \(O(n)\) overhead rather than \(O(n\log n)\) in sorting based top-\(k\) selection
3. More friendly to GPU than sorting
**Ok-Topk parallel SGD algorithm**

1. **Inputs:** stochastic gradient $G^i(\cdot)$ at worker $i$, value $k$, learning rate $\alpha$.
2. Initialize $\epsilon^i_0 = 0$, $G^i_0 = 0$
3. **for** $t = 1$ **to** $T$ **do**
   4. $\text{acc}^i_t = \epsilon^i_{t-1} + \alpha G^i_{t-1}(w_{t-1})$ ☀ Accumulate residuals
   5. $u_t, \text{indexes} = \text{Ok\_sparse\_allreduce}(\text{acc}^i_t, t, k)$
   6. $\epsilon^i_t = \text{acc}^i_t - \text{acc}^i_t(\text{indexes})$ ☀ Update residuals
   7. $w_t = w_{t-1} - \frac{1}{p} u_t$ ☀ Apply the model update
4. **end for**
Convergence analysis for Ok-Topk SGD

\[ \text{True global top-k gradient} \quad \text{Ok-topk gradient} \quad \text{Dense gradient} \]

For full proof refer to: Dan Alistarh, et al., The convergence of sparsified gradient methods, NeurIPS’18

The effect of \( \xi \) is dampened by both small learning rates and \( P \).
Evaluation

- **CSCS Piz Daint** supercomputer
  - Each node contains an Intel Xeon E5-2690 CPU, and one NVIDIA Tesla **P100 GPU**
  - **Cray Aries** interconnected network
  - **mpi4py** as the communication library, built against Cray-MPICH 7.7.16

Dense/Sparse algorithms used in evaluation

| Algorithms      | Bandwidth                                      |
|-----------------|------------------------------------------------|
| Dense [12]      | $2n \frac{P-1}{P} \beta$                      |
| TopkA [36, 47]  | $2k(P-1)\beta$                                |
| TopkDSA [36]    | $[4k\frac{P-1}{P} \beta, (2k + n)\frac{P-1}{P} \beta]^1$ |
| gTopk [42]      | $4k(\log P)\beta$                             |
| Gaussian k [41] | $2k(P-1)\beta$                                |
| Ok-Topk (ours)  | $[2k\frac{P-1}{P} \beta, 6k\frac{P-1}{P} \beta]^1$ |

Neural networks used for evaluation

| Tasks                  | Models      | Parameters    | Dataset      |
|------------------------|-------------|---------------|--------------|
| Image classification   | VGG-16 [44] | 14,728,266    | Cifar-10     |
| Speech recognition     | LSTM [21]   | 27,569,568    | AN4 [1]      |
| Language processing    | BERT [13]   | 133,547,324   | Wikipedia [13] |
Weak scaling evaluation

For Ok-Topk:

1. Much better scalability for the communication overhead than the others.
2. Threshold reuse strategy for top-k selection is very effective.
Model accuracy evaluation

- VGG training on 16 GPU nodes
- LSTM training on 32 GPU nodes
- BERT Pretraining on 32 GPU nodes
Conclusion

1. Model size rapidly grows
   - [Graph showing model size growing exponentially]

2. Sparse algorithms suffer from scalability issue
   - [Table showing scalability analysis for sparse Allreduce algorithms]

3. O(k) sparse Allreduce
   - [Diagram showing O(k) sparse Allreduce]

4. O(k)-Topk SGD and convergence analysis
   - [Diagram showing convergence analysis for O(k)-Topk]

5. Evaluation on supercomputer
   - [Table showing evaluation metrics and results]

6. For the future work, we will study how to use O(k)-Topk with a hybrid data and pipeline parallelism.

For more questions: shigangli.cs@gmail.com