

To appear at SIGCOMM '26

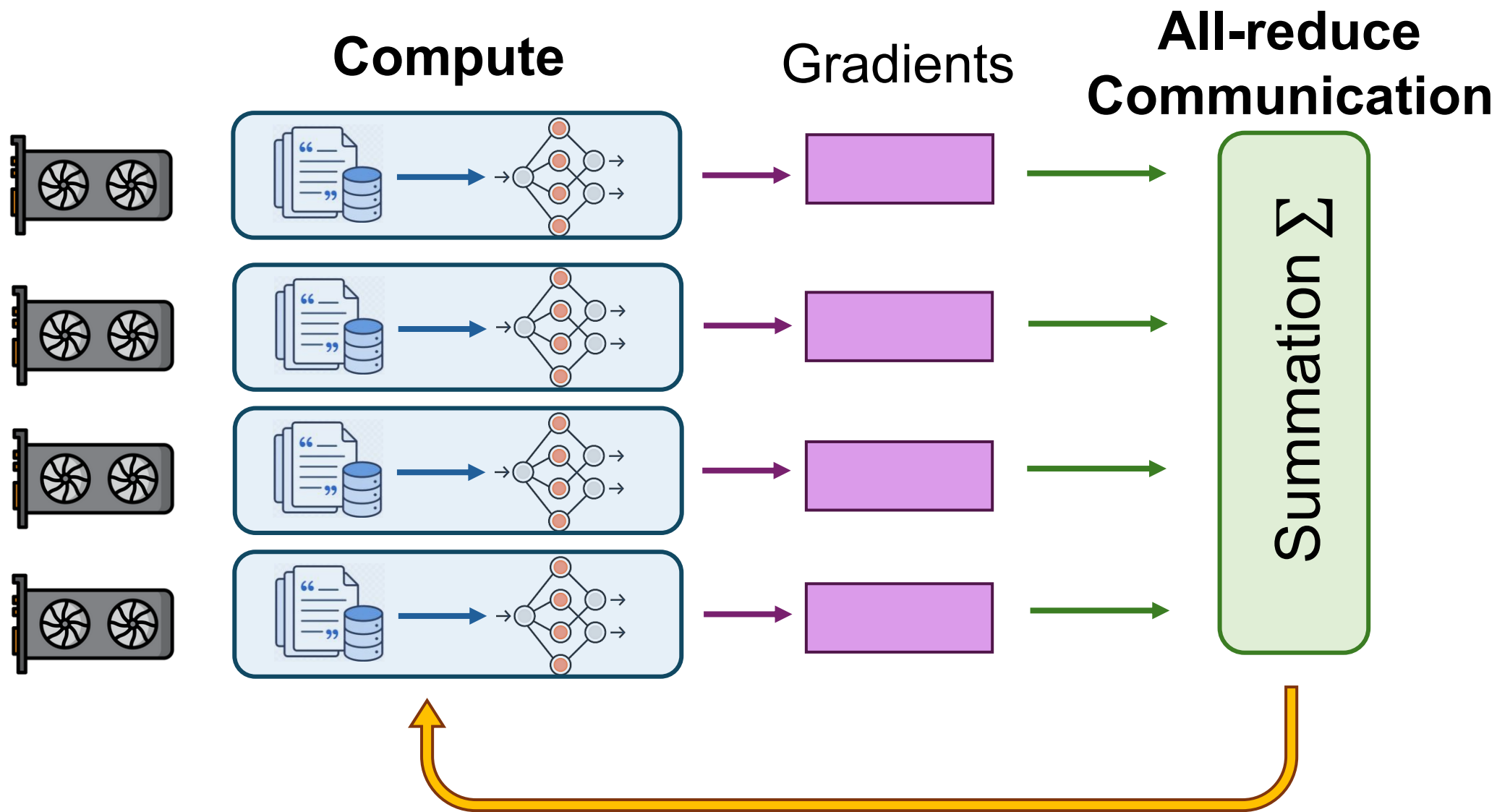
DynamiQ: Accelerating Gradient Synchronization using Compressed Multi-hop All-reduce

Wenchen Han, Shay Vargaftik

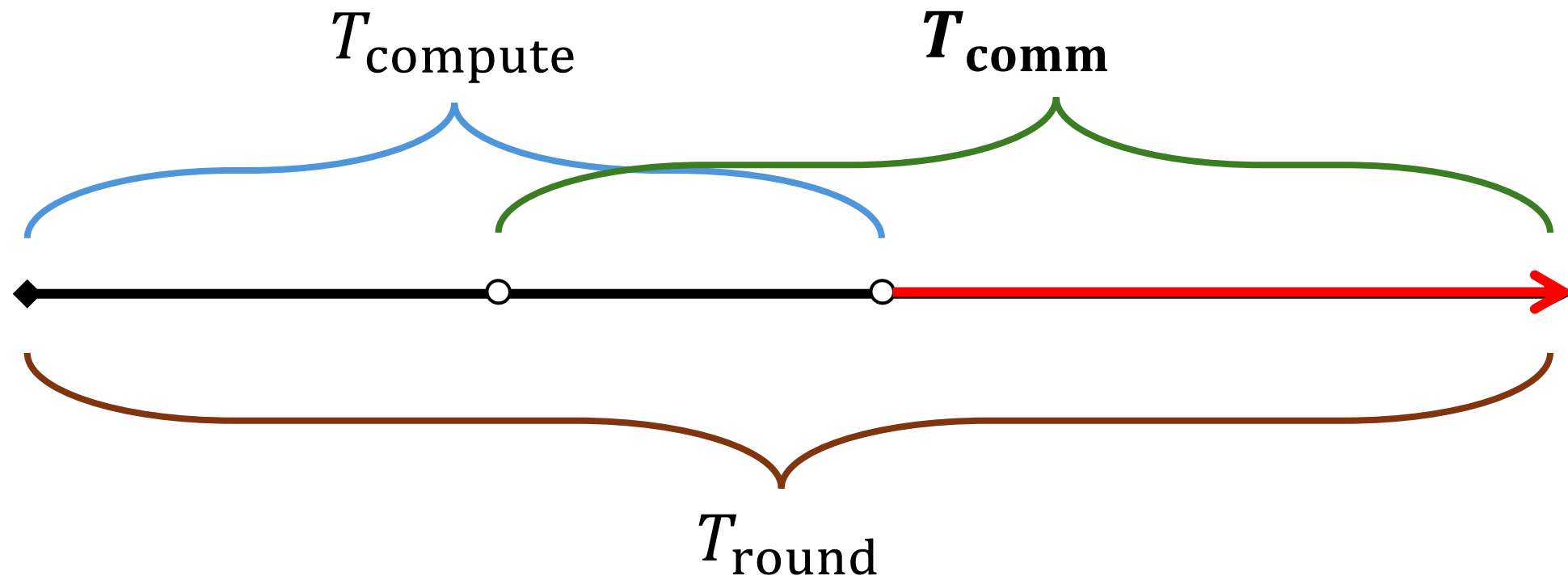
Michael Mitzenmacher, Ran Ben Basat



Data Parallelism in Distributed LLM Training

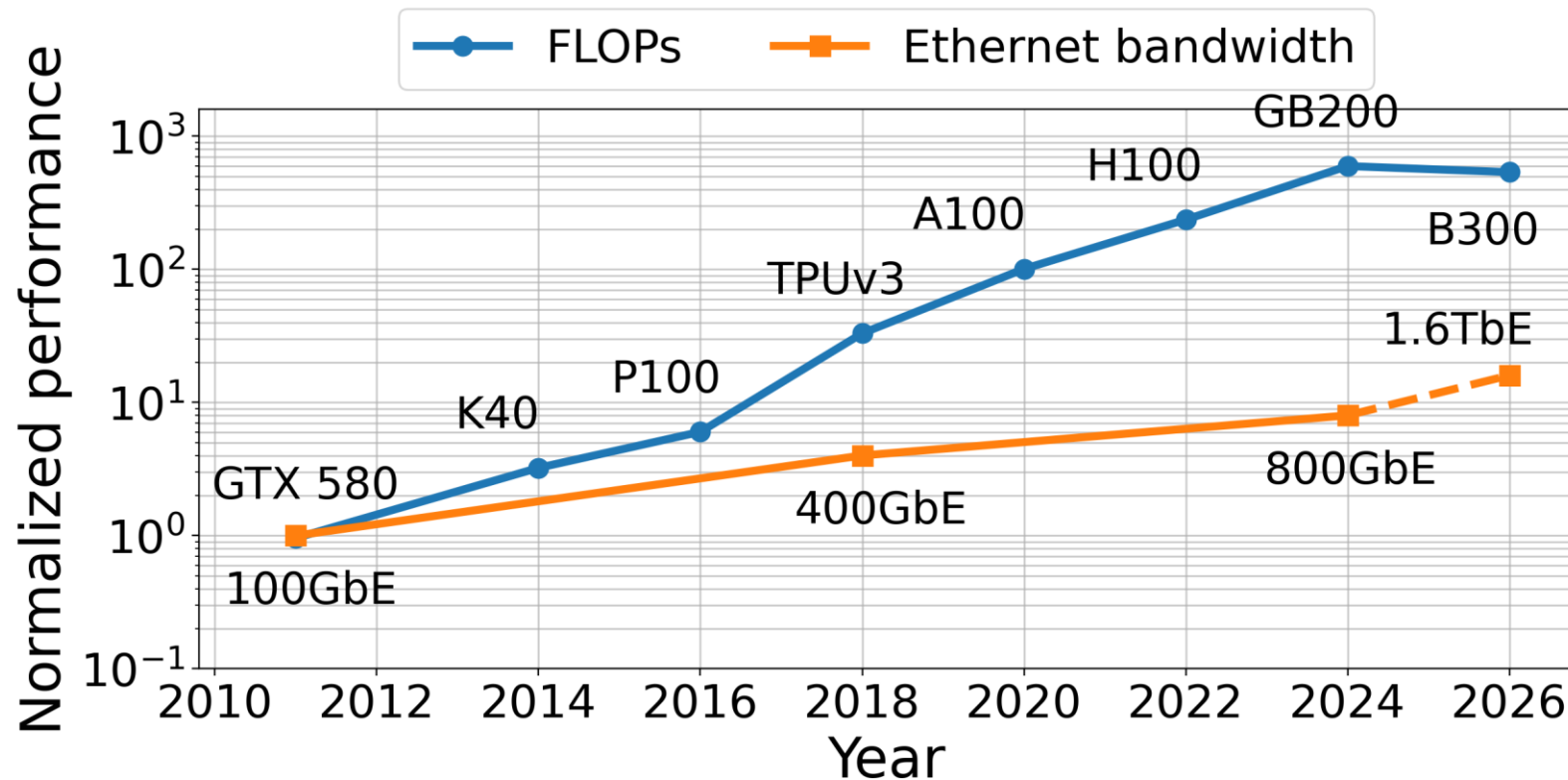


Data Parallelism in Distributed LLM Training



Gradient Communication Remains a Bottleneck

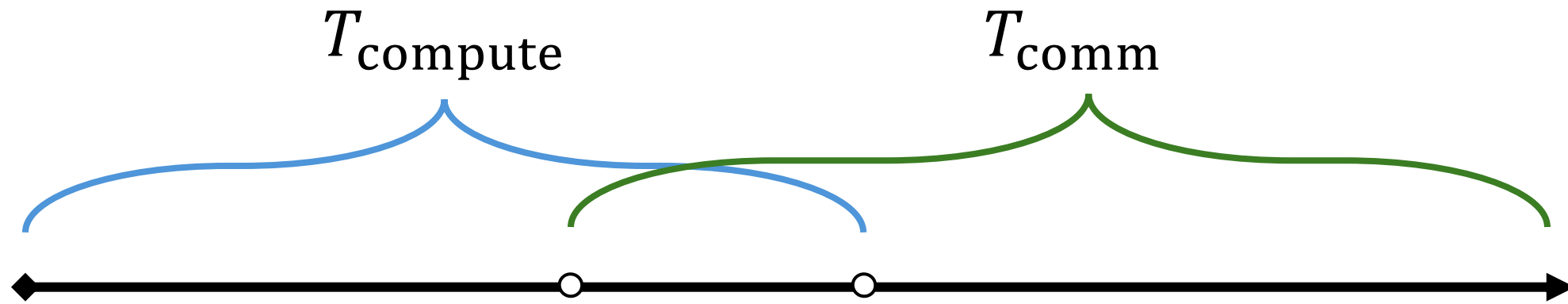
- Up to 44% in large LLM training, 800Gbps network [1].
- Compute grows faster than bandwidth.



One Line of Promising Solutions: Compression

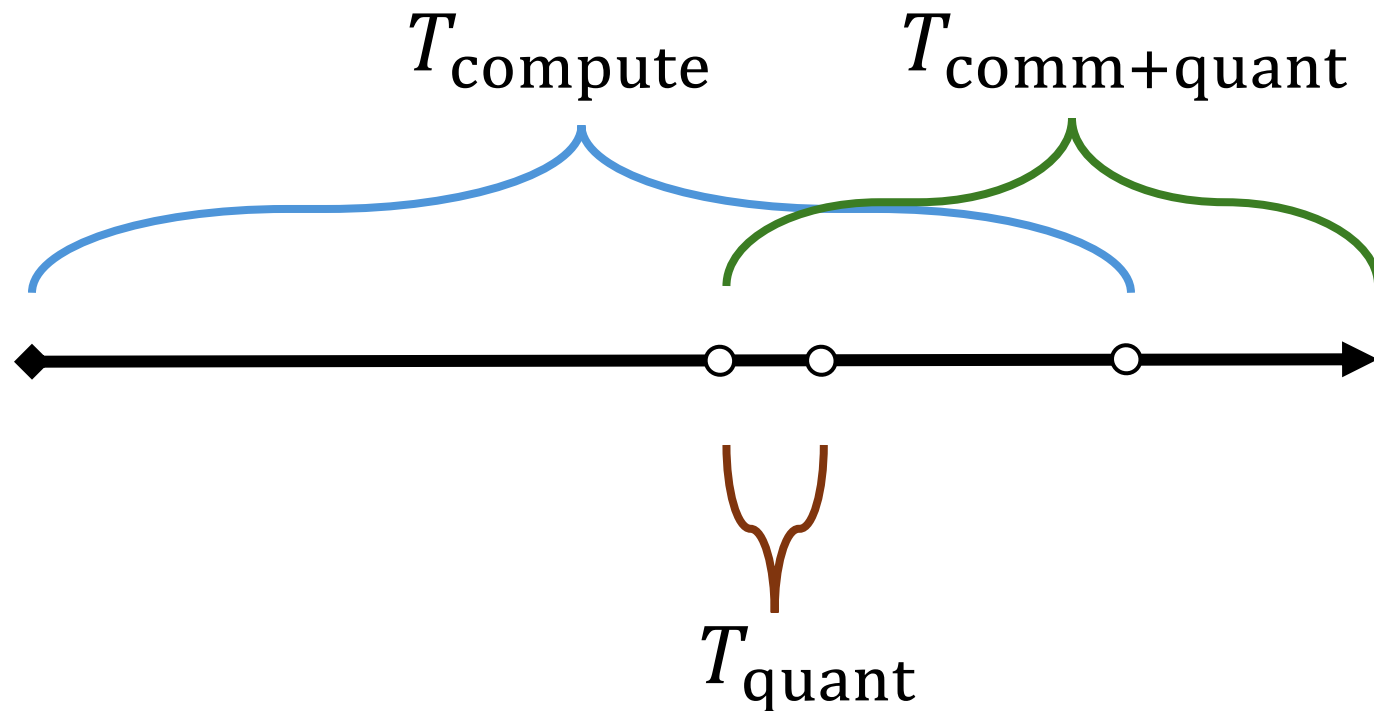
BF16

0.1	-1.0	-0.3	0.9
-----	------	------	-----



One Line of Promising Solutions: Compression

$$\text{INT4} \quad \begin{bmatrix} 1 & -7 & -2 & 6 \end{bmatrix} \times \frac{1}{7}$$



One Line of Promising Solution: Compression

INT4

1

-7

T_{compute}

1

T_{comp}

Gradient Compression Supercharged High-Performance Data Parallel DNN Training

Youshui Bai¹, Cheng Li^{1,4}, Quan Zhou¹, Jun Yi², Ping Gong¹, Feng Yan³, Ruichuan Chen¹, Yulong Xu^{1,4}
¹University of Science and Technology of China ²University of Nevada, Reno ³Nokia Bell Labs
⁴Anhui Province Key Laboratory of High Performance Computing

Keywords: DNN training, gradient compression

ACM Reference Format:

Youshui Bai¹, Cheng Li^{1,4}, Quan Zhou¹, Jun Yi², Ping Gong¹, Feng Yan³, Ruichuan Chen¹, Yulong Xu^{1,4}, 2021. Gradient Compression Supercharged High-Performance Data Parallel DNN Training. In ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21), October 26–29, 2021, Virtual Event, Germany. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3477132.3483553>

Abstract

Gradient compression is a promising approach to alleviating the communication bottleneck in data parallel deep neural network (DNN) training by significantly reducing the data volume of gradients for synchronization. While gradient compression is being actively adopted by the industry (e.g., Facebook and AWS), our study reveals that there are two critical but often overlooked challenges: 1) inefficient coordination between compression and communication during gradient synchronization incurs substantial overheads, and 2) developing, optimizing, and integrating gradient compression algorithms into DNN systems imposes heavy burdens on DNN practitioners, and ad-hoc compression implementations often yield surprisingly poor system performance.

In this paper, we first propose a compression-aware gradient synchronization architecture, CaSync, which relies on a flexible composition of basic computing and communication primitives. It is general and compatible with any gradient compression algorithms and gradient synchronization strategies, and enables high-performance computation-communication pipelining. We further introduce a gradient compression toolkit, ComplL, to enable efficient development and automated integration of on-GPU compression algorithms into DNN training. HiPress is a 16-node framework HiPress with CaSync and ComplL. HiPress is open-sourced and runs on mainstream DNN systems such as MXNet, TensorFlow, and PyTorch. Evaluation via a 16-node cluster with 128 NVIDIA V100 GPUs and 100Gbps network shows that HiPress improves the training speed over current compression-enabled systems (e.g., BytePS-onebit and Ring-DGC) by 17.2%-69.5% across six popular DNN models.

CCS Concepts: • Computer systems organization → Distributed architectures; Neural networks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from <https://doi.org/10.1145/3552326.3557955>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from <https://doi.org/10.1145/3552326.3557955>

1 Introduction

To efficiently train large DNN models over the continuously growing datasets, it has been a norm to employ data parallel DNN training to explore massive parallelism in an increasingly large cluster of GPU nodes [18, 45, 47, 64, 81]. In a typical setting, each node iterates over its own data partition in parallel, and exchanges a large volume of gradients with other nodes per iteration via a gradient synchronization strategy like Parameter Server (PS) [30, 34] or Ring-allreduce [7]. However, in recent years, the fast-growing computing capability, driven by the booming of GPU architecture innovations [49] and domain-specific compiler techniques [14, 17, 57, 58], tends to result in more frequent and heavier gradient synchronization during data parallel DNN training. This trend puts high pressure on the slower-growing bandwidth and reduces the chance of pipelining computation and communication during training. We have found that, even with the latest highly-optimized BytePS [30] and Ring-allreduce [64] synchronization strategies, the communication time for gradient synchronization still accounts for 63.6% and 76.8% of the total time for training the Bert-large and Transformer models across 16 AWS EC2 instances, each with 8 NVIDIA V100 GPUs, in a 100Gbps network. Thus, there is a fundamental tension between gradient communication and computation in data parallel DNN training [61].

Gradient compression algorithms have a great potential to substantially reduce the data volume being synchronized with a negligible impact on training accuracy and convergence [4, 37, 67, 74, 76]. This practice of gradient compression is being adopted by the industry. In fact, the efforts from Facebook and AWS have begun since June 2020 [5, 20]. However, our experiment shows that the actual training speedups of compression-enabled DNN systems are far behind their expectations. For instance, applying gradient compression to

Espresso: Unleashing the Compression with Strategies

T. S. Eugene Ng
Rice University

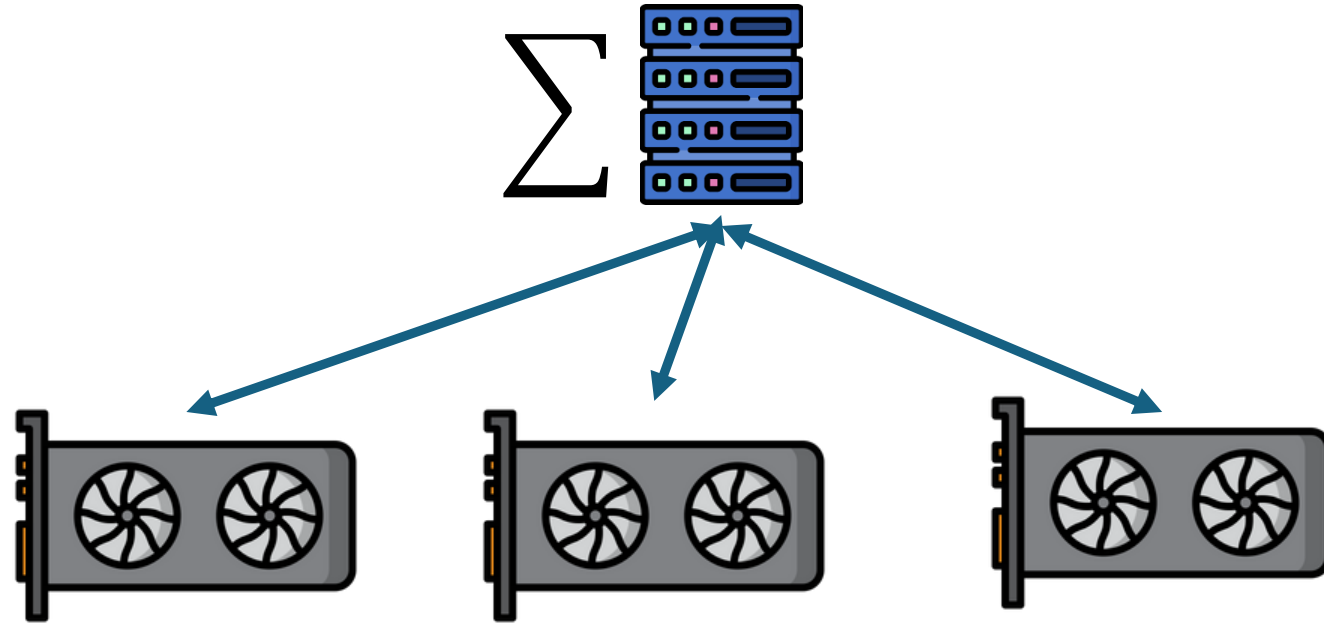
Machine Learning
organization → Dis-
tributed methodologies →

keep
4) or lin-
of workers
time decreases
therefore, train-
and, in the latter case, the
(ideally) remains constant.
time increases with the num-
berheads [35, 50, 64]. Moreover,
ade training quality [31]. To en-
hance communication overheads by
significant gap between the measured
scaling.

abel, pipeline-parallel, and asynchronous data
out of scope.

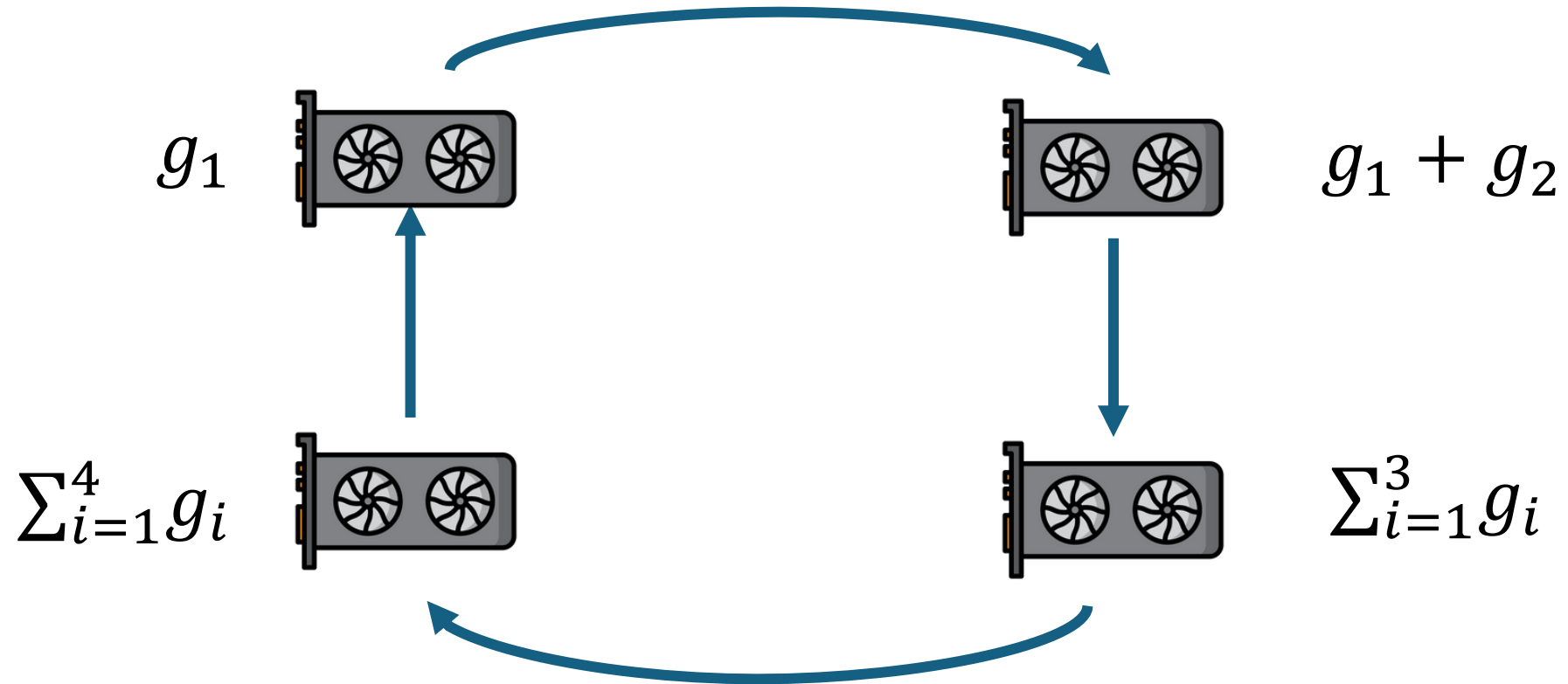
As such, as wait-free back-
ground aggregation algorithms [17,
74], priority-based schedul-
ing for the latest state-of-the-art
BERT-base [20] with 64 NVIDIA
connected by a 100Gbps Eth-

Most Compression Schemes Are Designed for PS[2]



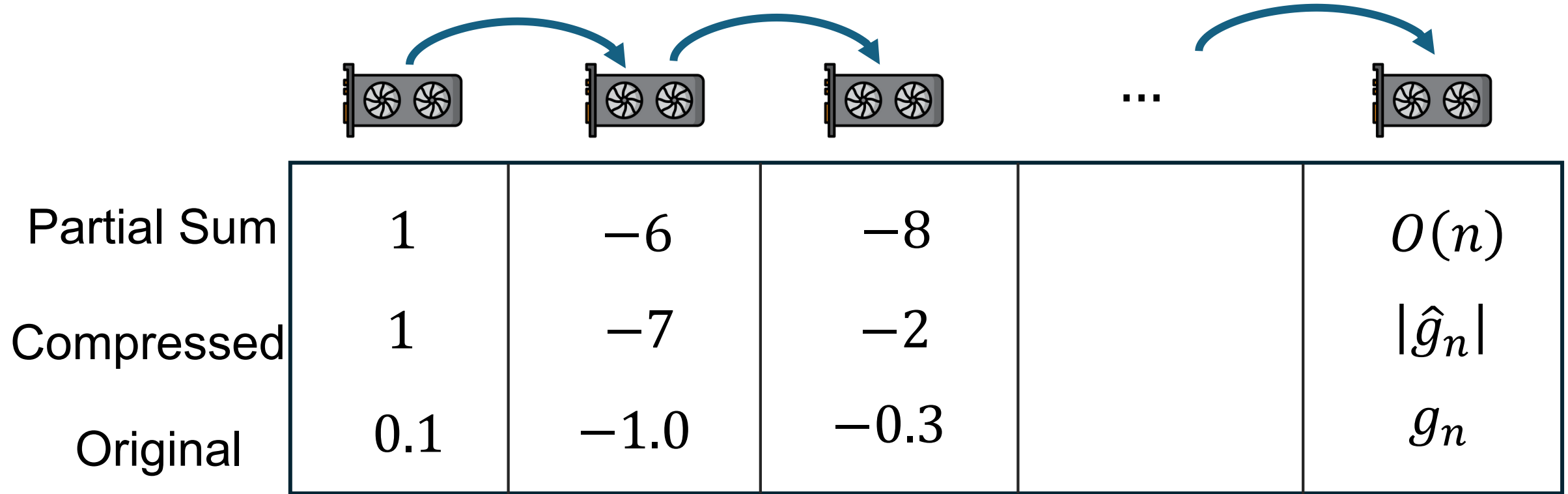
- Parameter Server (PS): Single-hop, centralized

None Are Optimized for **Multihop All-reduce**



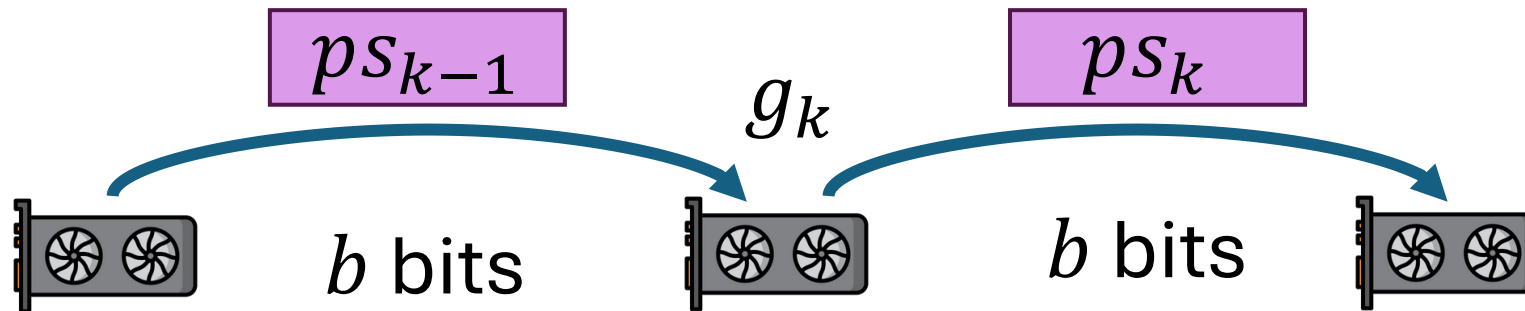
- Multi-hop All-reduce, decentralized.
- Some variants are scalable, de-facto standard.

The Main Challenge: Overflows.



Decompress Accumulate Recompress

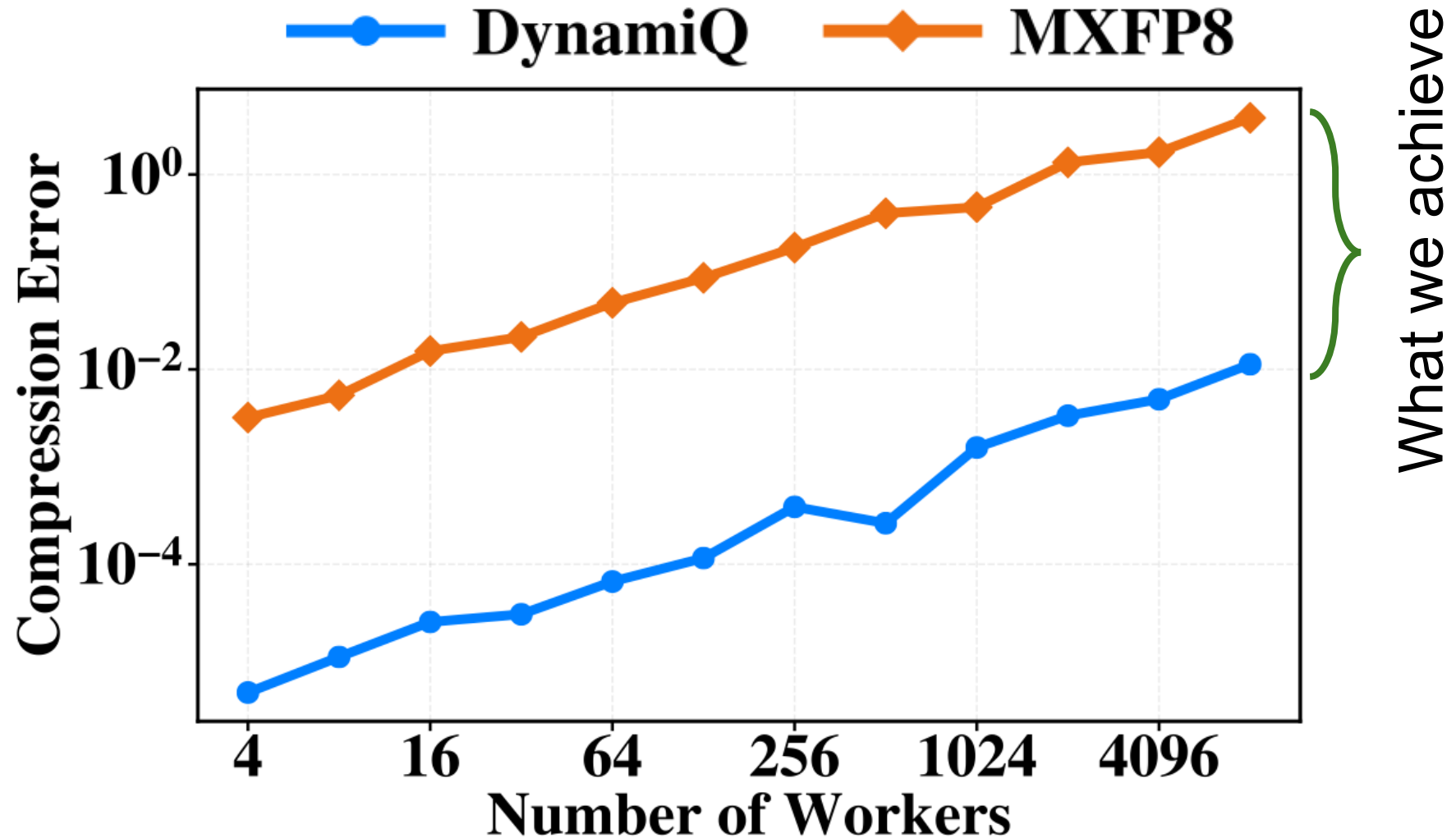
- $ps_k = \text{recompress}(\text{decompress}(ps_{k-1}) + g_k)$



Challenges

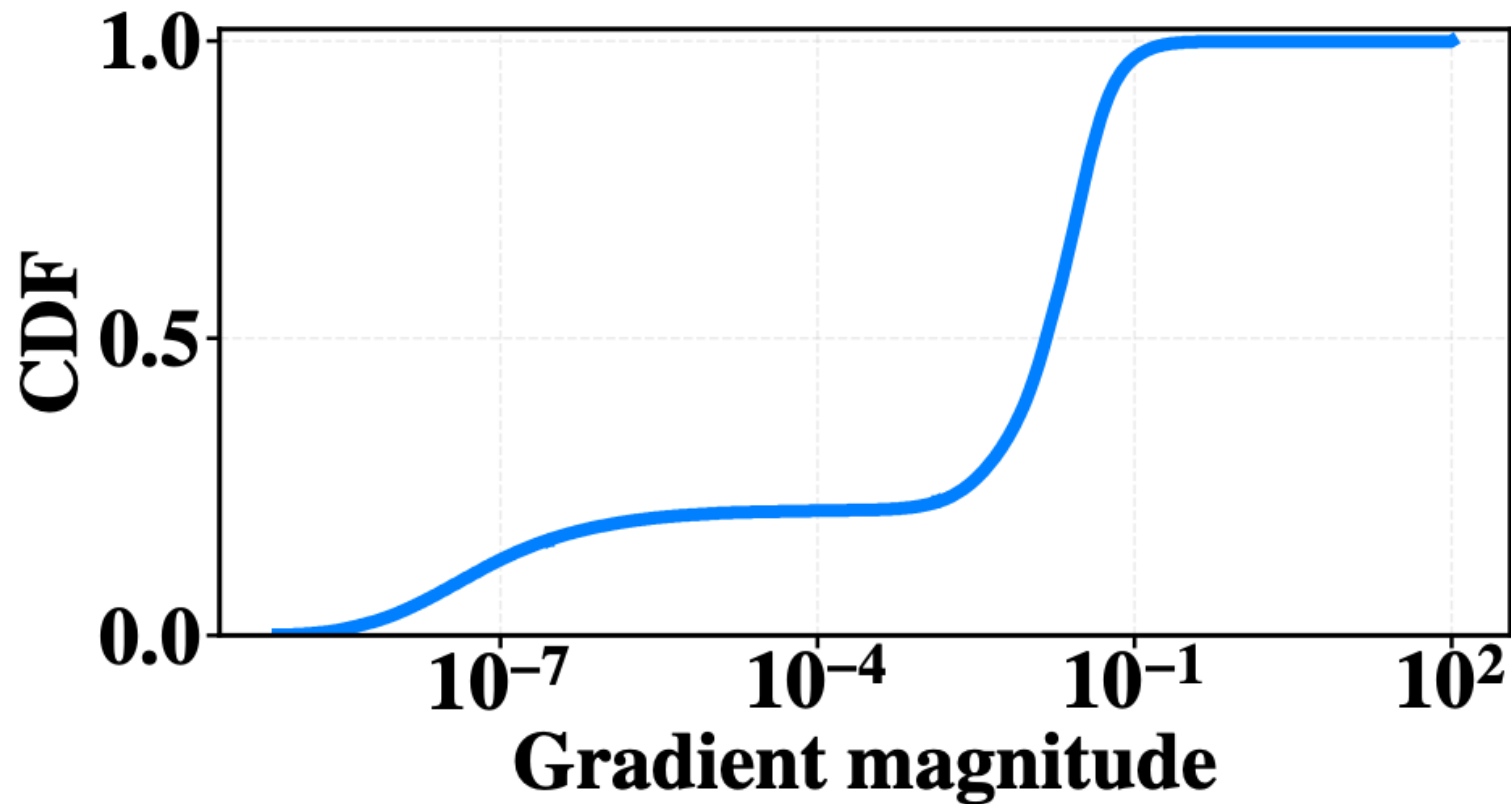
- Compression error that accumulates over hops
- Computational overhead for de/recompress

Relative Compression Error



Minimizing Compression Error: Multi-bitrates

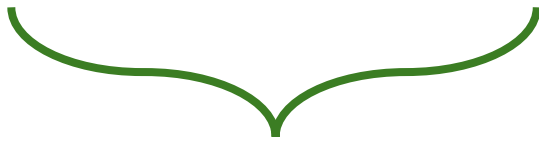
0.1	0.4	3.1	5.3	5E-3	1E-3	8E-7	2E-6
-----	-----	-----	-----	------	------	------	------



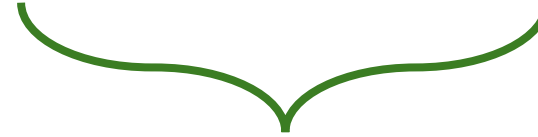
Highly Skewed Gradient

Minimizing Compression Error: Multi-bitrates

0.1	0.4	3.1	5.3	5E-3	1E-3	8E-7	2E-6
-----	-----	-----	-----	------	------	------	------



Compress with 8 bits



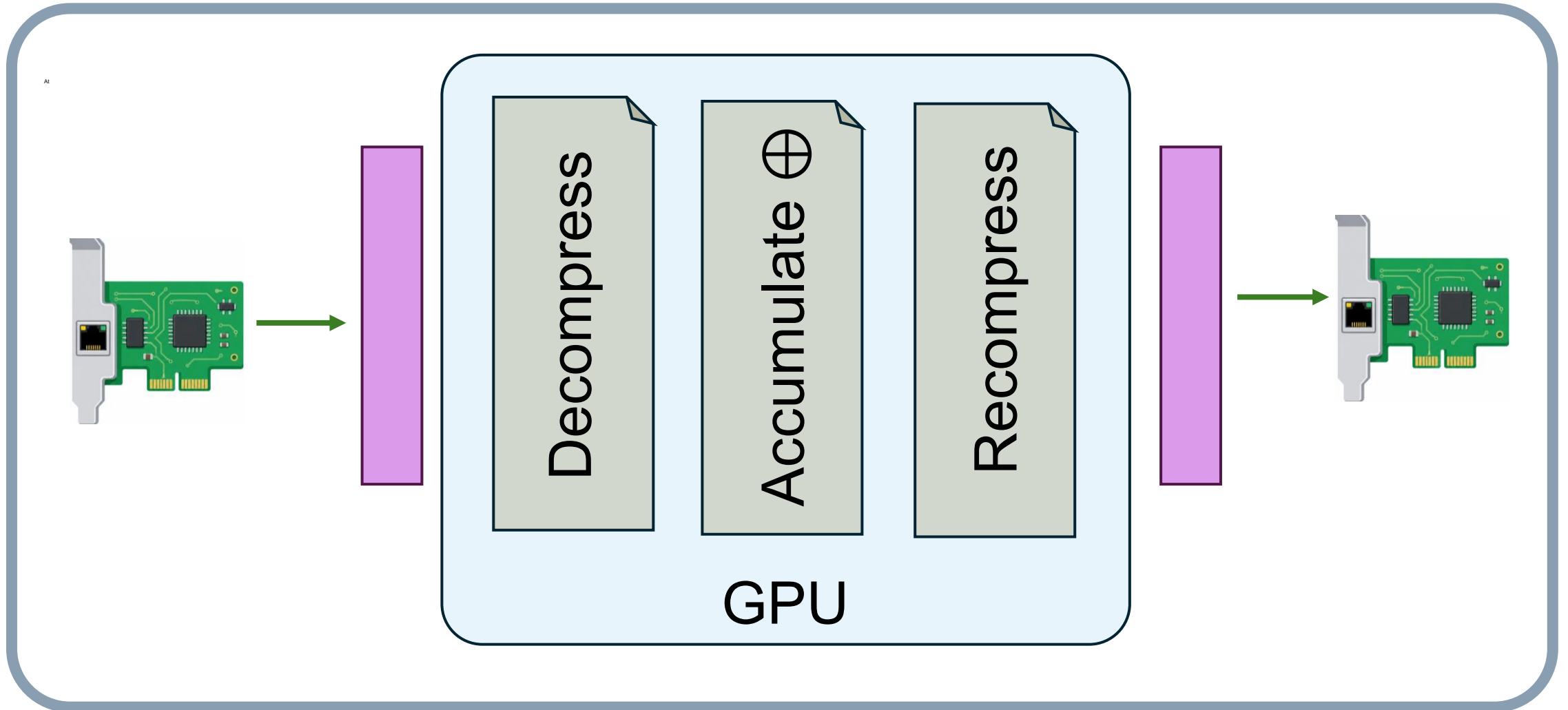
Compress with 2 bits

Idea: allocate more bits to more important entries

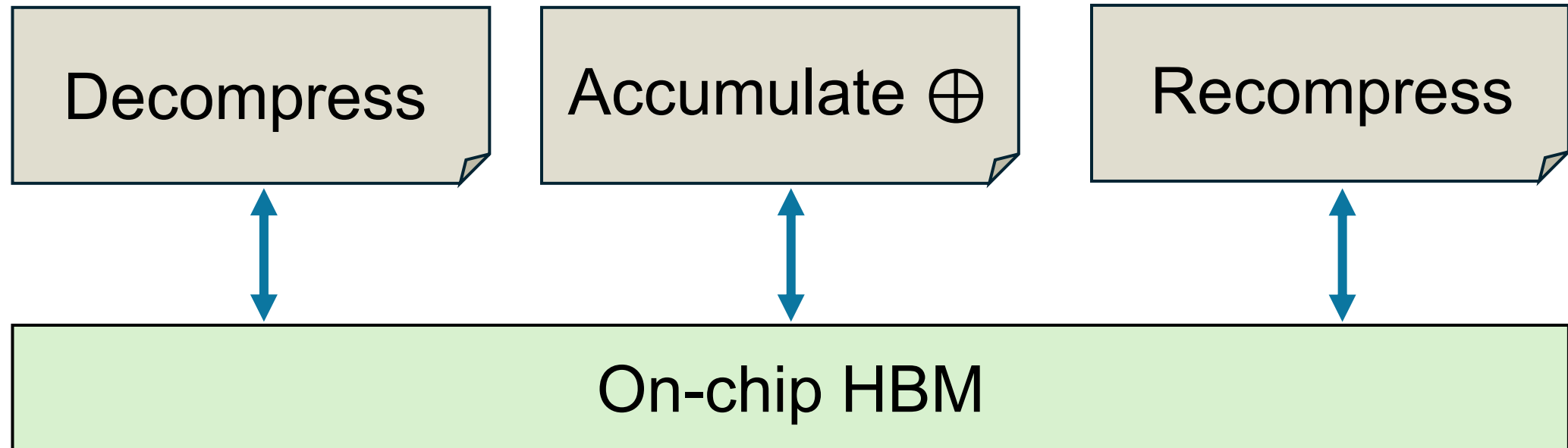
Challenges

- Compression error that accumulates over hops
- Computational overhead for de/recompress

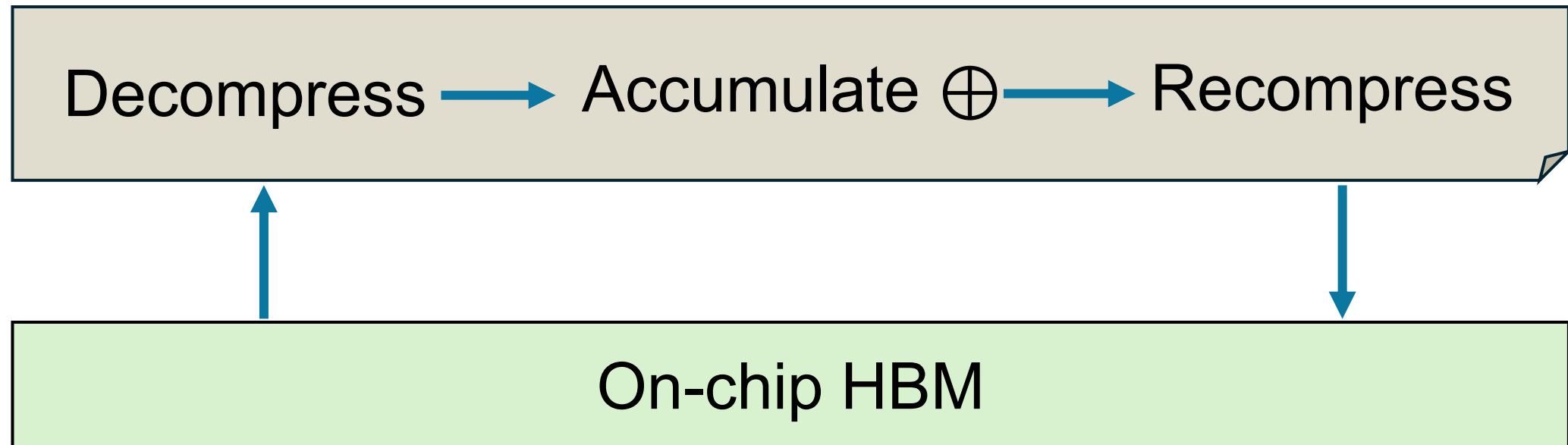
Computational Overhead



Optimization 1: Fused Kernels.

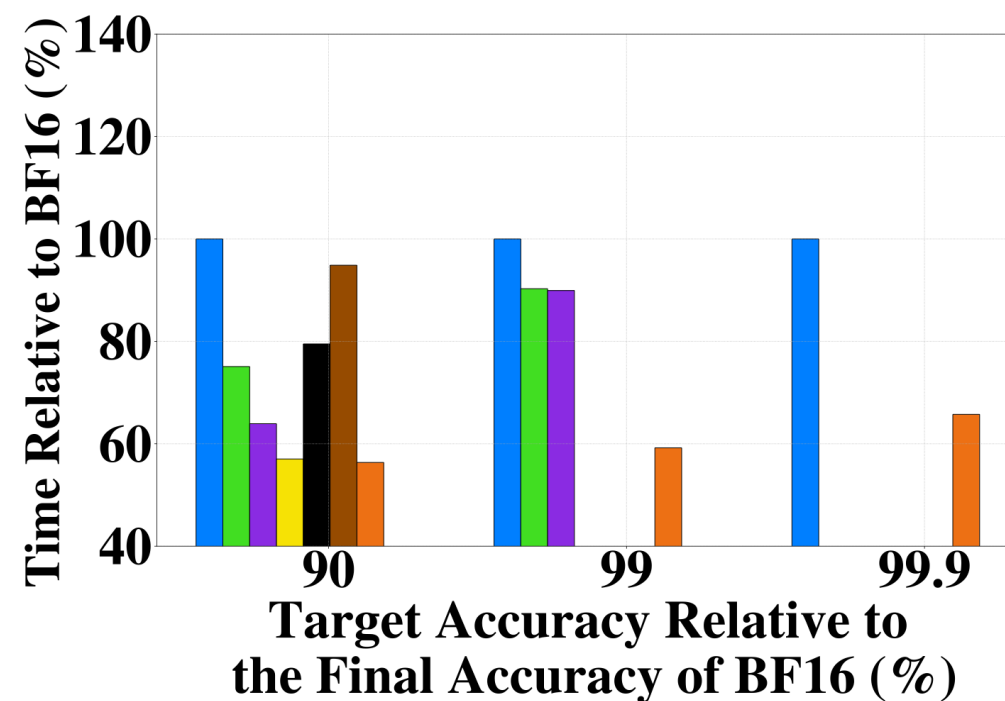
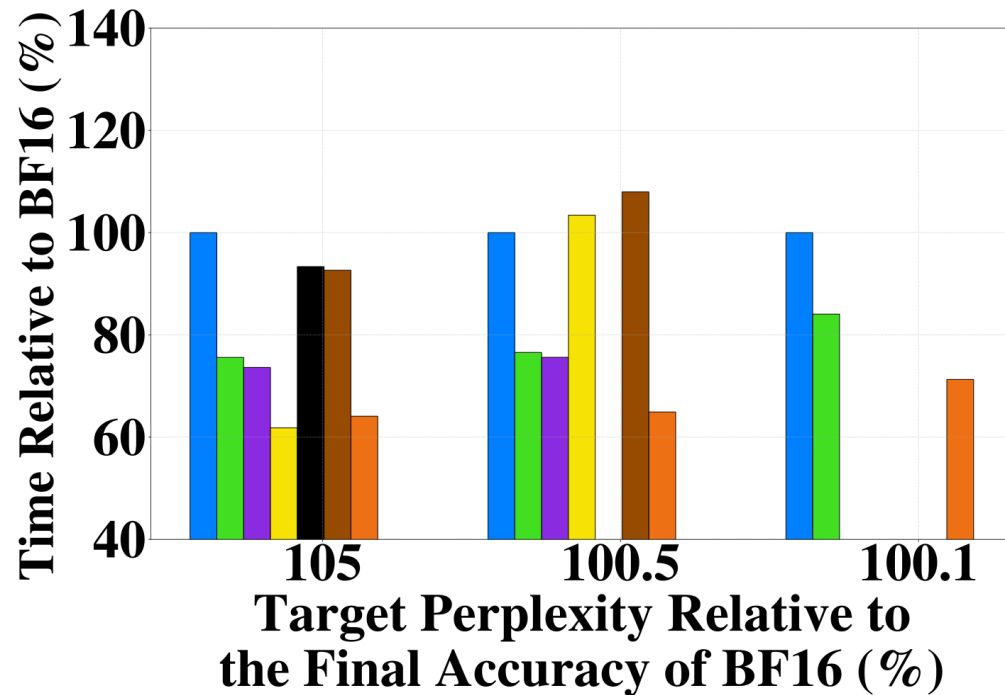
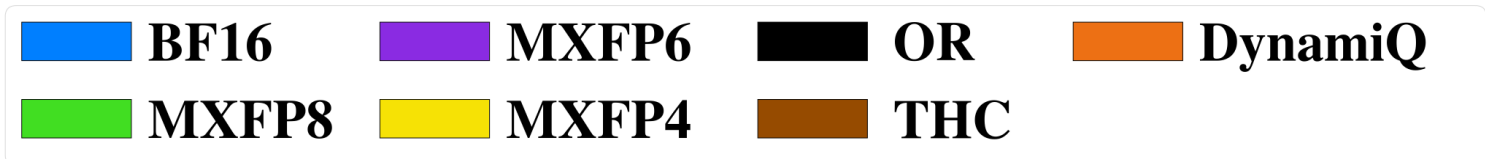


Optimization 1: Fused Kernels.



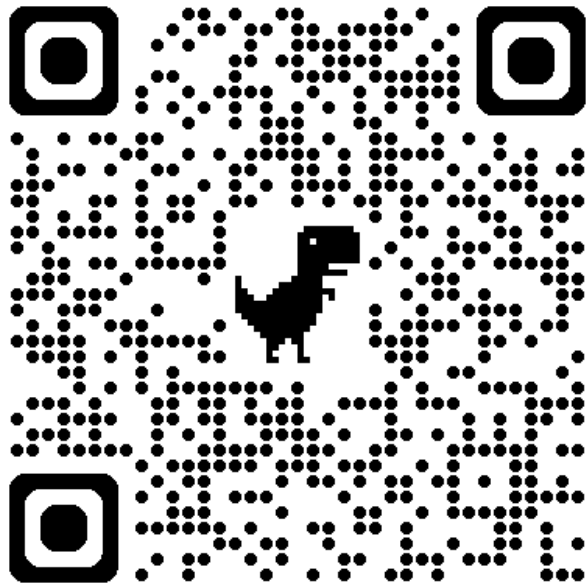
2-3x speedup

Main Evaluation Results

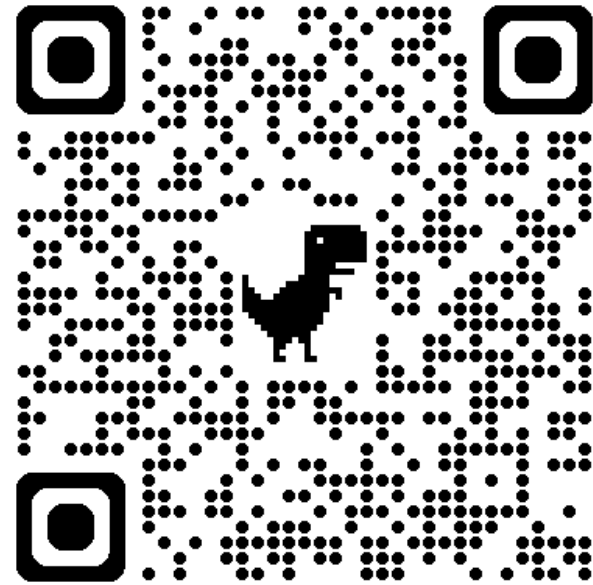


Thank you!

- Multi-hop all-reduce with compression is now practical with DynamiQ.



Our Paper

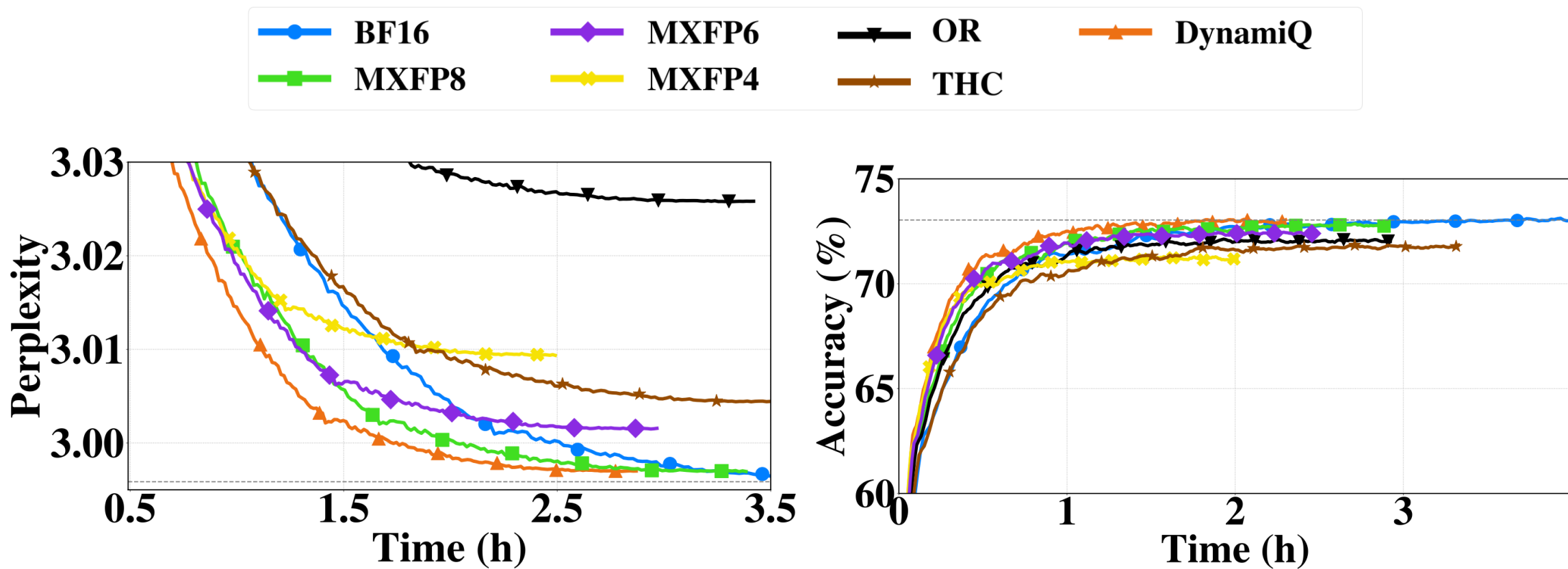


Our Code

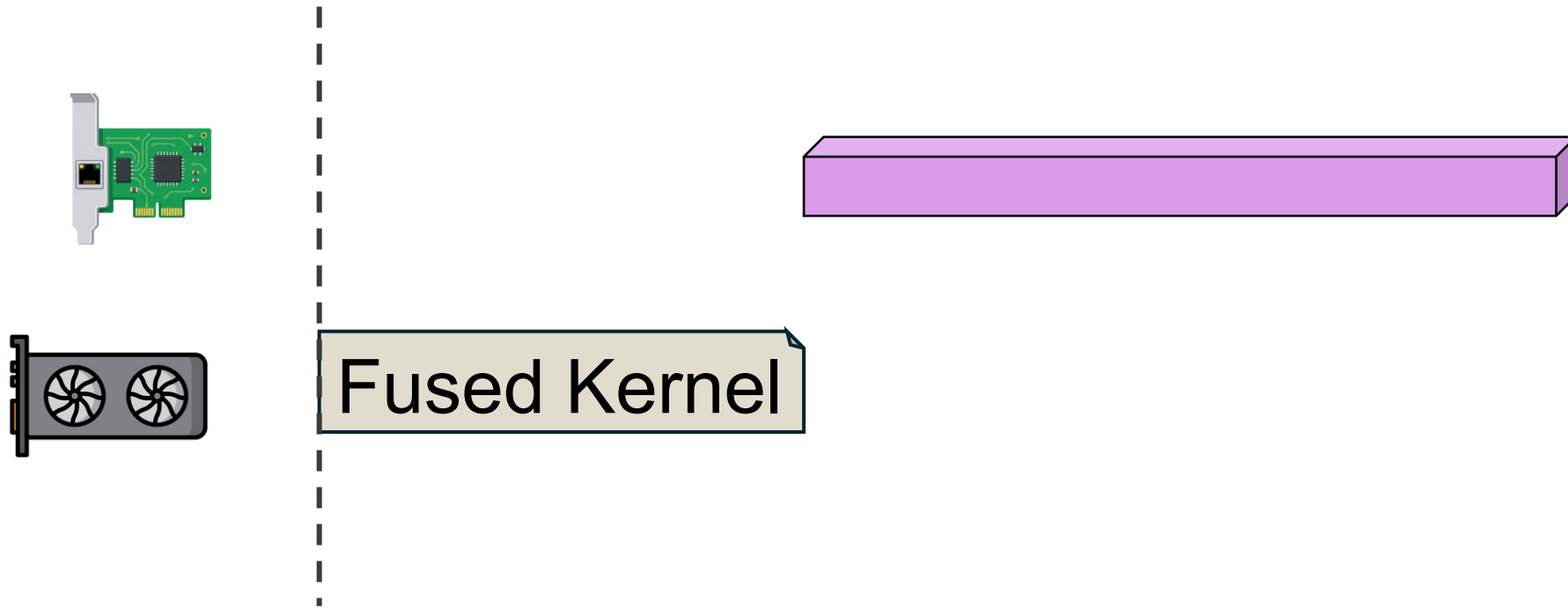
References

- [1] Wang et al. ZeRO++: Extremely Efficient Collective Communication for Giant Model Training. Microsoft Research.
- [2]. Han et al. Beyond Throughput and Compression Ratios: Towards High End-to-end Utility of Gradient Compression. In HotNets '24.

Main Evaluation Results

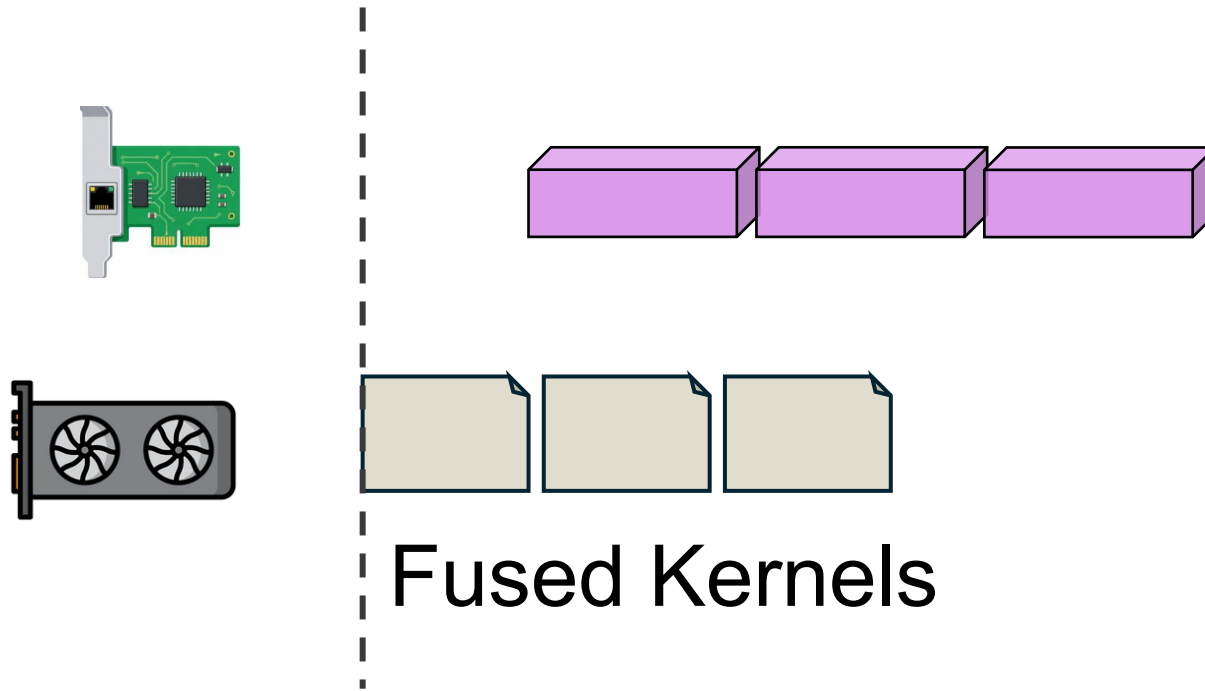


Optimization 2: Pipelined Communication



Serialized computation and communication

Optimization 2: Pipelined Communication



Pipelined, blockwise computation and communication

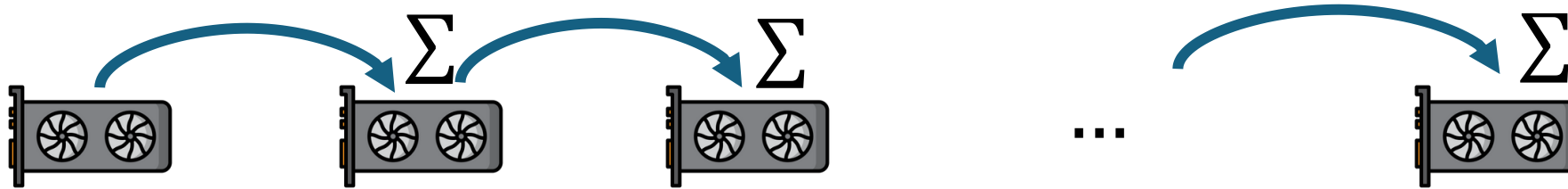
Why This Works: Intuition

- Each group i contributes to $e_i = \frac{|g_i|^2}{4^{b_i}}$.
- We want to minimize $\sum e_i$ subject to $\frac{1}{n} \sum b_i = \bar{b}$
- Intuitively, allocating bits to groups with larger values contributes to more error reduction.

Summary of Additional Results

- Running background jobs over shared network.
- Experiments with butterfly all-reduce.
- Scalability analysis up to 8192 workers.

The Main Challenge: Overflows.



- Partial-sum range grows linearly along the AR topo.
- With limited bit-width overflows happen.