Overlap Communication with Dependent Computation via Decomposition in LargeDeep Learning Models

Shibo Wang, Jinliang Wei, Amit Sabne, et al., Google/Waymo

ASPLOS '23

Presenter:Muhan Zhang

Background & Motivation

Why Overlap Communication & Computation?

Trend of Large Models

- Rapid growth in NLP, recommender systems, speech recognition, etc.
- Models like GPT-3 or GLaM can have hundreds of billions to over a trillion parameters.

Challenge

- Single-device memory and compute resources are insufficient.
- Model parallelism is necessary to split large models across multiple accelerators.

Intra-layer Model Parallelism

Definition

- Splits a single layer across multiple devices instead of splitting the model by layers (pipeline parallelism).

Why is it important?

- Single layer (e.g., large feed-forward or embedding layer) might not fit into one accelerator’s memory.
- Meet the need for excessively large global batch sizes.

Common communication operations

- AllGather, ReduceScatter, AllReduce, CollectivePermute, etc.

Collective Communication Primitives

Key Operations Used in Intra-Layer Model Parallelism

AllGather

- Gathers distinct data partitions from all devices and concatenates them such that every device ends up with the full tensor.
- Example: If each device has a shard A0, A1, A2, etc., AllGather lets every device receive [A0, A1, A2, ...].

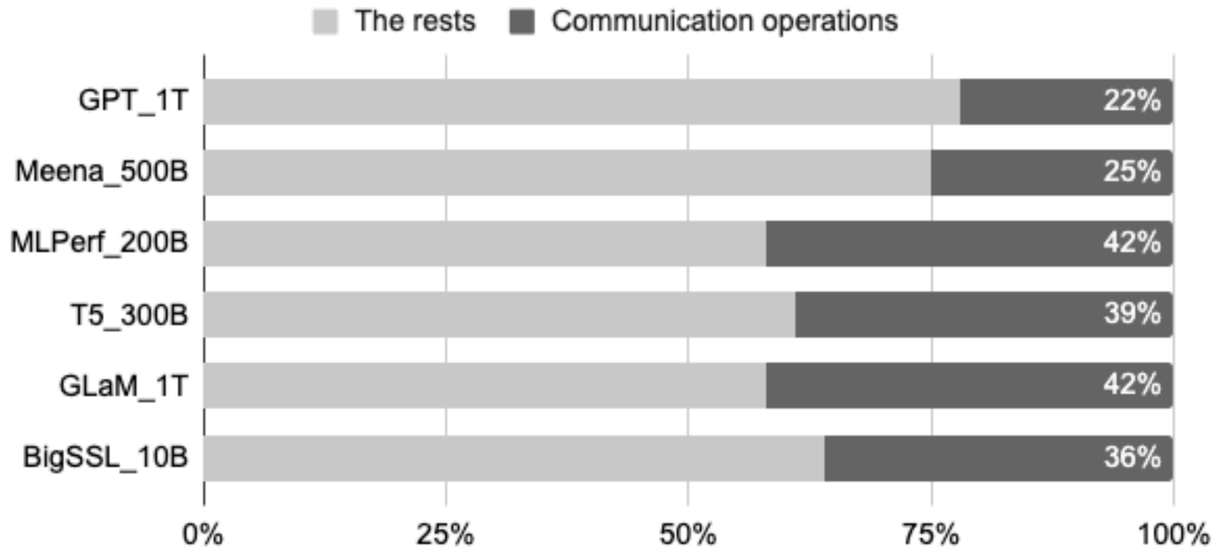
ReduceScatter

- Performs an element-wise reduction (such as sum) across data from all devices, then scatters the reduced result partitions back to them.
- Inverse of AllGather: each device receives only the partition it owns of the final reduced tensor.

Communication Overhead in Practice

Training Step Time Breakdown for Large Models

- Communication Bottleneck
- Intra-layer model parallelism introduces significant cross-device communication (e.g., AllGather, ReduceScatter).
- Communication can dominate runtime in large-scale training.

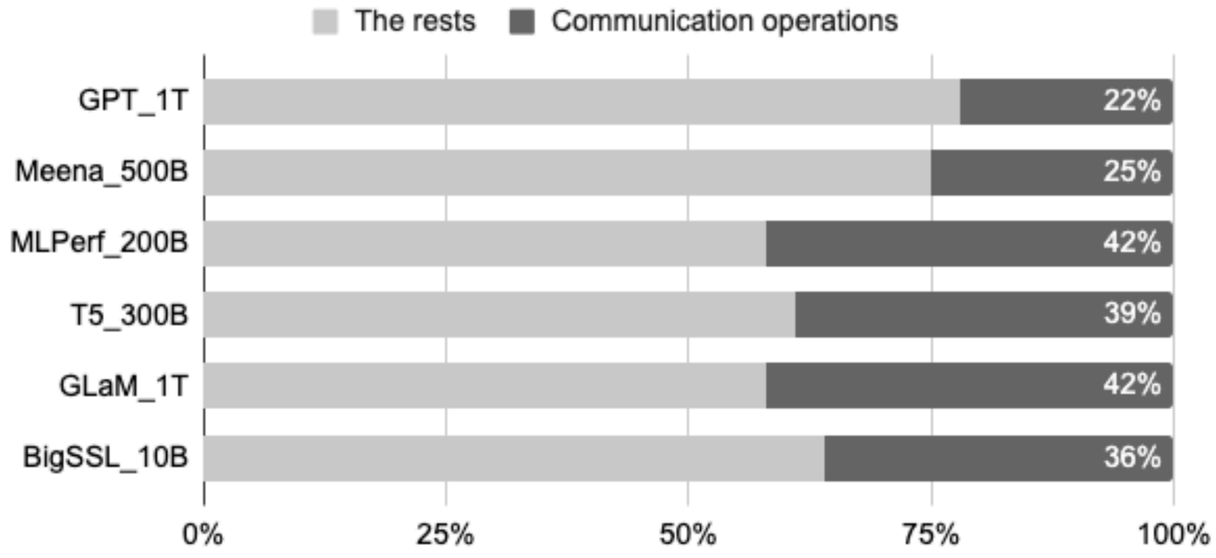


Training step time breakdown of large models.

Communication Overhead in Practice

Training Step Time Breakdown for Large Models

- Shows real measurements on TPU v4 Pods with 128–2048 chips.
- Communication can consume from ~22% (GPT_1T) up to 42% (MLPerf_200B, GLaM_1T) of the step time.
- Emphasizes the need to hide or reduce communication latency in large-scale training.



Training step time breakdown of large models.

Method Overview:

Overlapping Communication & Computation

Challenge: Large-scale models require intra-layer model parallelism, which introduces expensive collective communication (AllGather, ReduceScatter, etc.).

Core Idea: Decompose large collective operations and their dependent Einsums into smaller pieces and run in a loop to overlap computation with communication.

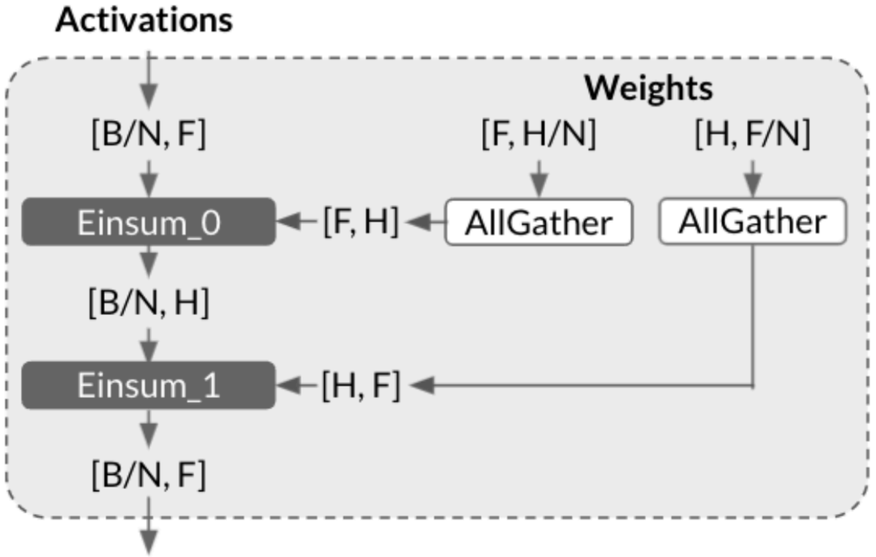
Main Techniques:

- Identifying AllGather/Einsum and Einsum/ReduceScatter patterns
- Decomposing them into looped partial compute + partial communication
- Using loop unrolling and bidirectional communication
- Adjusting fusion in the compiler for better overlap

Example

Single-Dimension Partition

- Partitions along one dimension: activations [B/N,F], weights [F,H/N].
- **AllGather** is needed to restore full weights before each Einsum.
- Traditional approach: compute waits until all communications are done (sequential).

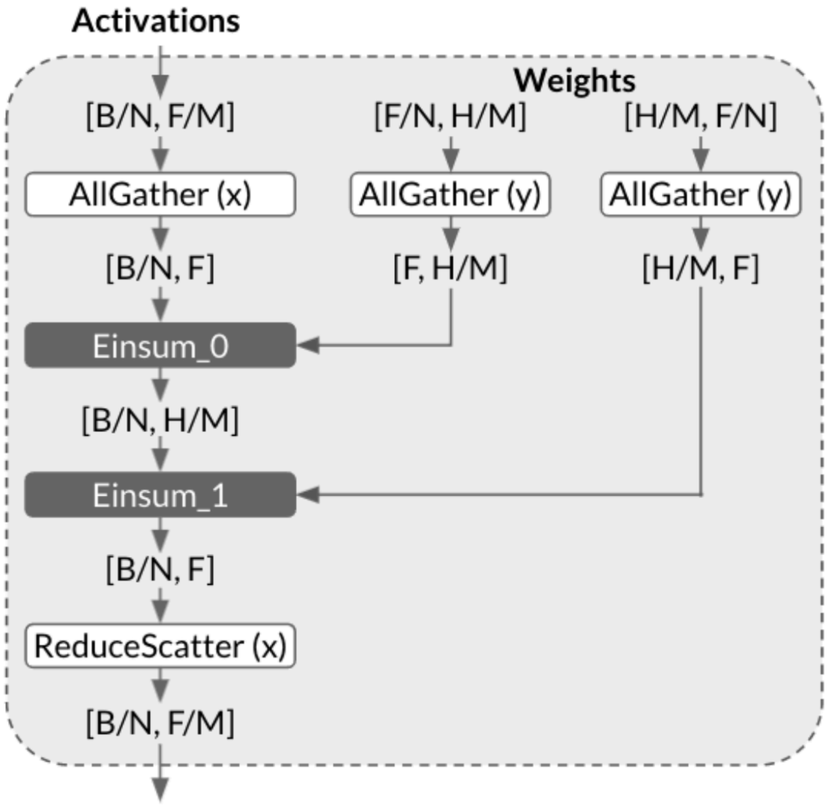


Forward-propagation of a two-layer MLP with N-way partitioning along one dimension.

Example

Two-Dimension Partition

- Device mesh of shape [M,N].
- Tensors are split along both x and y dimensions, e.g. [B/N,F/M].
- Multiple AllGathers and a ReduceScatter can be involved, increasing communication cost.
- Lays the foundation for more complex overlap.

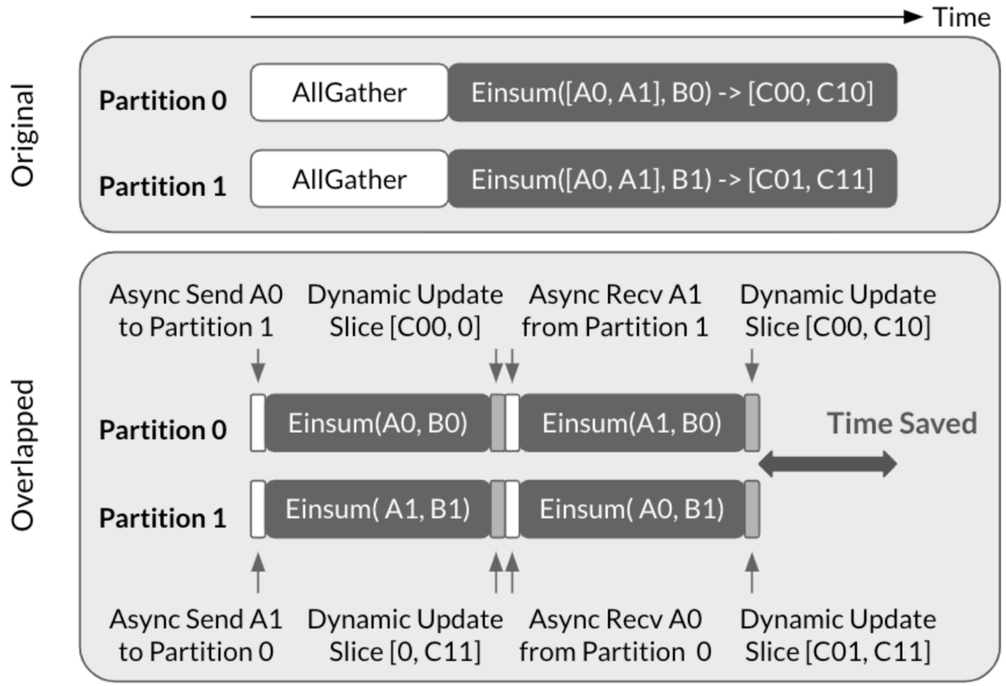


Forward-propagation of a two-layer MLP with N*M-way partitioning along two dimensions.

Overlapping AllGather + Einsum

- Original: Must finish AllGather first, then do the full Einsum.

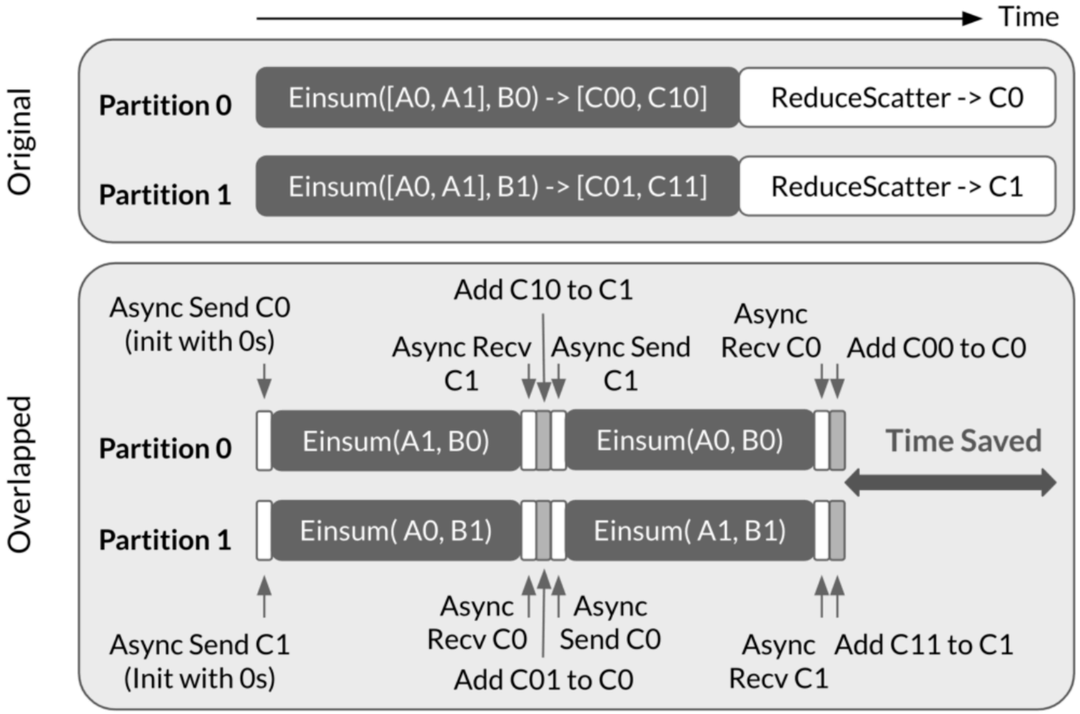
- Overlapped: Decompose communication and computation into partial steps. Process one data shard at a time and use asynchronous collectives so that communication of the next shard overlaps with current shard's computation.



An illustrative example on the key idea of AllGather-Einsum with 2-way intra-layer model parallelism.

Overlapping Einsum + ReduceScatter

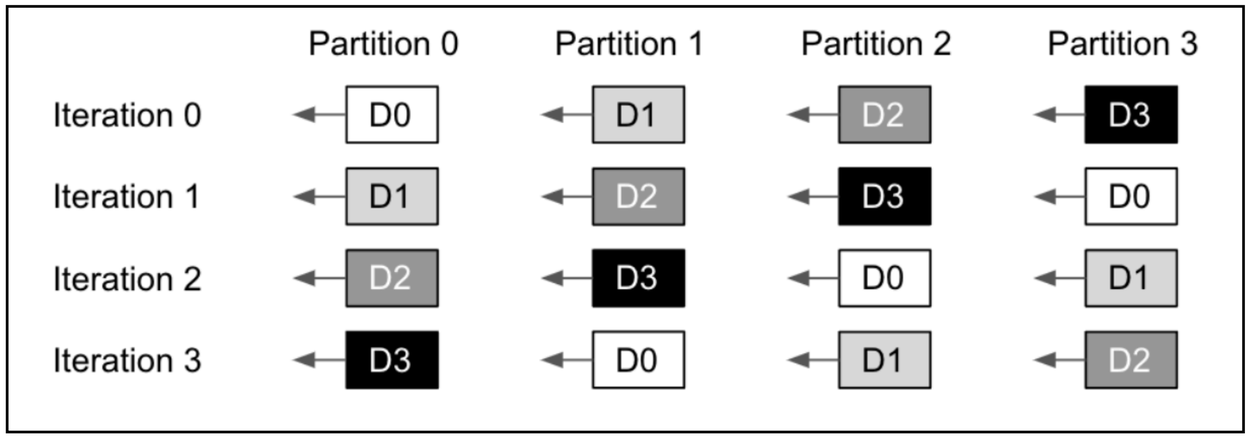
- Original: Compute the entire Einsum result, then run a ReduceScatter.
- Overlapped:
- Split the result into partial shards.
- Start sending out the partial result shards asynchronously while computing the next portion of the Einsum.
- Accumulate (Add) each partial result with the shard received from other devices.



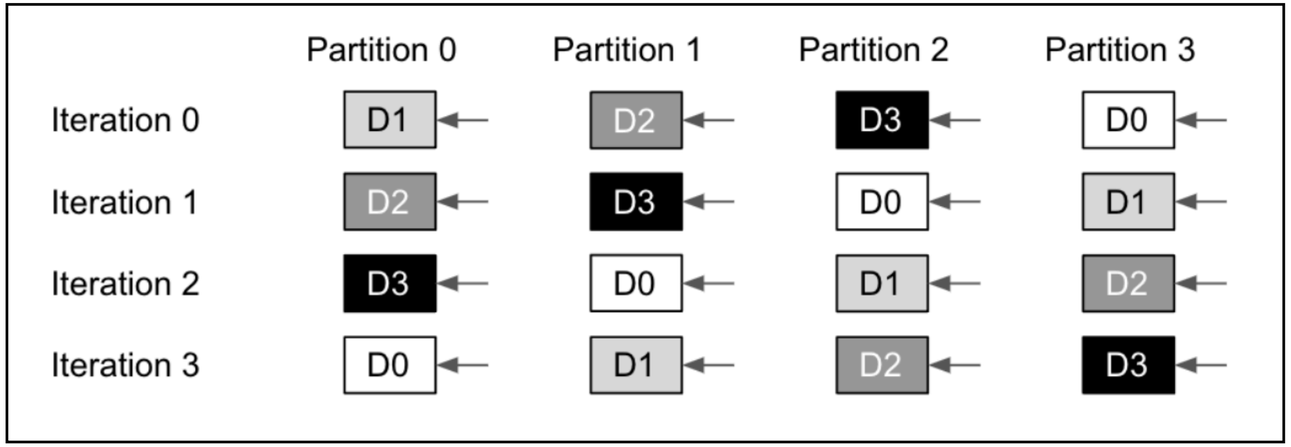
An illustrative example on the key idea of Einsum-ReduceScatter with 2-way intra-layer model parallelism.

Looped Communication Patterns

- The original communication-computation pair is transformed into a loop with N iterations, where N is the number of partitions along the relevant dimension.
- Each partition processes or accumulates one shard, while the next shard is transferred.



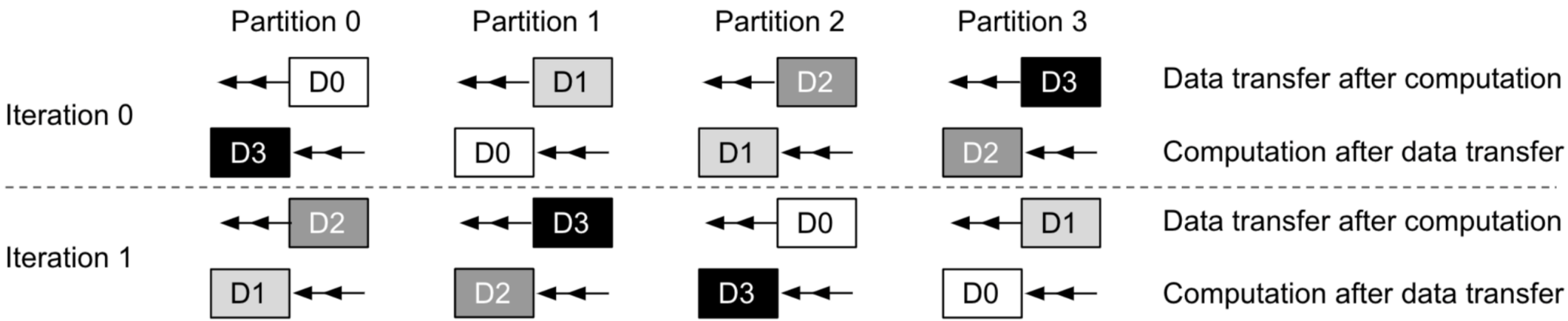
An illustrative example of data shard transfer in a looped AllGather-Einsum with 4-way intra-layer model parallelism.



An illustrative example of data shard transfer in a looped Einsum-ReduceScatter with 4-way intra-layer model parallelism.

Loop Unrolling to Increase Overlap

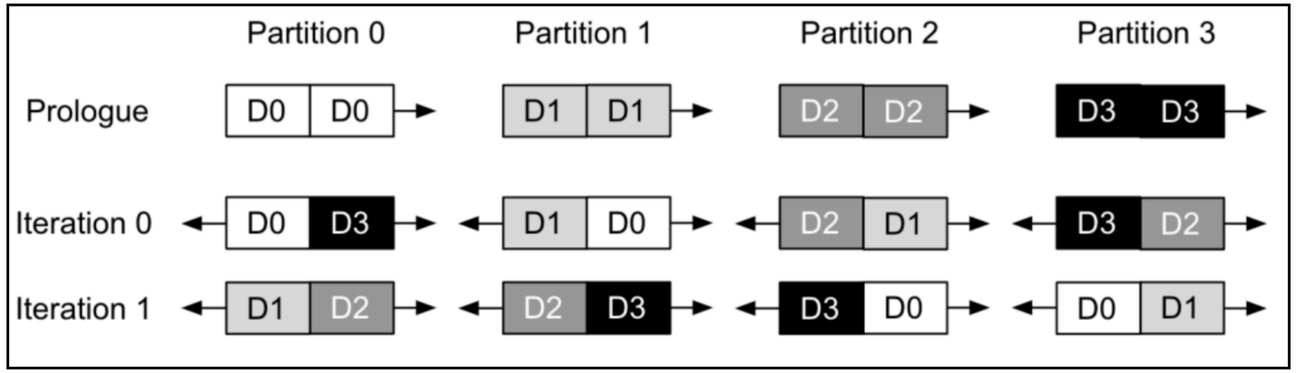
- Unrolling the loop means handling two shards per iteration (double-buffering effect).
- One shard can be in computation while another shard is in communication.
- Further reduces idle time, increasing overlap.



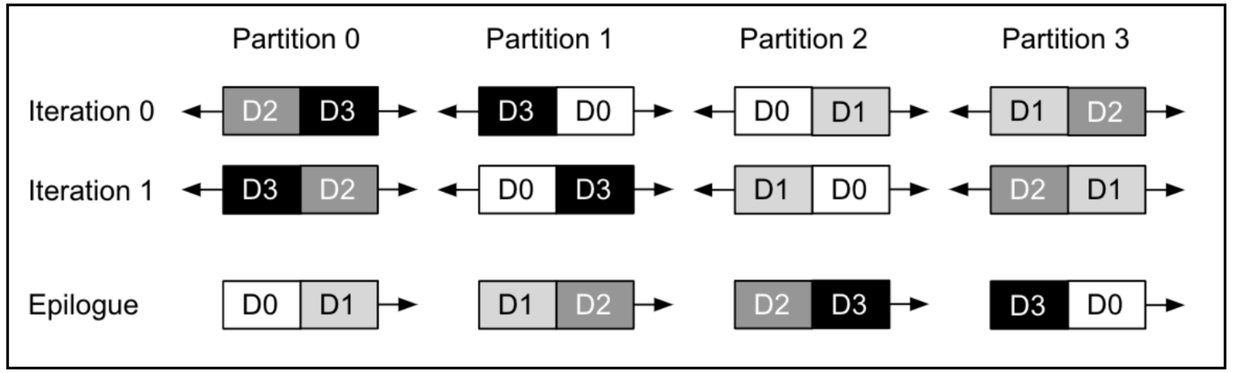
An illustrative example of a looped Einsum-ReduceScatter with unrolling degree of 2 and 4-way intra-layer model parallelism.

Bidirectional Communication

- Challenge: Small computation cannot overlap with communication
- Solution: Process two blocks simultaneously with bidirectional transfer
- Benefit: 2x computation per iteration + full bandwidth utilization



An illustrative example of data shard transfer in a looped AllGather-Einsum with bidirectional data communication and 4-way intra-layer model parallelism.



An illustrative example of data shard transfer in a looped Einsum-ReduceScatter with bidirectional data communication and 4-way intra-layer model parallelism.

Experimental Setup

Models

- GPT_1T (1 trillion params)
- Meena_500B, MLPerf_200B, T5_300B
- GLaM_1T (MoE architecture)
- BigSSL_10B (ASR model)

Hardware: TPU v4 Pods

- Different model sizes use 128~2048 TPU chips.
- XLA compiler automatically handles sharding & scheduling.

Overall Results

Speedup: 1.14 - 1.38× over the baseline

FLOPS utilization:

- Dense models (e.g., GPT, Meena, MLPerf) reach 60-72%
- T5, GLaM, BigSSL see slightly lower due to residual communication patterns or smaller partial computation.

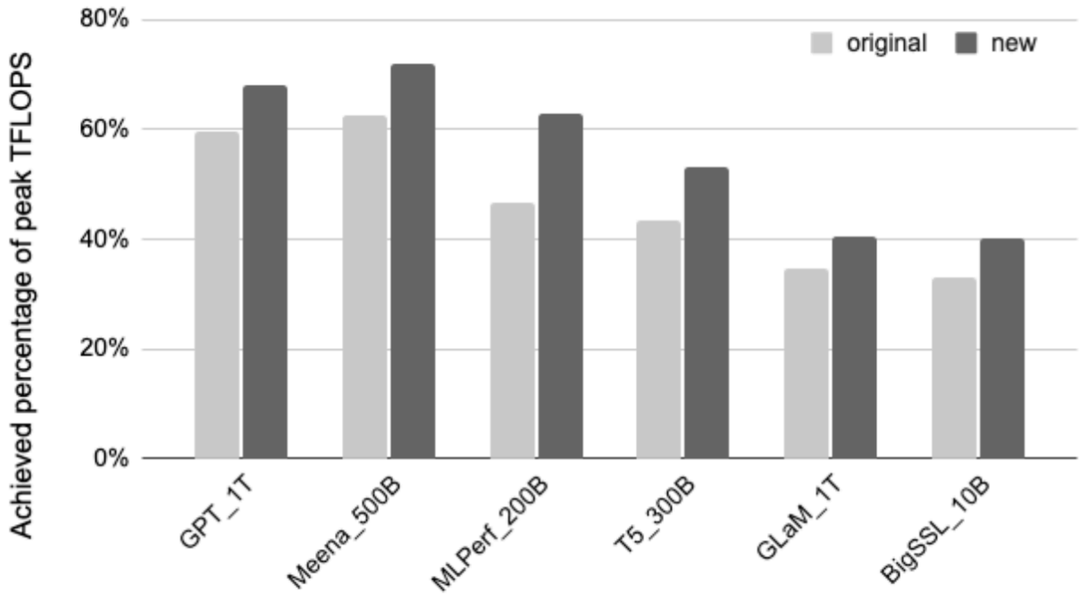


Figure 12: Performance of the evaluated applications.

Weak Scaling with GPT

GPT Variants: 32B, 128B, 200B, 500B, 1T parameters

Method

- Each model runs on appropriately sized TPU v4 Pod.

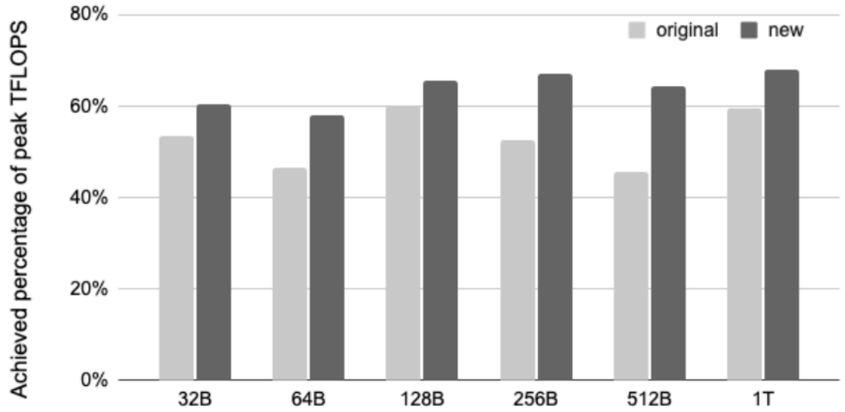


Figure 13: Performance of the weakly scaled GPT models.

- Compare baseline vs. overlapped approach as model size grows.

Outcome

- Consistent 1.1-1.4× speedups at different scales.
- Overlap strategy maintains similar efficiency despite increasing communication volumes.

Results of Loop Unrolling

- Compared to no unrolling, it reduces overheads and fosters concurrency in Einsum-ReduceScatter scenarios.

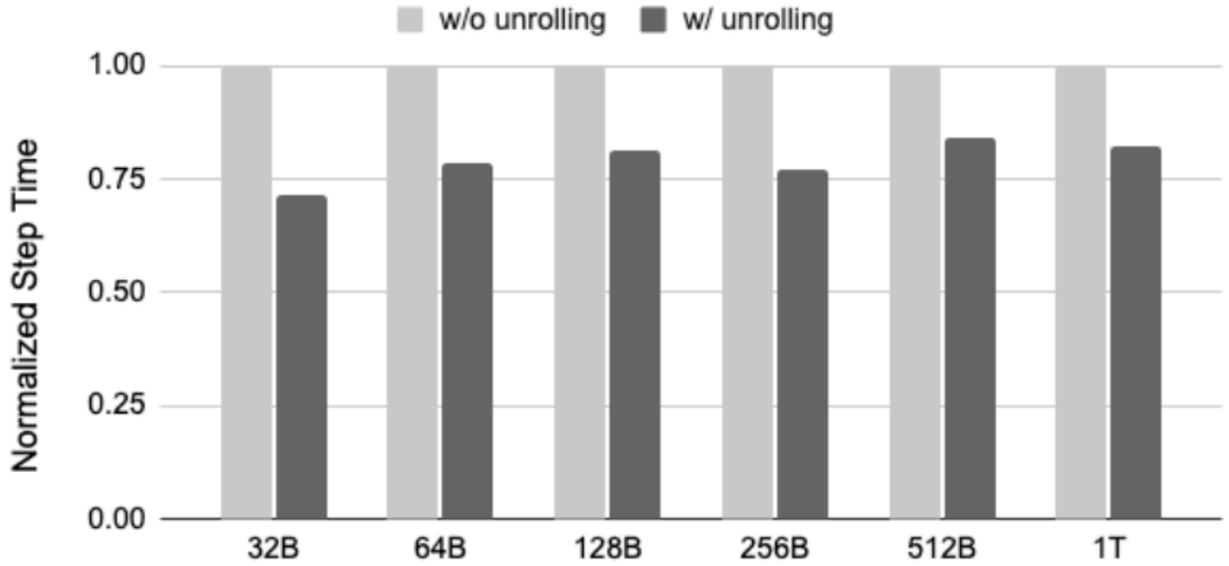


Figure 14: Performance improvements provided by loop unrolling.

Results of Bidirectional Transfer

- Compare unidirectional vs. bidirectional data transfer in large GPT models (GPT_32B to GPT_1T).

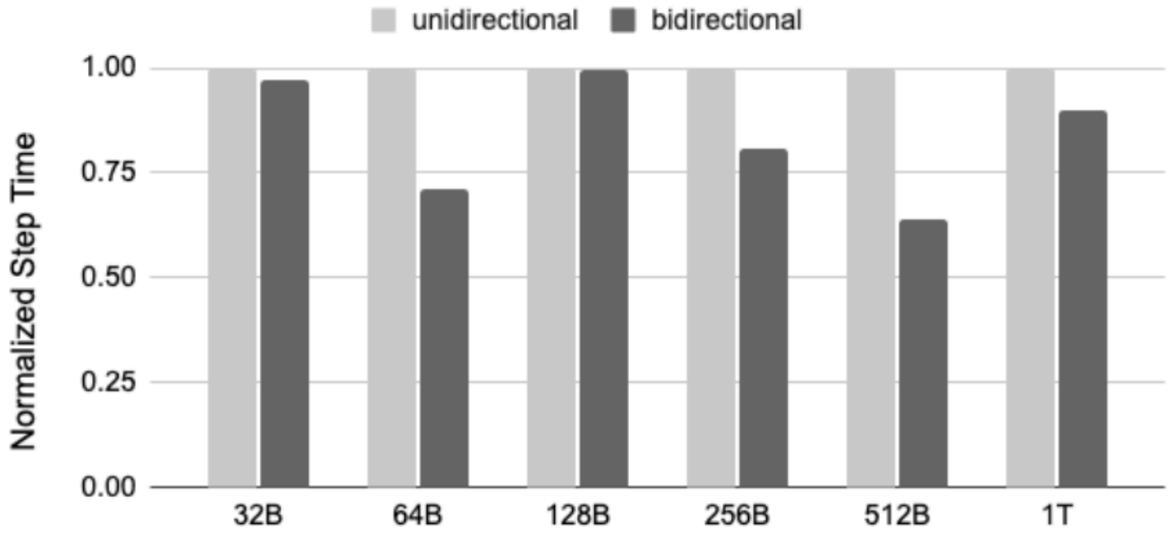


Figure 15: Performance improvements provided by bidirectional data transfer.

Limitations

- Single-Focus Limitation:
  Current work mainly addresses intra-layer (tensor) model parallelism, assuming AllGather/ReduceScatter. Real systems often use pipeline + tensor + data parallelism in tandem.

- No Mixed Parallelism Experiments:
  Lack direct evidence on how our method interacts with pipeline-parallel stages, pipeline "bubbles," or more complex collective primitives (e.g., AllToAll).

Potential Directions:

- Extend overlapping techniques to mixed or multi-stage parallel plans.

- Evaluate interactions with pipeline microbatching and scheduling.