ZeRO: Memory Optimizations

(Zero Redundancy Optimizer)

Adrien Dubois
9/19/2025

Papers I will discuss

- ZeRO: Memory Optimizations Toward Training Trillion Parameter Models
- Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Why is this important to learn

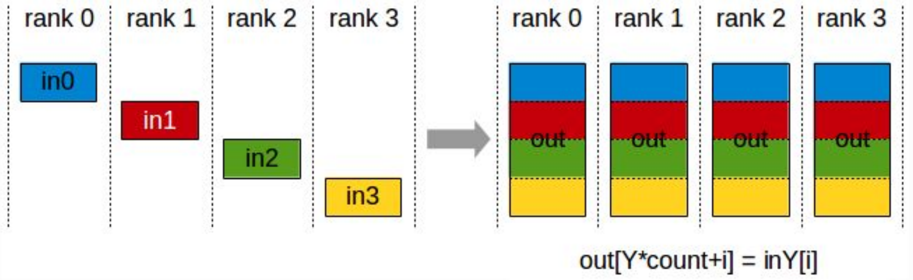
- If you want to train very large models
- If you want to run the biggest open source LLMs
- If you want to work in industry (even just in interviews this is important)

Cuda Distributed Functions:

AllGather

The AllGather operation gathers N values from k ranks into an output buffer of size k*N, and distributes that result to all ranks.

The output is ordered by the rank index. The AllGather operation is therefore impacted by a different rank to device mapping.



AllGather operation: each rank receives the aggregation of data from all ranks in the order of the ranks.

Note: Executing ReduceScatter, followed by AllGather, is equivalent to the AllReduce operation.

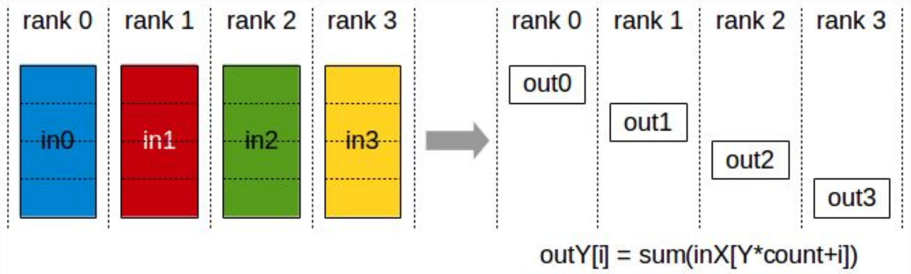
Related links: `ncclAllGather()`.

Cuda Distributed Functions:

ReduceScatter

The ReduceScatter operation performs the same operation as Reduce, except that the result is scattered in equal-sized blocks between ranks, each rank getting a chunk of data based on its rank index.

The ReduceScatter operation is impacted by a different rank to device mapping since the ranks determine the data layout.



Reduce-Scatter operation: input values are reduced across ranks, with each rank receiving a subpart of the result.

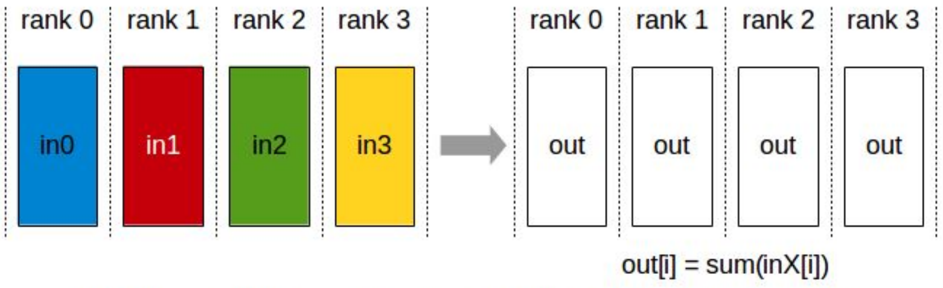
Related links: `ncclReduceScatter()`

Cuda Distributed Functions:

AllReduce

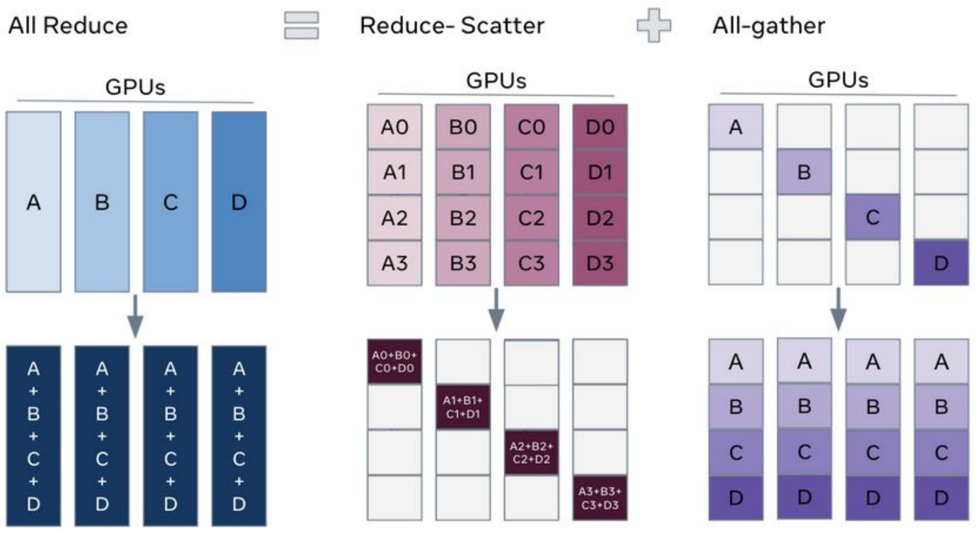
The AllReduce operation performs reductions on data (for example, sum, min, max) across devices and stores the result in the receive buffer of every rank.

In a *sum* allreduce operation between *k* ranks, each rank will provide an array in of N values, and receive identical results in array out of N values, where out[i] = in0[i]+in1[i]+...+in(k-1)[i].



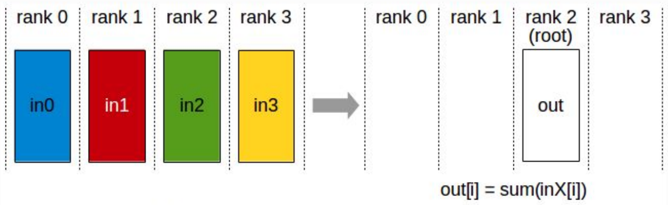
All-Reduce operation: each rank receives the reduction of input values across ranks.

Pytorch Distributed Functions:



Reduce

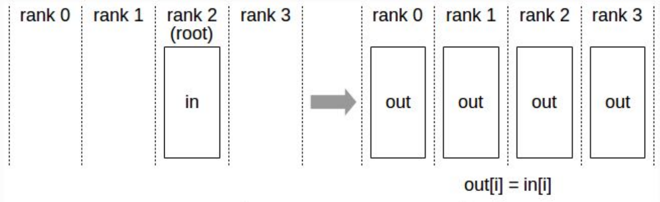
The Reduce operation performs the same operation as AllReduce, but stores the result only in the receive buffer of a specified root rank.



Reduce operation: one rank receives the reduction of input values across ranks.

Broadcast

The Broadcast operation copies an N-element buffer on the root rank to all ranks.

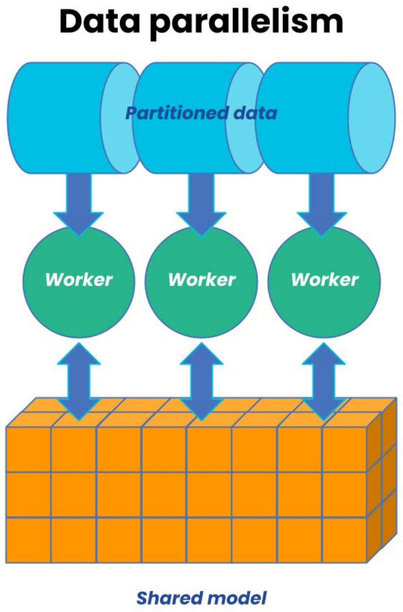


Broadcast operation: all ranks receive data from a "root" rank.

Note: A Reduce, followed by a Broadcast, is equivalent to the AllReduce operation.

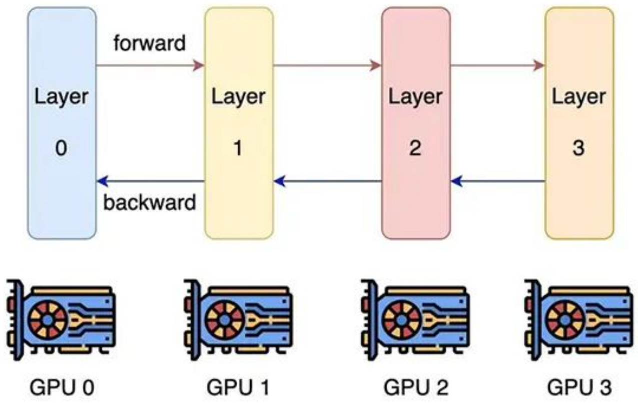
Data Parallelism (DP)

- Each GPU has the full weights, optimizer states
- Data is split between GPUs
- Requires 2 (main) points of synchronization
- Good compute/communication efficiency
- Poor memory efficiency



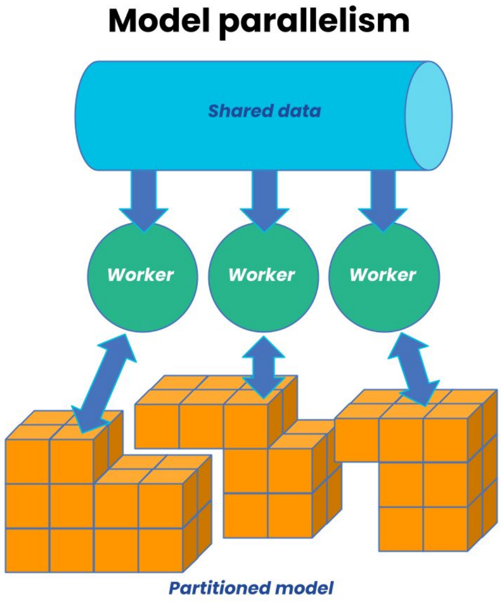
Pipeline Parallelism

- Split the model into sequential stages
- Pros:
  - Peak memory reduction
  - Good for inference
- Cons:
  - Slow for training (only 1 GPU used at a time for baseline implementation)
  - Pipeline Bubbles (idle time at inference)
  - Ramp up time for continuous inference



Model Parallelism

- Aka Tensor Parallelism
- Splits layers across different GPUs.
- Each GPU gets the full data
- As far as I know, this was first proposed in:
    - Megatron-LM by Nvidia



Full runtime diagram

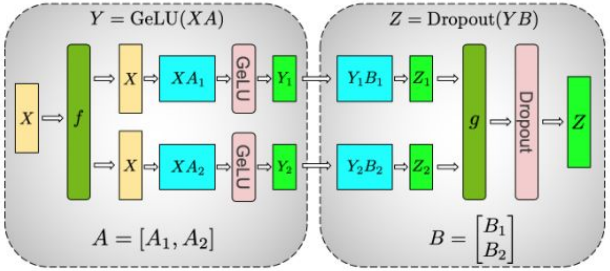
- Linear layer & Self-Attention require 2 synchronization functions per pass:

Forward pass:

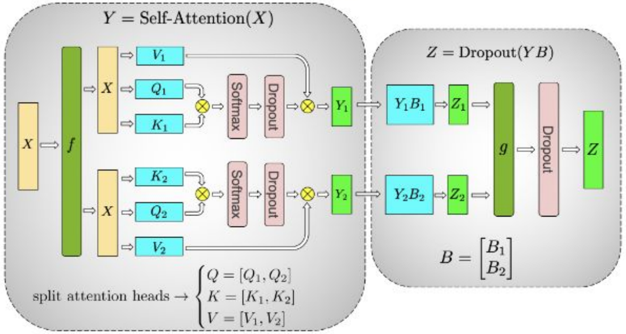
- f is an identity operator (splits X vertically into n_d vectors)
- g is an all-reduce (moves local outputs onto both gpus)

Backward pass:

- g is an identity operator (splits activation vertically into n_d vectors)
- f is an all-reduce (moves local param updates onto both gpus)



(a) MLP



(b) Self-Attention

How it doesn't work:

- **Horizontal Splitting** across layer parameters
  - Split weight matrix A along its rows and input X along its columns

$$\mathrm{Y} = \mathrm{GeLU}(X_1A_1 + X_2A_2)$$
$$\mathrm{GeLU}(X_1A_1 + X_2A_2) =/= \mathrm{GeLU}(X_1A_1) + \mathrm{GeLU}(X_2A_2)$$

Normal Linear Layer	**Horizontal-wise MP Linear Layer:**
$X = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$ $A = \begin{bmatrix} 1,2 \\ 3,4 \end{bmatrix}$ $Y = ReLU\left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1,2 \\ 3,4 \end{bmatrix}\right)$ $= ReLU\left(\begin{bmatrix} 7 & 10 \\ 1 & 2 \end{bmatrix}\right)$	$X = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \rightarrow X_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, X_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ $[A_1, A_2] = \begin{bmatrix} [1,2] \\ [3,4] \end{bmatrix}$ $[Y_1, Y_2] = \left[ReLU\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} [1,2] + \begin{bmatrix} 2 \\ 0 \end{bmatrix} [3,4]\right)\right]$ $= \left[ReLU\left(\begin{bmatrix} 1,2 \\ 1,2 \end{bmatrix} + \begin{bmatrix} 6,8 \\ 0,0 \end{bmatrix}\right)\right]$

How it works:

- **Vertical Splitting** across layer parameters
    - Just split A along its columns

Normal Linear Layer	Column-wise MP Linear Layer:
$X = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$ $A = \begin{bmatrix} 1, 2 \\ 3, 4 \end{bmatrix}$ $Y = ReLU\left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1, 2 \\ 3, 4 \end{bmatrix}\right)$ $= ReLU\left(\begin{bmatrix} 7 & 10 \\ 1 & 2 \end{bmatrix}\right)$	$X = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$ $[A_1, A_2] = \left[\begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix}\right]$ $[Y_1, Y_2] = \left[ReLU\left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix}\right), ReLU\left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix}\right)\right]$ $= \left[ReLU\left(\begin{bmatrix} 7 \\ 1 \end{bmatrix}\right), ReLU\left(\begin{bmatrix} 10 \\ 2 \end{bmatrix}\right)\right]$

Transformer Layer Splitting

- A transformer block needs 4 points of synchronization:
  - 2 for self-attention
  - 2 for FFN

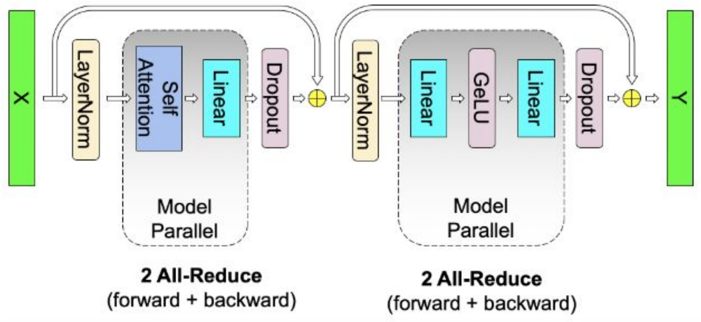


Figure 4. Communication operations in a transformer layer. There are 4 total communication operations in the forward and backward pass of a single model parallel transformer layer.

Pros and Cons

Pros:

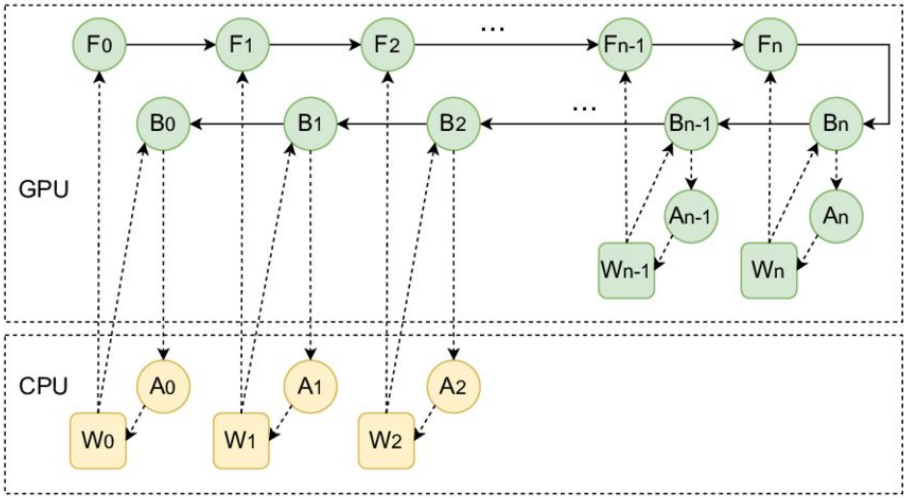
- Train larger models!
- Good memory efficiency

Cons:

- Poor compute/communication efficiency
- Reduces the granularity of computation
- Significant costs when working between GPU nodes

CPU-Offloading

- Move some of the computation/memory to CPU
- Prefetch memory back before it is needed
- Cons:
    - It is very slow



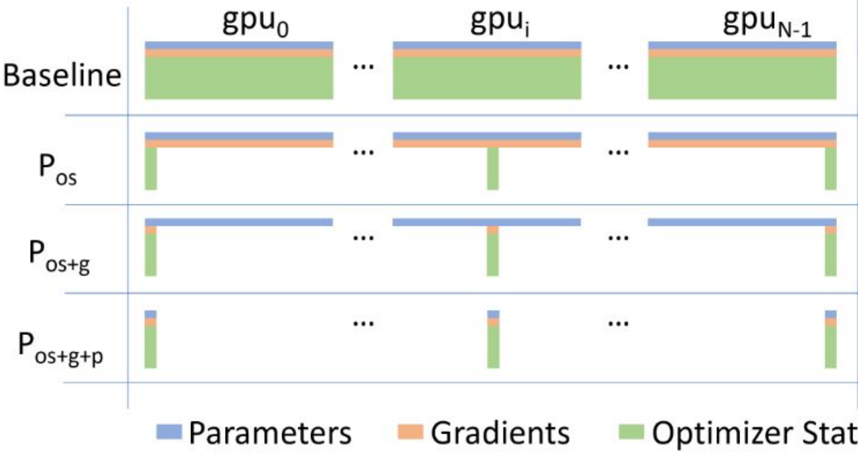
Optimization Model State Memory

Other methods require the full model parameters to be loaded, even when they aren't in use

We will go through these advances in 3 steps:

- Optimizer State Partitioning (P_os): 4x memory reduction, same communication volume as DP
- Add Gradient Partitioning (P_{os + g}): 8x memory reduction, same communication volume as DP
- Add Parameter Partitioning (P_{os+g+p}): Memory reduction is linear with DP degree N_d. 50% increase in communication volume (they say it is a minor increase).

Optimization Model State Memory



Memory Consumed	K=12 $\mathbf{\Psi}$=7.5B N_d=64	K=12 Ψ=7.5B N_d=4
$(2+2+K)*\mathbf{\Psi}$	120GB	120
$2\mathbf{\Psi}+2\mathbf{\Psi}+\frac{K*\mathbf{\Psi}}{N_d}$	31.4GB	52.5
$2\mathbf{\Psi}+\frac{(2+K)*\mathbf{\Psi}}{N_d}$	16.6GB	41.25
$\frac{(2+2+K)*\mathbf{\Psi}}{N_d}$	1.9GB	30 +50% comm costs

K=12?

- Optimizer States require:
  - Fp32 copy of the parameters (4Ψ)
  - Fp32 copy of momentum for every parameter (4Ψ)
  - Fp3 copy of variances for every parameter (4Ψ)
- Even if you are doing mixed precision training, you add at minimum:
  - Fp16 copy of params (2Ψ)
  - Fp16 copy of the gradients (2Ψ)
- So for Adam related parameters: K=12, + 4 for forward computation with mixed precision.
- Benefit: You save a lot on activation memory
  - Activation memory is proportional to: transformer layers × hidden dimensions × sequence length × batch size

Optimizing Residual Memory

Partitioned activation checkpointing (Pa):

- Goal: reduce replicated memory
- Offload activation partitions to CPU for extremely large models if needed

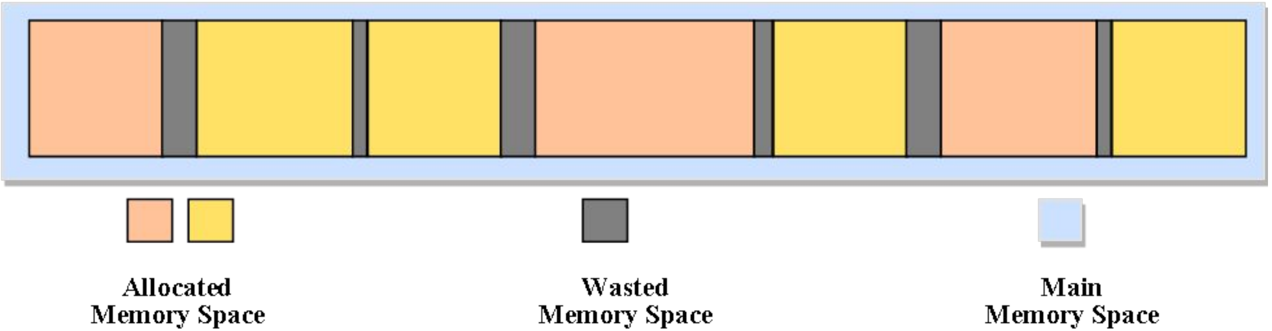
Constant-size temporary buffers:

- We define fixed size buffers for temporary information to prevent OOM
- Tradeoff: more synchronization points

Optimizing Residual Memory (part 2)

Memory defragmentation:

- Memory fragmentation occurs from short/long-lived tensors getting interleaved in memory
- Buffers need to be contiguous, so holding specific parts of memory hostage long-term can block new allocations



Speedup from ZeRO

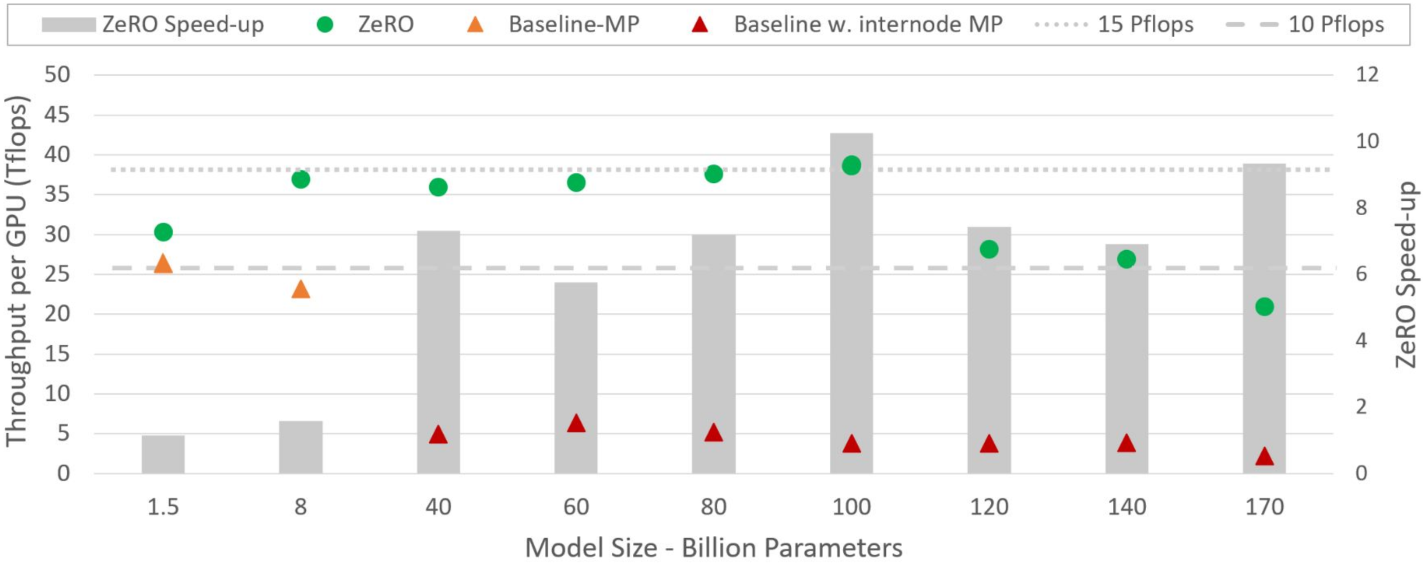


Figure 2: *ZeRO* training throughput and speedup w.r.t SOTA baseline for varying model sizes. For *ZeRO*, the MP always fit in a node, while for baseline, models larger than 40B require MP across nodes.

Recap

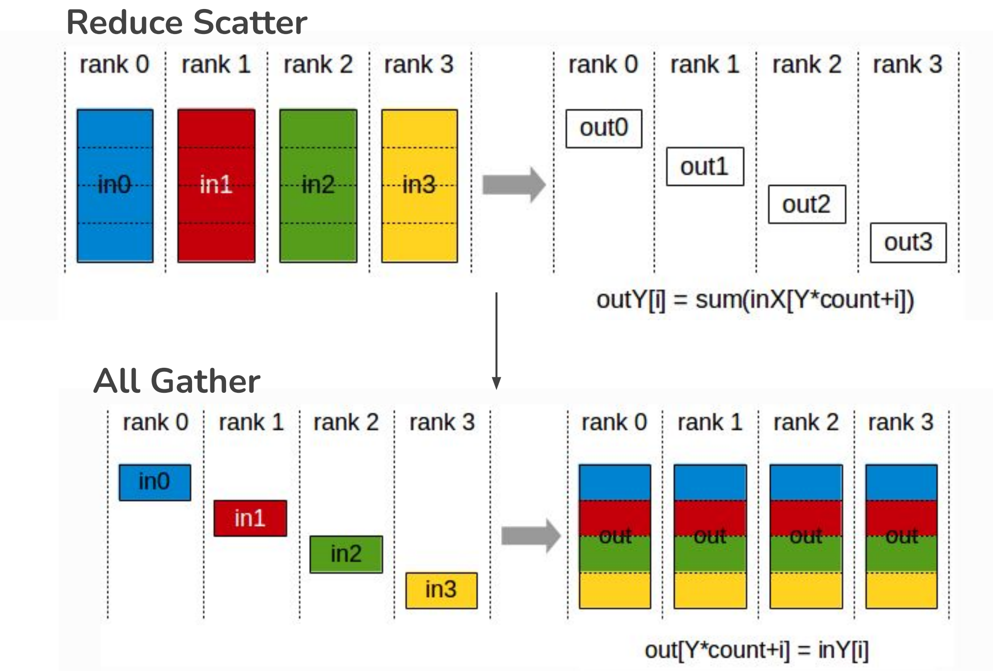
- ZeRO-3 reduces peak memory usage via reduce-scatter
- Largest full layer + partitions must still fit on one GPU

Communication Costs: D.P. (baseline)

- Standard DP requires 2 communication points (technically 3):
  - 1 reduce-scatter operation on the data (small)
  - 1 **all-reduce** on the gradients (depending on the gradient accumulation steps, this could be infrequent)

So: 2Ψ

Note: technically this is $2*(p-1)/p*\Psi$ where p is num_devices. The authors don't mention this approximation, but since they work with p=64-1024+ GPUs it is close enough

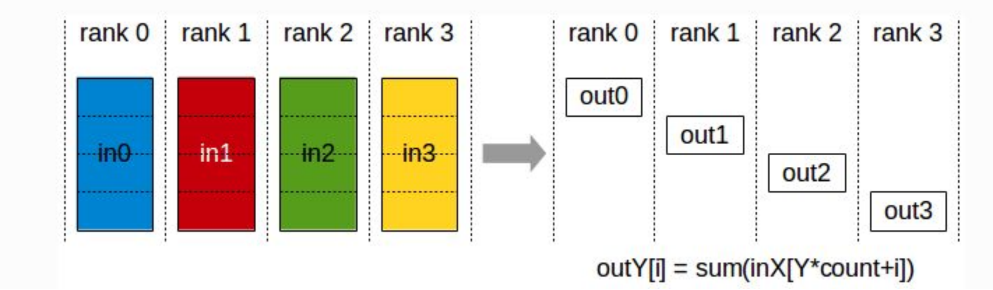


Communication Costs: P_{os+g}

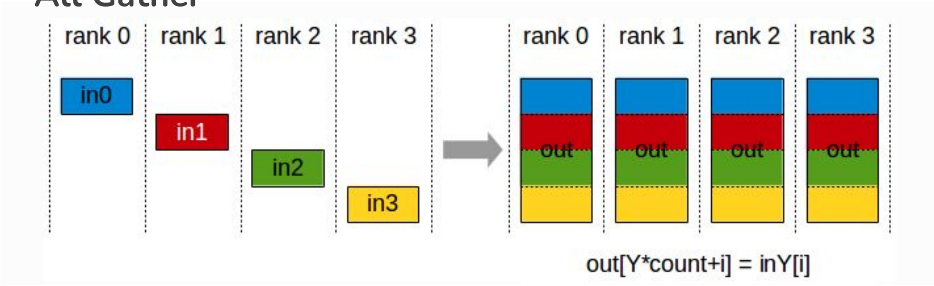
- Replaces gradient all reduce with scatter-reduce
  - Gradients get partitioned on each device
- Parameters all-gathered after local updates
  - Parameters must be gathered to run each layer

Same as DP: 2Ψ

Reduce Scatter



All Gather

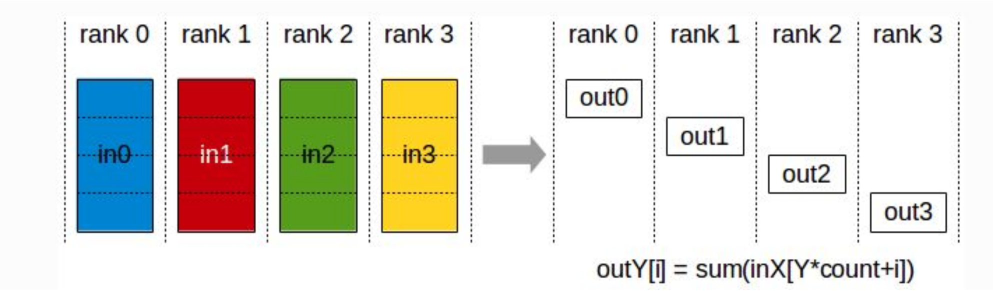


Communication Costs: P_{os+g+p} (+Params)

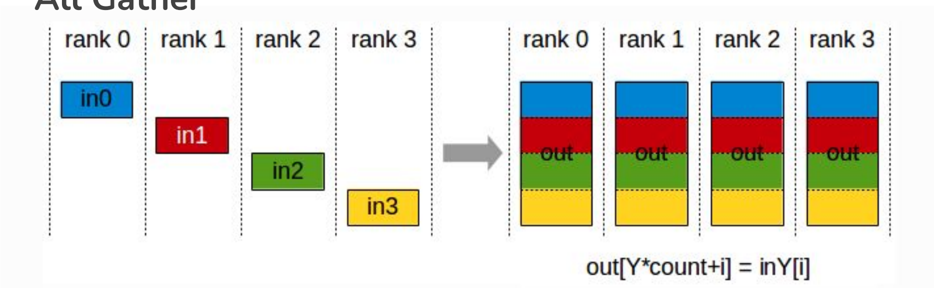
- Parameters are partitioned across devices as well.
  - All gather during forward prop
    - Extra params are discarded after computation
  - All gather during backward prop
    - Extra params are discarded after
- Gradients are reduce-scatters
  - You want to re-partition them after

Total Volume: 3Ψ (1.5x)

Reduce Scatter



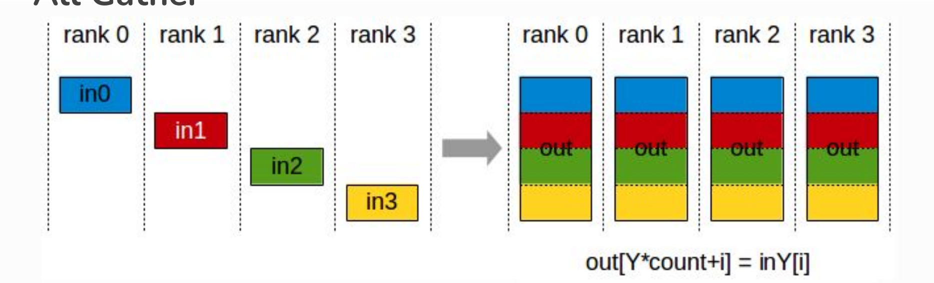
All Gather



Communication Costs: P_{a} Activations

- Additional all-gather before forward recomputation
- This all-gather is not on all the parameters, only the activations:
    - hidden dimension × sequence length
- Memory cost for activations is reduced by a factor of the number of devices
- You can greatly increase batch sizes

All Gather



Thanks!