Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

Deepak Narayanan‡, Mohammad Shoeybi† , Jared Casper† , Patrick LeGresley† , Mostofa Patwary† , Vijay Korthikanti† , Dmitri Vainbrand† , Prethvi Kashinkunti† , Julie Bernauer† , Bryan Catanzaro† , Amar Phanishayee∗ , Matei Zaharia‡
†NVIDIA ‡Stanford University ∗Microsoft Research

Motivation

- Cannot fit the model parameters in a single GPU (min. 2.2 TB memory to train 175B model)
- Training requires a high number of compute operation
- Results in **unrealistically long** training time * (≈ 288 years for 175 B model on 1 GPU!)
- Solution : Parallelism

Problem

- Using parallelism methods in isolation limits scaling
  - Pipeline: long GPU idle time depending on pipeline schedule
  - Tensor: Slow when deployed across multi-GPU servers
  - Data: # GPUs limited to the batch size

PTD-P Contribution

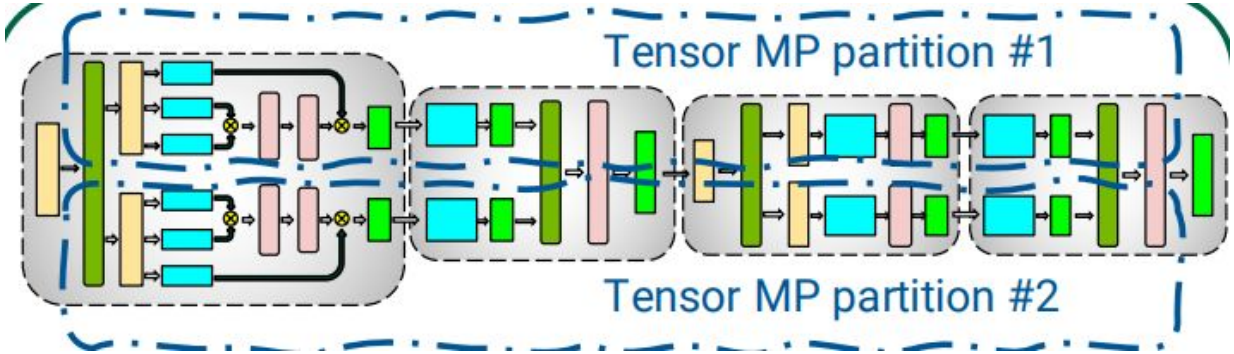
- Combine
  - Pipeline
  - Tensor
  - Data
- Propose novel *interleaved* pipeline schedule
- Show non-trivial interactions between all 3

Data Parallelism

- Each GPU(or cluster of GPU) hosts the **full** model
- Data is sharded
- Gradients periodically aggregated

Tensor Model Parallelism

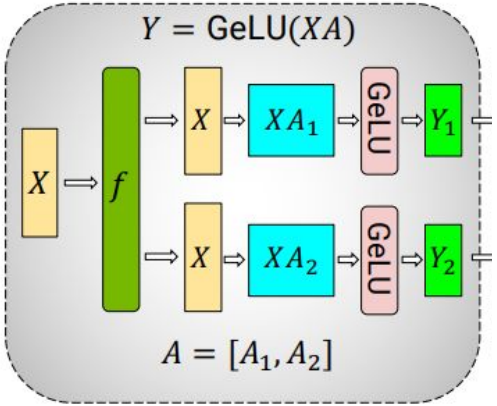
- Layers themselves are split over devices



MLP

$$Y = \mathrm{GeLU}(XA). \quad Z = \mathrm{Dropout}(YB).$$

$$[Y_1, Y_2] = [\mathrm{GeLU}(XA_1), \mathrm{GeLU}(XA_2)]$$

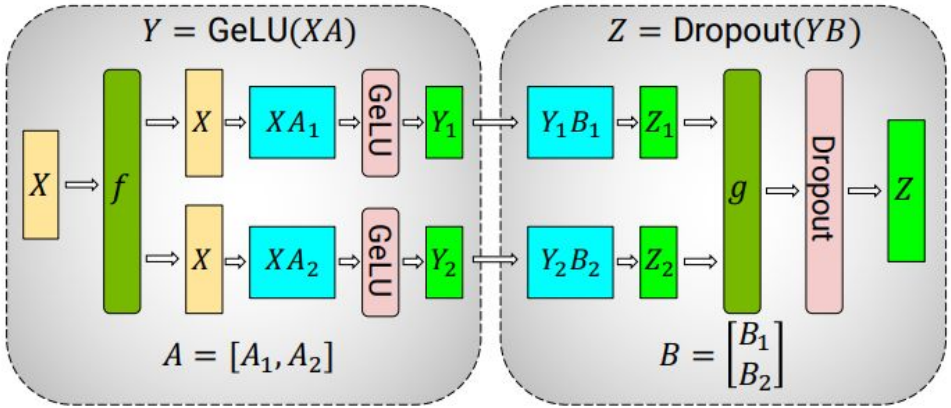


MLP

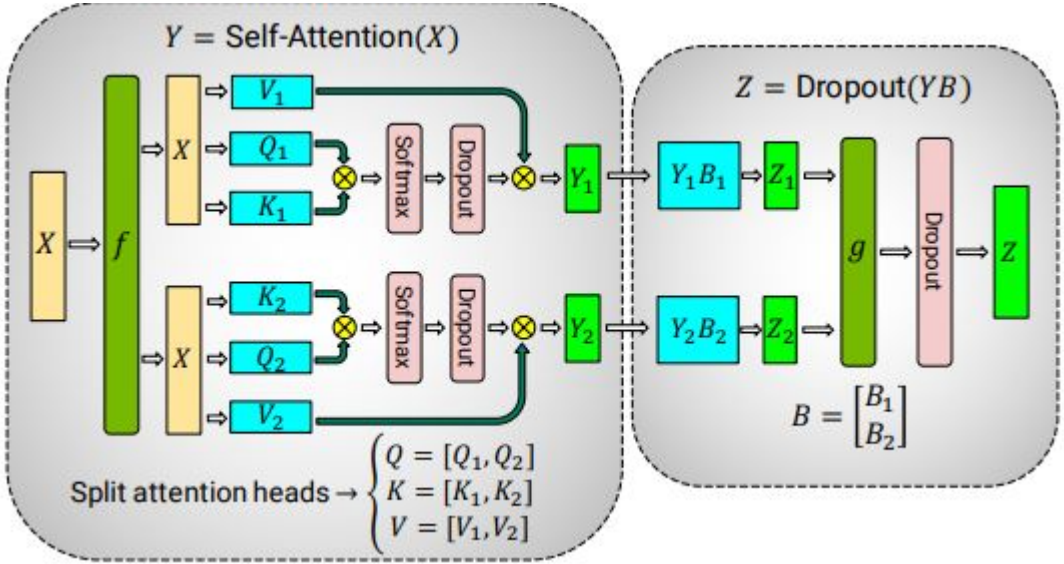
$$Y = \mathrm{GeLU}(XA). \quad Z = \mathrm{Dropout}(YB).$$

$$[Y_1, Y_2] = [\mathrm{GeLU}(XA_1), \mathrm{GeLU}(XA_2)]$$

$$B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \; Y = [Y_1, Y_2].$$



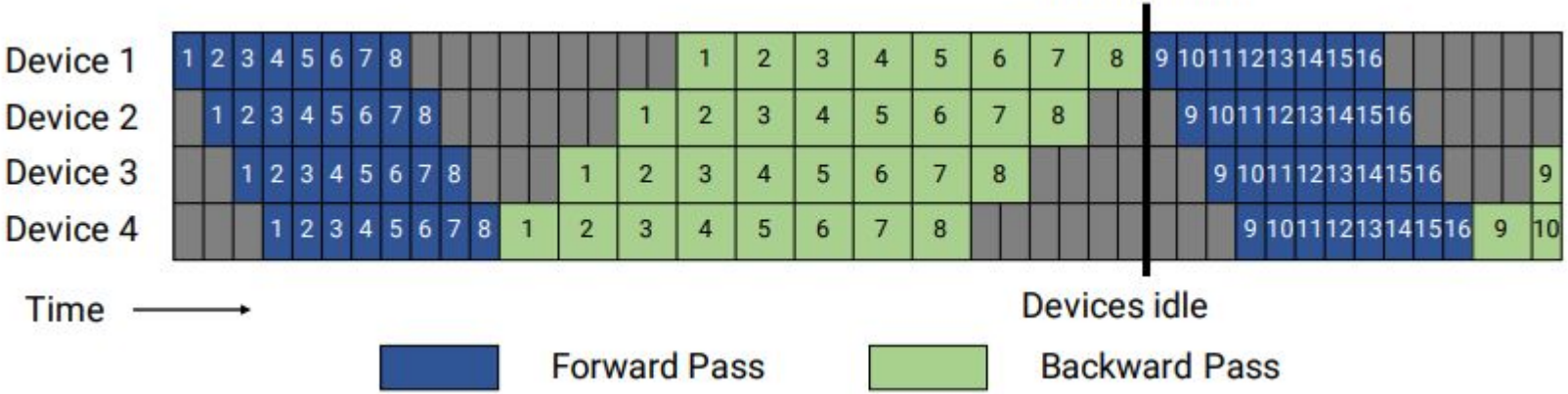
Attention

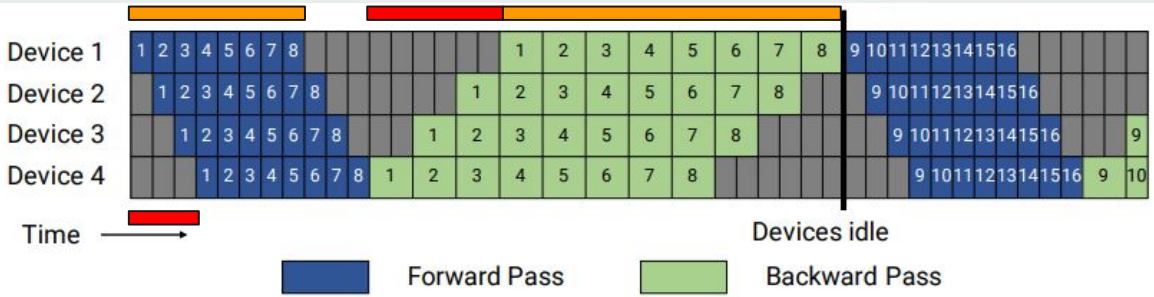

Y = Self-Attention(X)
X
f
X
V_1
Q_1
K_1
Softmax
Dropout
Y_1
K_2
Q_2
V_2
Softmax
Dropout
Y_2
Split attention heads →
$Q = [Q_1, Q_2]$
$K = [K_1, K_2]$
$V = [V_1, V_2]$
Z = Dropout(YB)
Y_1B_1
Z_1
Y_2B_2
Z_2
g
Dropout
Z
$B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

Pipeline Model Parallelism

- Layers of the model are sharded
  - Each device has an equal number of layers
- Batches are split into microbatches
- Periodic *flushes* : synchronization
- Goal : reduce *pipeline bubble* time
  - Idle GPU time

Default Schedule (Gpipe)





t_{pb} = time of the pipeline bubble

p = #devices

t_f = microbatch's forward pass time

t_b = microbatch's backward pass time

t_{id} = ideal time per iteration

m = number of microbatches

$$t_{pb} = (p-1)\cdot(t_f + t_b)$$

$$t_{id} = m\cdot(t_f + t_b)$$

$p = \#\text{devices}$

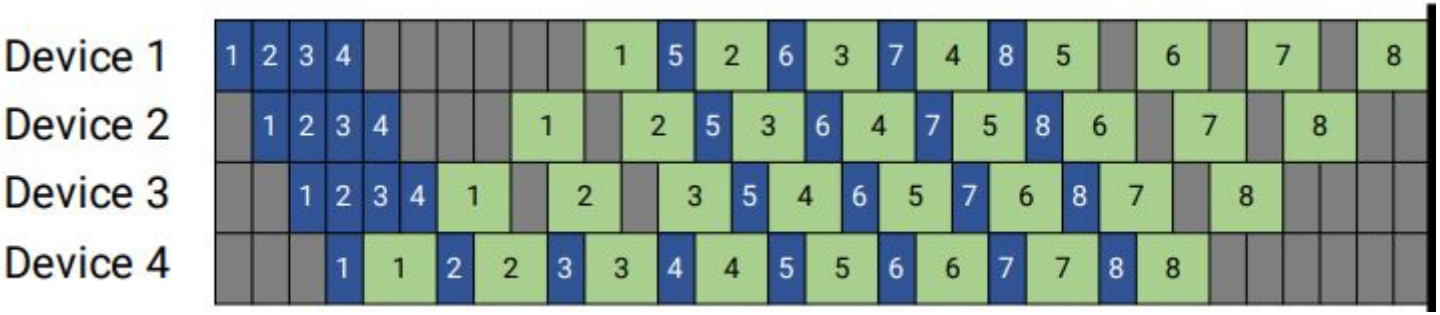
$m = \text{number of microbatches}$

$$\text{Bubble time fraction (pipeline bubble size)} = \frac{t_{pb}}{t_{id}} = \frac{p-1}{m}$$

- Need m >> p
- Large m -> high memory footprint
  - Requires caching intermediate activations for all *m* iterations for the backward pass

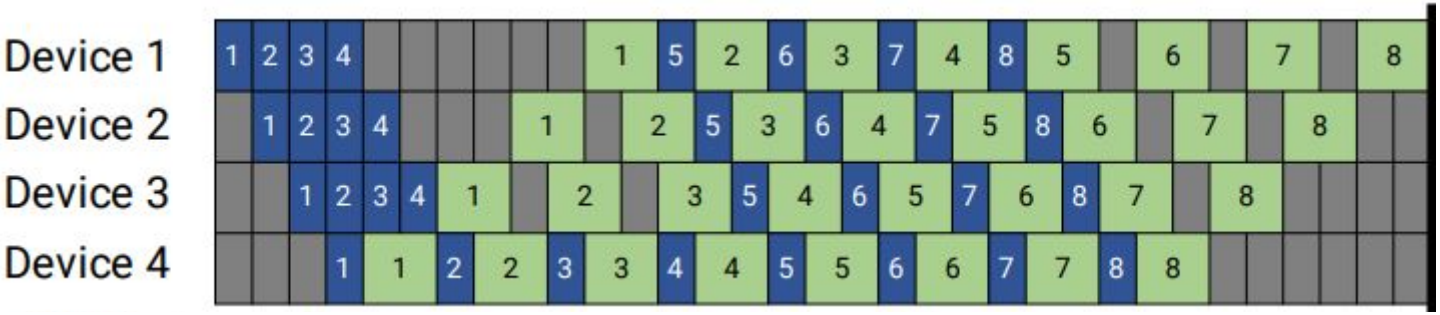
PipeDream-Flush schedule

- 1 forward, 1 backward



PipeDream-Flush schedule

- Advantage
  - Activations are stashed for *p* or fewer microbatches
  - More memory-efficient when *m* >> *p*



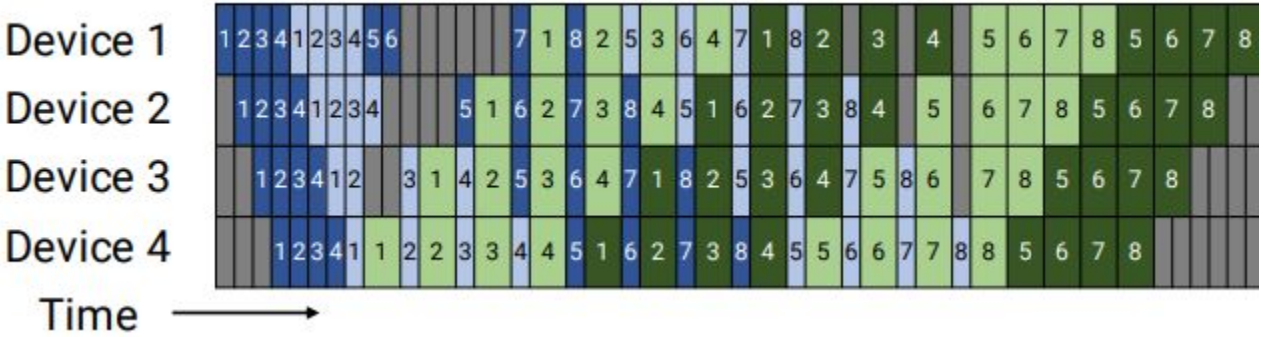
Schedule with Interleaved Stages

Forward Pass Backward Pass

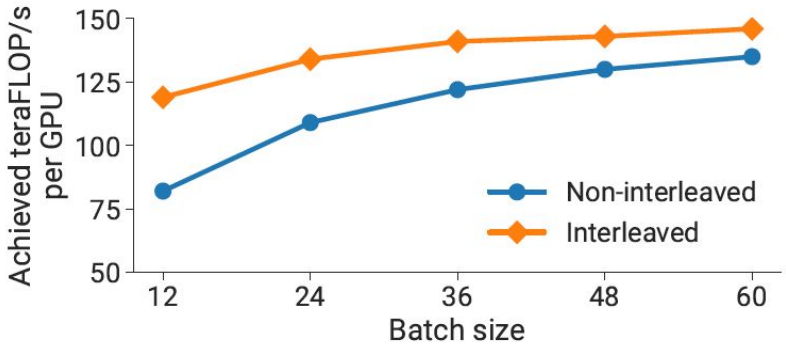
- Each model has multiple subset of layers (model chunk)
  - In
- 1F1B s

$$t_{pb}^{\text{int.}} = \frac{(p-1)\cdot(t_f+t_b)}{v}$$

$$\frac{1}{v} \cdot \frac{p-1}{m}$$



Interleaved vs. Non-interleaved: empirical result



- ✅ **Interleaved schedule** achieves higher throughput overall, especially with smaller batch size.
- **Non-interleaved schedule** is similarly performant with higher batch size (=smaller pipeline bubble size)

Code

Trade-offs between configurations

1. Tensor and Pipeline
2. Data and Pipeline
3. Data and Tensor

Quick notation

$$p, t, d = \text{parallelization dimension}$$

$$n = \text{number of GPUs}$$

$$B = \text{Global Batch Size}$$

$$p \cdot t \cdot d = n$$

$$b = \text{Microbatch size}$$

$$m = \frac{B}{bd}$$

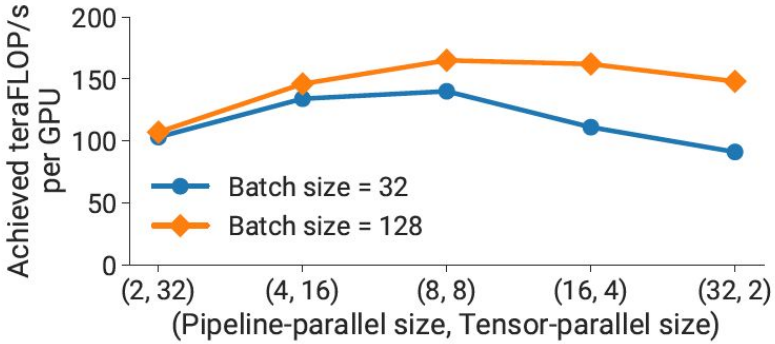
$p, t, d =$ parallelization dimension
$n =$ number of GPUs
$B =$ Global Batch Size
$b =$ Microbatch size
$m = \frac{B}{bd}$

1. Tensor & Pipeline

- Observation
  - Scaling **tensor** parallelism (t) reduces pipeline bubble size, but requires more communication
  - Scaling **pipeline** parallelism (p) increases GPU idle time, but requires fewer communication

Bubble time fraction: $$\frac{p-1}{m} = \frac{n/t - 1}{m}$$

1. Tensor & Pipeline: Empirical result



- ✅ The optimal configuration balances **tensor parallelism** and **pipeline parallelism**.
- ⛔ **Both** have communication overhead, leading to a slowdown in suboptimal combination

Bubble time fraction: $\frac{p-1}{m} = \frac{n/t - 1}{m}$

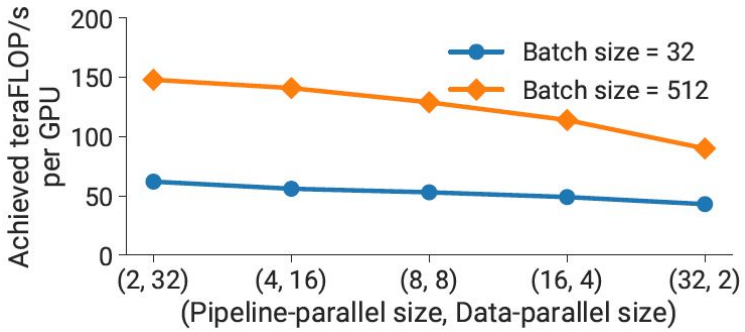
2. Data + Pipeline

- Observation
  - Scaling **data parallelism** (d) also reduces pipeline bubble size
  - Using **more microbatches** (b') is more effective for both data and pipeline parallelism

Bubble time fraction:

$$\frac{p-1}{m} = \frac{n/d - 1}{b'/d} = \frac{n-d}{b'}.$$

2. Data & Pipeline: Empirical result



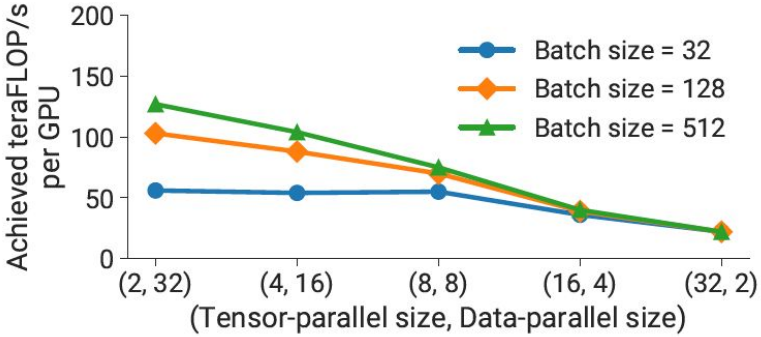
- ✅ Increasing the **data parallel size** decreases the pipeline bubble size, which increases throughput
- 🚫 Increasing the **pipeline parallel** size increases the pipeline bubble size, which increases the idle GPU time

Bubble time fraction: $\frac{p-1}{m} = \frac{n/d - 1}{b'/d} = \frac{n-d}{b'}.$

3. Data & Tensor

- Observation
  - Tensor parallelism requires communication once every microbatch
  - Data parallelism requires communication once every batch

3. Data & Tensor: Empirical result



- Increased communication with **tensor parallelism** decreases GPU utilization

$p, t, d =$ parallelization dimension

Pipeline/Tensor/Data Interaction Takeaway

- Solution
  - Use tensor parallelism within a node (t)
  - Add pipeline parallelism to scale to multiple nodes (t * p)
  - Use data parallelism to scale up training to more GPUs (*t * p * d)*

Training a Trillion Parameter Model

- Parallelism is indispensable to training a large model
- Tensor, pipeline, data parallelism can be combined effectively
  - GPT-3 (175 billion parameter model on 300 billion tokens, 1024 A100 GPUs) ≈ **34 days**
  - 1 trillion parameter model on 450 billion tokens, 3072 A100 GPUs ≈ **84 days**
- To put in perspective,
  - 1024 GPU * 34 days * 24 hours = 835,584 hours
  - 3072 GPU * 84 days * 24 hours = 6,193,152 hours

Thank you for listening

If you're interested, the paper also touches upon:

- Communication optimization
- Activation recomputation
- Comparison with ZeRO
- Microbatch size