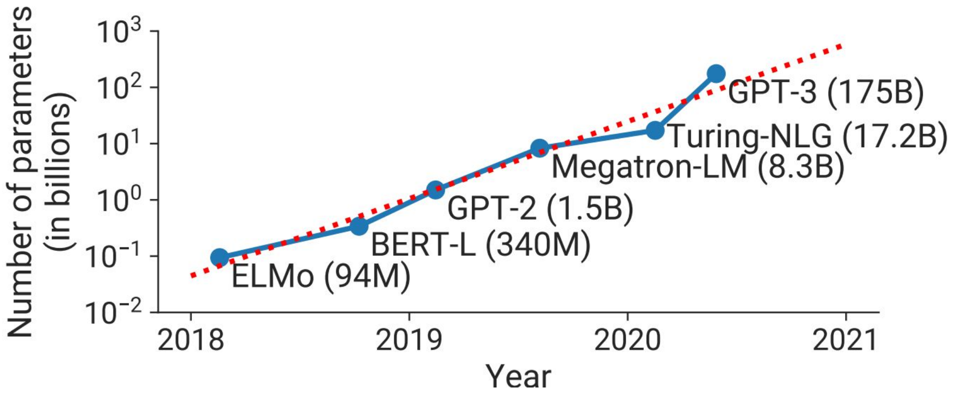
Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

D. Narayanan et al.
Nvidia, Stanford University, Microsoft Research
SC'21

Presented by
Qinjun Jiang, Tong Wei

Motivation And Background

- **Why do we need large language models?**
  - Large language models tend to be effective zero- or few-shot learners with high accuracy
  - These large language models have a number of exciting downstream applications
- **Why has LLM training efficiency become important?**
  - Computation at scale has become more available and datasets have become larger
  - Number of parameters have grown at an exponential rate



Motivation And Background

- What are some challenges of training large language models?
  - Parameters of these models can't fit in the memory of even the largest GPU
  - Large parameter volumes lead to increased compute operations and training times

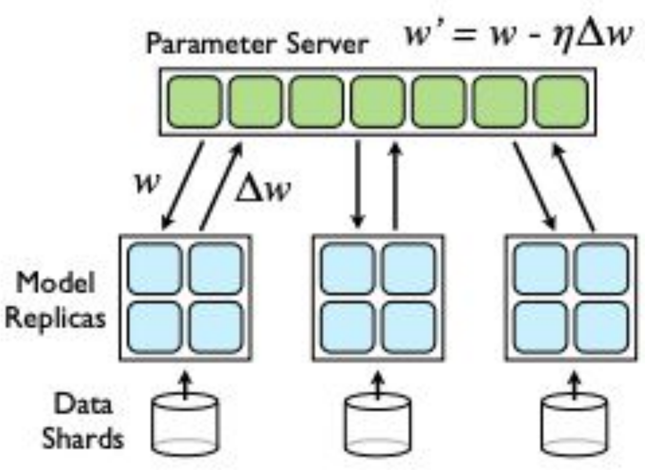
	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	**54x**	**547x**
Memory Footprint	5.12GB	24GB	**275GB**	**2800GB**

From slide "AI Efficiency: Systems and Algorithms Overview & Key Challenges in LLMs Training Systems"

Existing Work

- Data Parallelism

  DP usually has a good scale-out ability, but suffers from two limitations:

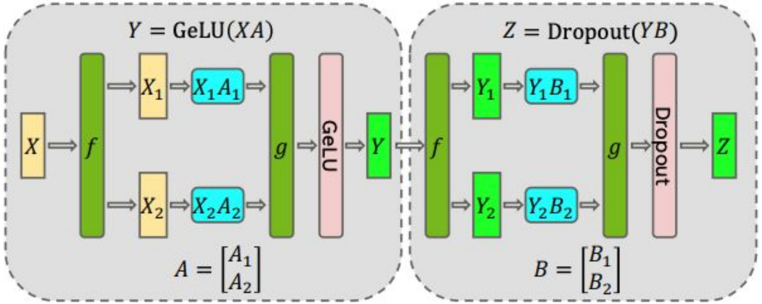


  - For a fixed global batch size, the per-GPU batch size becomes too small beyond a certain point.
  - The maximum number of devices that can be used is determined by the batch size.

Dean, Jeffrey, et al. “Large Scale Distributed Deep Networks.” NIPS. 2012.

Existing Work

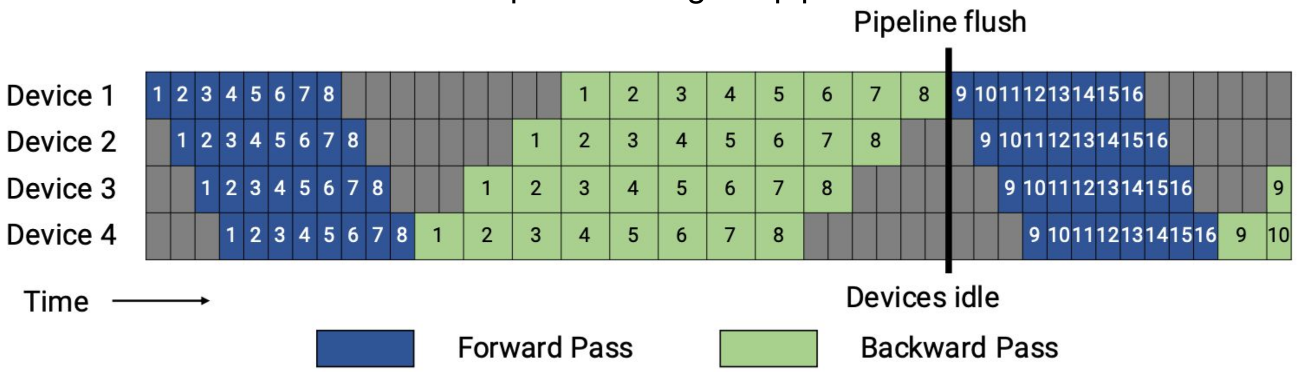
- Tensor Model Parallelism - Megatron-LM
  - Split tensor across GPUs.
  - Inter-GPUs links works well for models inside one server
- Problems when need to split models across multiple servers:
  - The all-reduce communication can't go through NVlinks
  - High model parallelism can create small matrix multiplications, reducing GPU utilization



Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Existing Work

- Pipeline model parallelism
  - Layers of a model are striped over multiple GPUs
  - A batch is divided into microbatches, with pipelined execution across them
  - Layer assignment and scheduling strategy cause performance trade-offs
- Overhead of flushing the pipeline
  - Has strict semantics and requires optimizer step synchronization and pipeline flushing at the end of every batch
  - As much as 50% of time can be spent flushing the pipeline



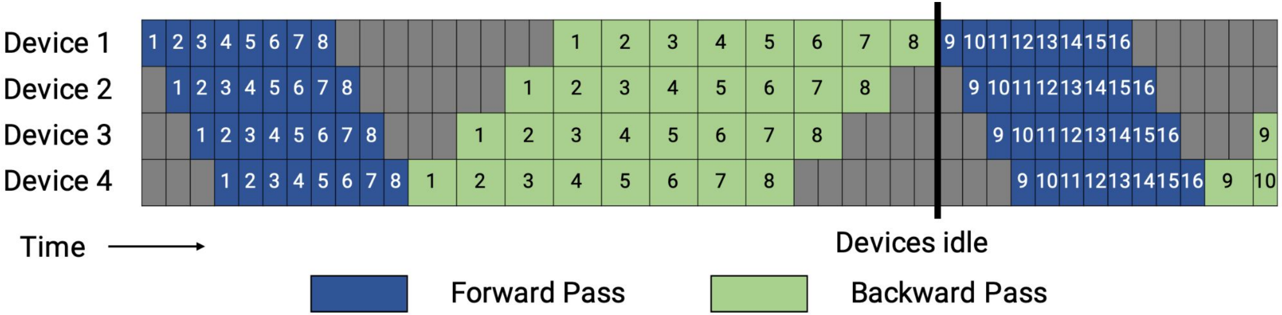
Contributions - Interleaved stage scheduling for PP

Three possible ways of scheduling forward and backward:

Notation:
t_{pb} = Size of pipeline bubble(t_pb)
m = #microbatches(m)
p = Pipeline stages(p)
t_{id} = Ideal time per iteration(t_id)
t_f = time of forward
t_b = time of backward

- Default schedule (GPipe to PipeDream-Flush):

  Gpipe:



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

$$Bubble\ time\ fraction = \frac{t_{pb}}{t_{id}} = \frac{p-1}{m}$$

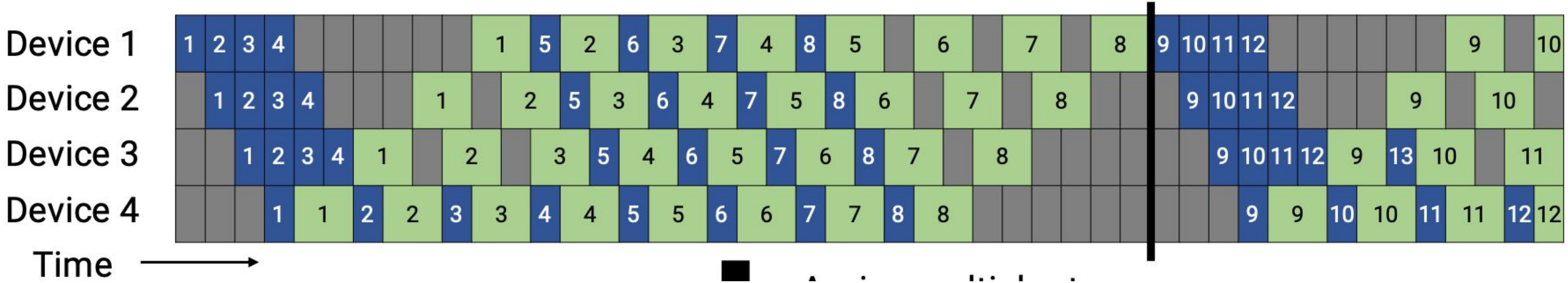
We want m $\gg$ p

However, such a large m has a high memory footprint as it requires stashed intermediate activations

Contributions - Interleaved stage scheduling for PP

- Default schedule (GPipe to PipeDream-Flush):

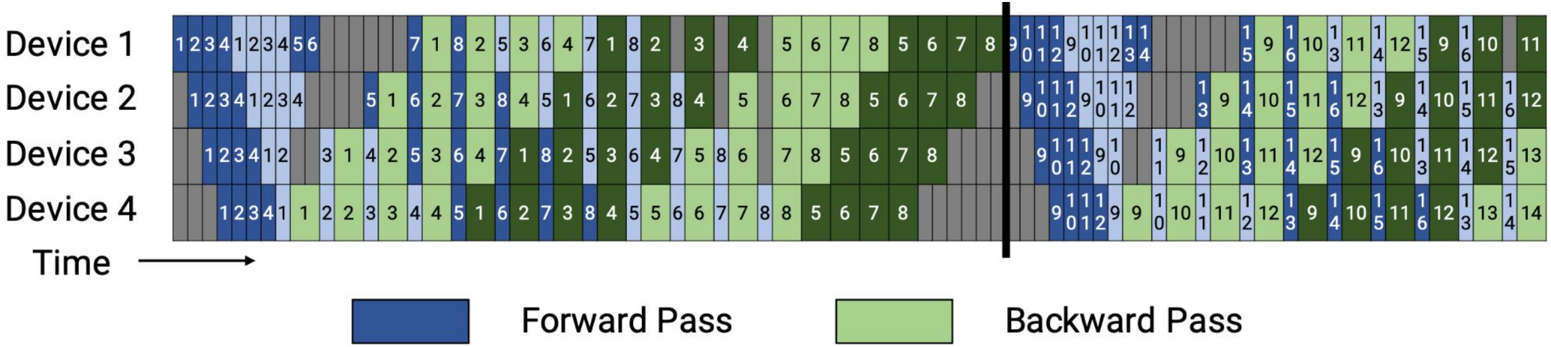
PipeDream-Flush schedule:



  - Limits the number of in-flight microbatches.
  - In steady states, worker will perform one forward pass followed by one backward pass.
  - Only required activations to be stashed for *p* microbatches, compared to m microbatches for GPipe
  - We can have larger m, and will be more memory efficient.

Contributions - Interleaved stage scheduling for PP

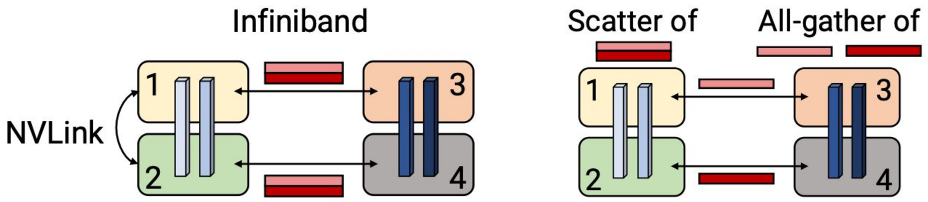
- Schedule with Interleaved Stages – attempting to reduce the bubble size
  - Each device can perform computation for multiple subsets of layers(model chunk)
    - i.e. device 1 had layers 1 − 4, device 2 had layers 5 − 8, and so on at first. After model chunk, device 1 has layers 1, 2, 9, 10; device 2 has layers 3, 4, 11, 12; and so on.
  - Extend the 1F1B schedule.
  - If each device has v stages (or model chunks)
  - pipeline bubble time thus reduces to $t_{pb} = \frac{(p-1)\cdot(t_f+t_b)}{v}$ and $BFT = \frac{(p-1)}{m}\cdot\frac{1}{v}$

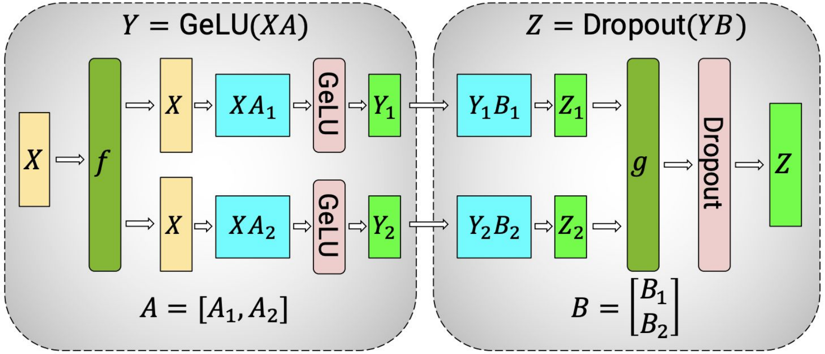


Dark colors show the first chunk and light colors show the second chunk. The size of the pipeline bubble is smaller (the pipeline flush happens sooner in the interleaved timeline).

Scatter/gather communication optimization

- Scatter/gather optimization as an extension to the Megatron-LM
  - This reduced pipeline bubble size does not come for free
  - The output of each transformer layer is replicated (after g in MLP block)
  - They are sending and receiving the exact same set of tensors
  - Split the sending message to equal size of chunk and perform an all-gather on receivers





(a) MLP.

Performance Analysis of Combined Parallelism

- Tensor and Pipeline Model Parallelism
  - t ⇧, pipeline bubble ⇩

$$\frac{p-1}{m} = \frac{n/t-1}{m}$$

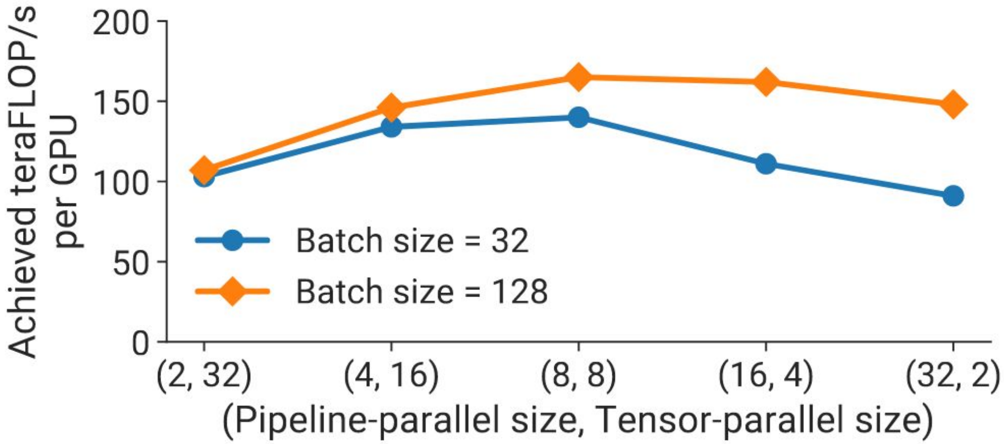
- (p, t, d): Parallelization dimensions, where p is the pipeline-model-parallel size, t is the tensor-model-parallel size, and d is the data-parallel size.
- n: Number of GPUs, satisfying $p \cdot t \cdot d = n$.
- B: Global batch size.
- b: Microbatch size.
- $m = \frac{B}{b \cdot d}$: Number of microbatches per pipeline.

- Communication overhead
  - All-reduce communication for tensor model parallelism is expensive!
  - Especially when cross servers

Takeaway #1: Use tensor model parallelism within a server and pipeline model parallelism to scale to multiple servers.

Evaluation - TP vs. PP

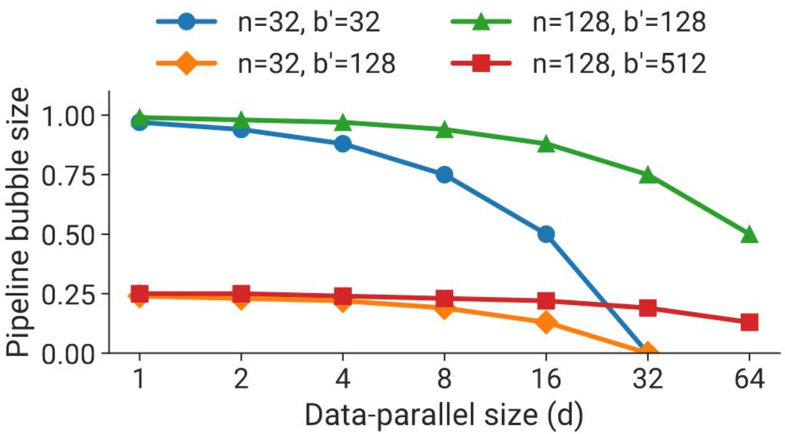
- Tensor versus Pipeline Parallelism
  - 161-billion param. GPT
  - Peak performance achieved when t = p = 8
  - Need a conjunction of both types of model parallelisms



Performance Analysis of Combined Parallelism

- Data versus Pipeline Parallelism

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

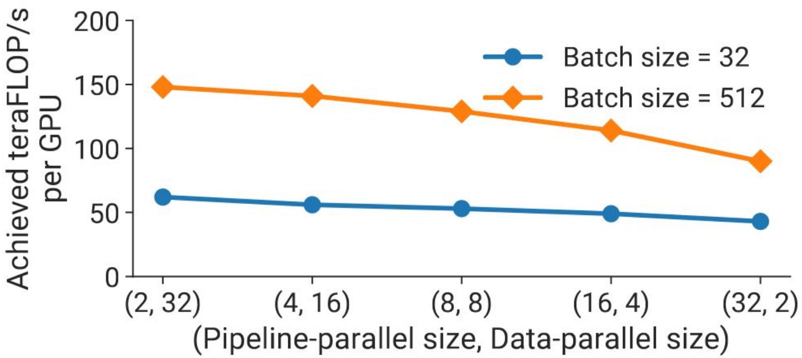


- Data versus Tensor Parallelism
  - DP is less communication heavy than TP
    - All-reduce once per batch vs. All-reduce once per microbatch
  - Tensor parallelism can lead to hardware underutilization

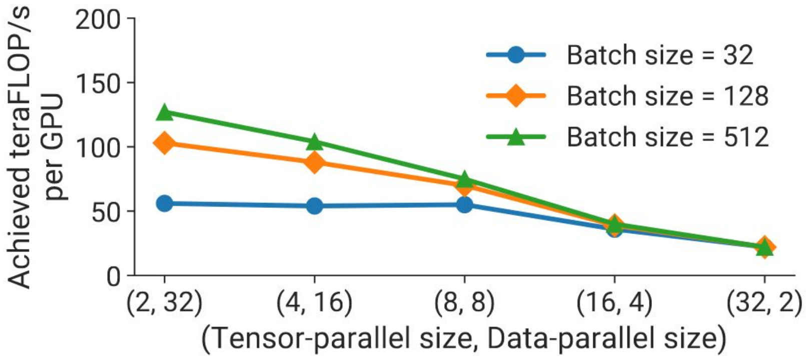
Takeaway #2: Decide tensor-parallel size and pipeline-parallel size based on the GPU memory size; data parallelism can be used to scale to more GPUs.

Evaluation - DP vs. Model Parallelism

- Pipeline-parallelism vs. Data-parallelism
  - 5.9-billion param. GPT
  - Throughput decreases as pipeline-parallel size increases



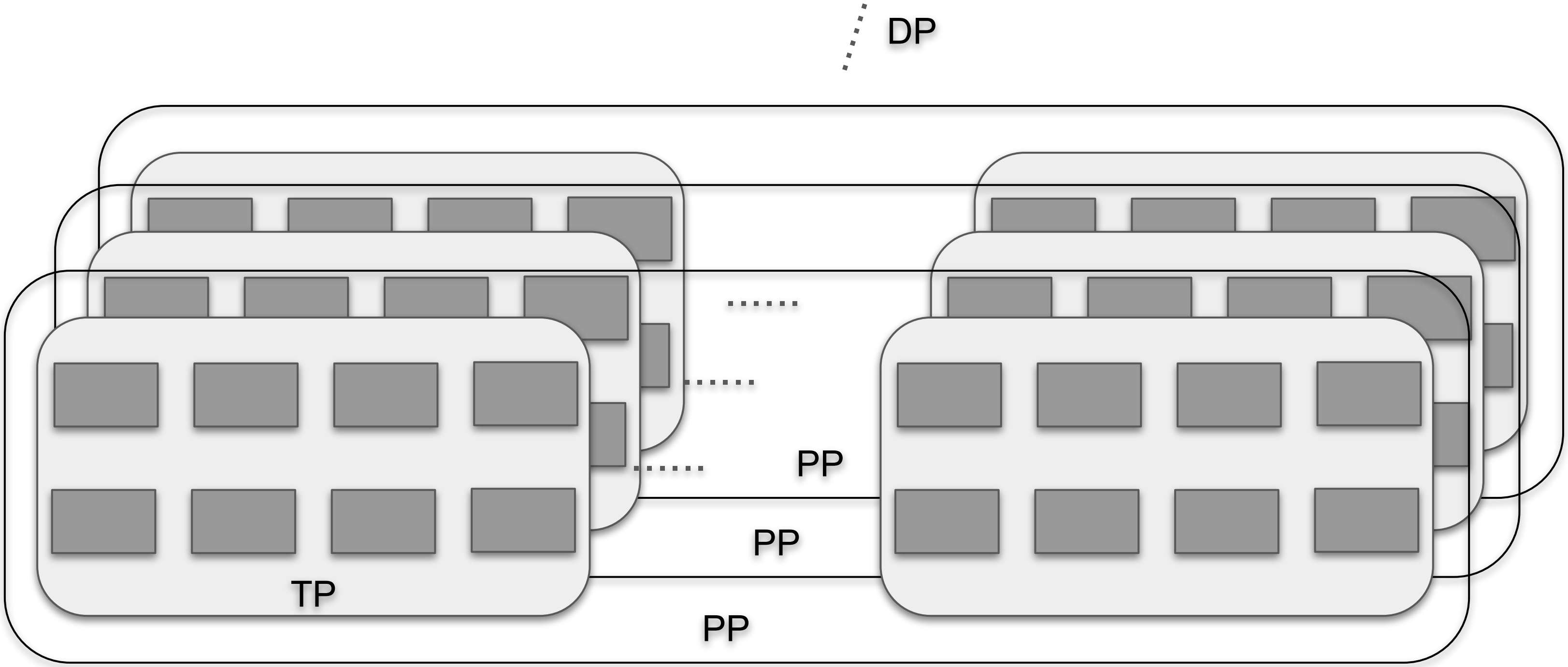
- Tensor-parallelism vs. Data-parallelism
  - 5.9-billion param. GPT
  - Throughput decreases as tensor-parallel size increases



Limitations of data-parallelism:

1. Memory capacity
2. Scaling limitation proportional to the batch size

3D Parallelism



Evaluation setup

- Megatron-LM extension
- Selene supercomputer
    - Each node has 8 NVIDIA 80-GB A100 GPUs
    - Inter-GPU: NVLink and NVSwitch
    - Inter-node: eight NVIDIA Mellanox 200Gbps HDR Infiniband HCAs
- Model: GPT

Evaluation - End-to-end Performance

- Superlinear scaling of throughput
  - Per-GPU utilization improves as the model get larger
  - Communication overhead is not significant

Number of parameters (billion)	Attention heads	Hidden size	Number of layers	Tensor model-parallel size	Pipeline model-parallel size	Number of GPUs	Batch size	Achieved teraFlOP/s per GPU	Percentage of theoretical peak FLOP/s	Achieved aggregate petaFLOP/s
1.7	24	2304	24	1	1	32	512	137	44%	4.4
3.6	32	3072	30	2	1	64	512	138	44%	8.8
7.5	32	4096	36	4	1	128	512	142	46%	18.2
18.4	48	6144	40	8	1	256	1024	135	43%	34.6
39.1	64	8192	48	8	2	512	1536	138	44%	70.8
76.1	80	10240	60	8	4	1024	1792	140	45%	143.8
145.6	96	12288	80	8	8	1536	2304	148	47%	227.1
310.1	128	16384	96	8	16	1920	2160	155	50%	297.4
529.6	128	20480	105	8	35	2520	2520	163	52%	410.2
1008.0	160	25600	128	8	64	3072	3072	163	52%	502.0

Evaluation - End-to-end Performance

- Estimated Training Time
  - T: number of tokens
  - P: number of parameters
  - n: number of GPUs
  - X: throughput
  - E.g. GPT3

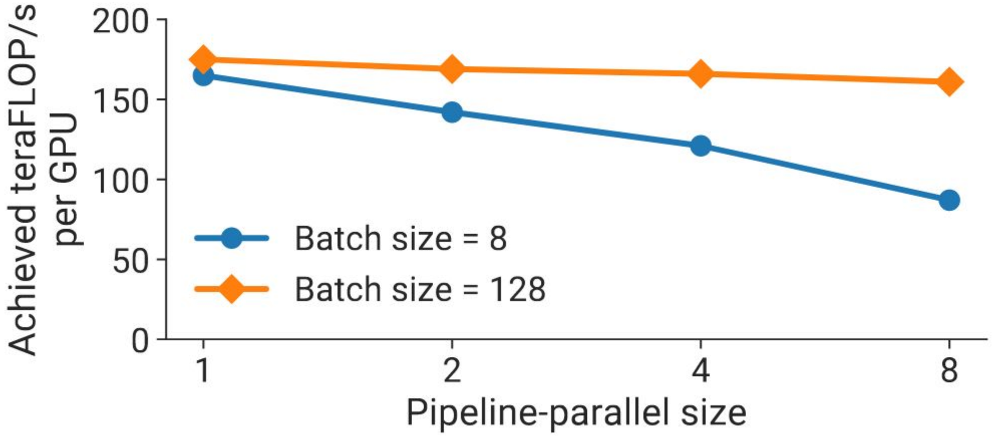
$$\text{End-to-end training time} \approx \frac{8TP}{nX}$$

T (billion)	P (billion)	n	X (teraFLOPs/s per GPU)	#Days
300	175	1024	140	34
1000	450	3072	163	84

288 years with a single V100 NVIDIA GPU

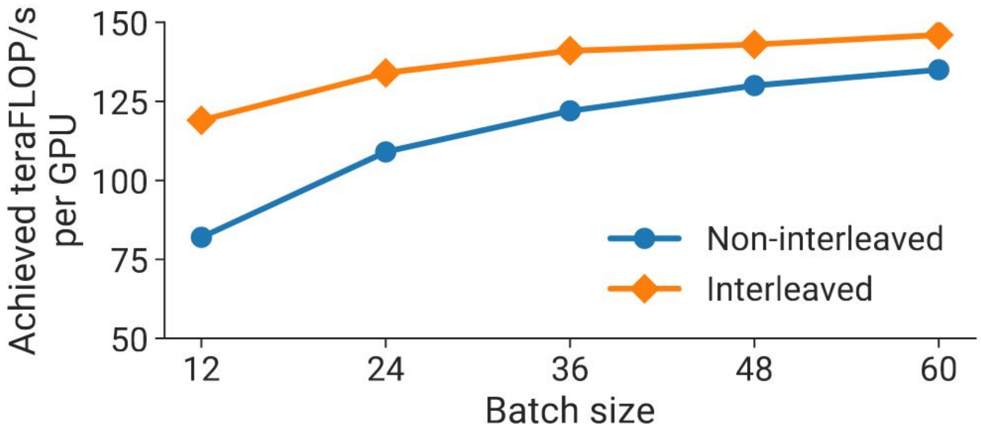
Evaluation - Pipeline Parallelism

- Weak Scaling - increase the #layers while increasing PP size
- Higher batch size scales better (p-1)/m



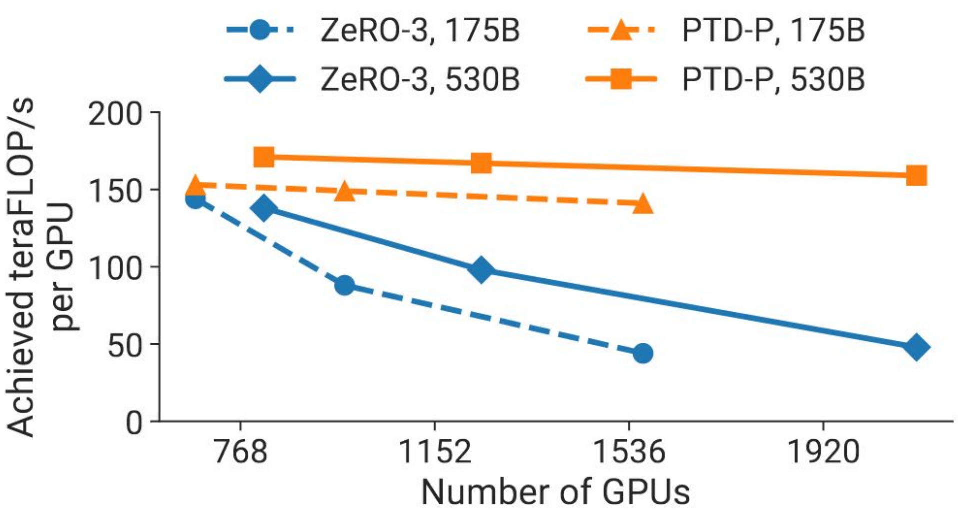
Evaluation - Pipeline Parallelism

- Interleaved schedule with scatter/gather optimization has higher throughput
  - The gap closes as the batch size increases
    - Bubble size decreases when batch size increases (i.e., more micro-batches)
    - Interleaved schedule features more communication cost per sample



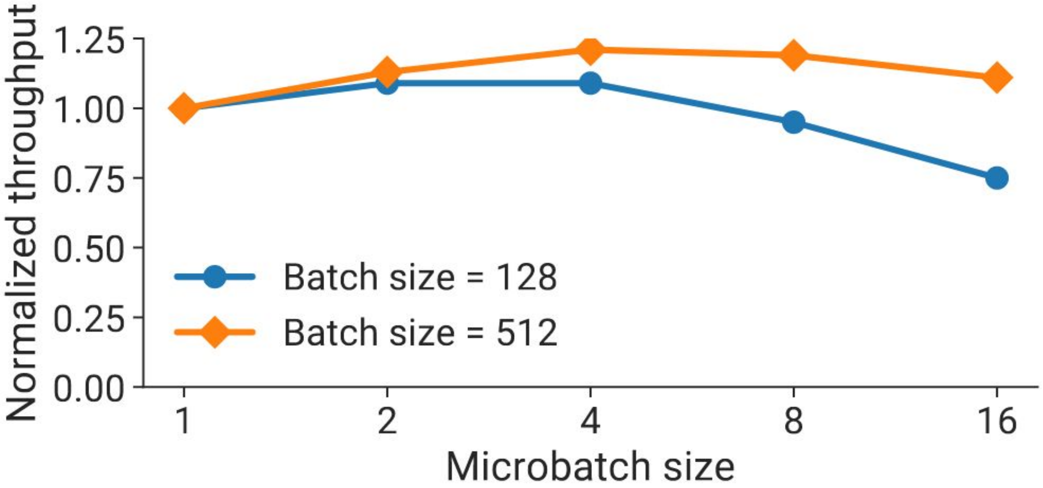
Evaluation - Comparison with ZeRO-3

- ZeRO-3: No model parallelism in use
- PTD-P scales more gracefully as the #GPUs increases
  - Less cross-node communication



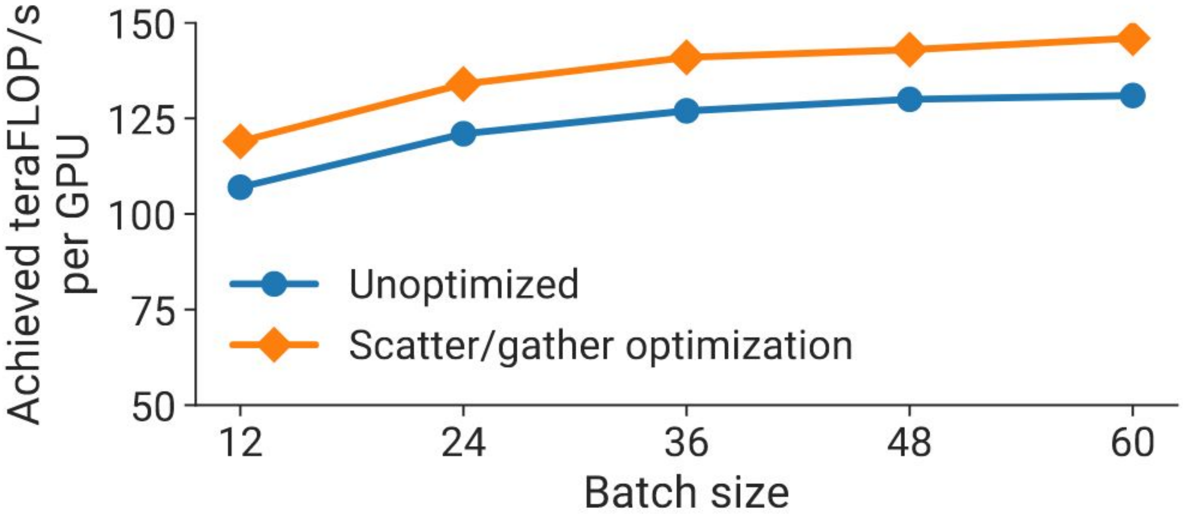
Selection of Microbatch size

- Optimal microbatch size is **model dependent**
  - Arithmetic intensity
  - Pipeline bubble size



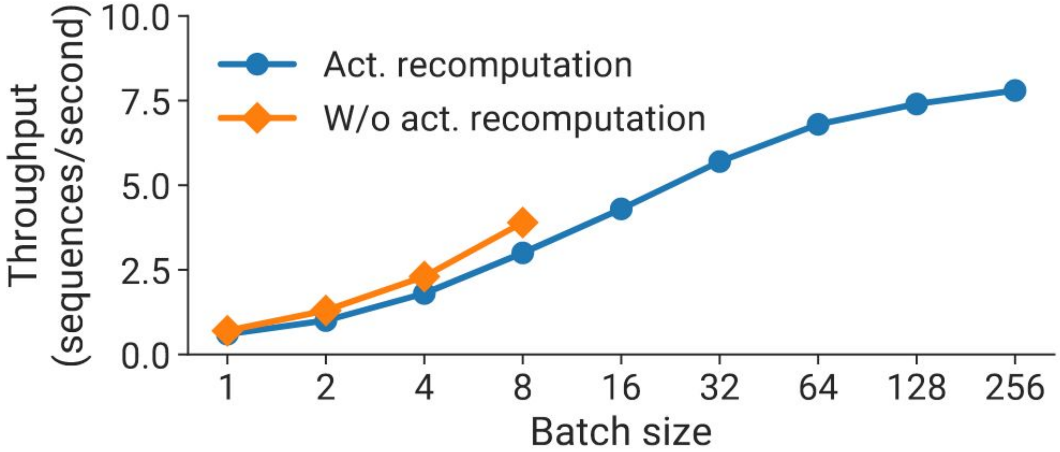
Evaluation - Scatter-gather optimization

- GPT model with 175 billion parameters using 96 A100 GPUs
- Up to 11% in throughput
  - Large batch size with interleaved schedules
  - Reduce cross-node communication cost



Activation Recomputation

- How many activation checkpoints should be used?
- $c \cdot A^{\text{input}} + l/c \cdot A^{\text{intermediate}}$ -> $c = \sqrt{l \cdot A^{\text{intermediate}}/A^{\text{input}}}$
- In general, checkpoint every 1 or 2 layers is optimal
- Evaluated on a GPT model with 145 billion parameters on 128 A100 GPUs, $(t, p) = (8, 16)$



Related Work

- Parallelism for large model training
  - Variations of pipeline model parallelism
    - Token level
    - Relaxed semantics
    - Asynchronous model updates
  - Combined data and model parallelism
    - **DeepSpeed**
- Shared Data Parallelism
- Automatic Partitioning
- HPC for training

Strengths and Weaknesses

+ 3D parallelism is effective at scaling large models to multiple servers
+ Provides a comprehensive reasoning framework for parameter selection in 3D parallelism, considering not only p, t, d, and also microbatch size and activation recomputation
- No enough information on the programming interface to the extension
  - How much code refactoring is needed?
  - Who is responsible for the refactoring?

Backup slides

Existing Work

- What are some existing techniques and their limitations?
  - Data Parallelism
  - Tensor Parallelism
    - Megatron-LM
  - Pipeline Parallelism
    - GPipe
    - PipeDream-Flush

Contributions

- Tow techniques
  - Interleaved stage scheduling for pipeline parallelism
  - Scatter-gather communication for tensor parallelism
- Performance modeling of combined pipeline, tensor, and data parallelism
- Implemented Megatron-LM extension

Move the end-to-end evaluation and pipeline parallelism evaluation up

Performance modeling of combined pipeline, tensor, and data parallelism

- Tensor and Pipeline Model Parallelism
  - The pipeline bubble size in terms of t is: $\frac{p-1}{m} = \frac{n/t-1}{m}$
  - As *t* increases, the pipeline bubble thus decreases
  - Pending

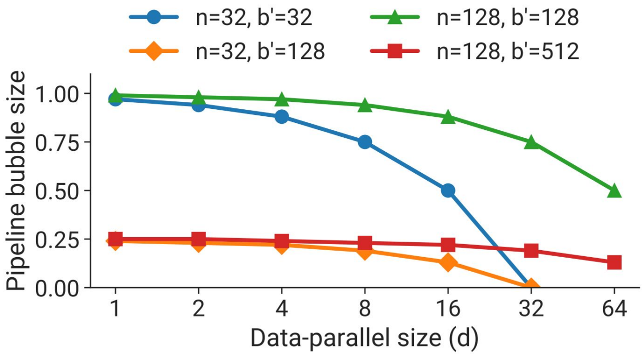
- (p, t, d): Parallelization dimensions, where p is the pipeline-model-parallel size, t is the tensor-model-parallel size, and d is the data-parallel size.
- n: Number of GPUs, satisfying $p \cdot t \cdot d = n$.
- B: Global batch size.
- b: Microbatch size.
- $m = \frac{B}{b \cdot d}$: Number of microbatches per pipeline.

Performance modeling of combined pipeline, tensor, and data parallelism

- Data Parallelism and Pipeline Model Parallelism
  - Let t = 1 (tensor-model-parallel size)

    Let $m = \dfrac{B}{(d \cdot b)} = \dfrac{b'}{d}$ and $b' := \dfrac{B}{b}$

    Then the pipeline bubble size $\dfrac{p-1}{m} = \dfrac{n/d-1}{b'/d} = \dfrac{n-d}{b'}$.
  - As d becomes larger, $n - d$ becomes smaller, and thus the pipeline bubble becomes smaller



Evaluation

- Hardware
  - Selene Supercomputer (Todo: draw a tree to show the topology)
- Model: GPT

Computation Optimizations

- Change the data layout
- Fused kernels for a sequence of element-wise operations
- Two custom kernels to enable the fusion of scale, mask, and softmax