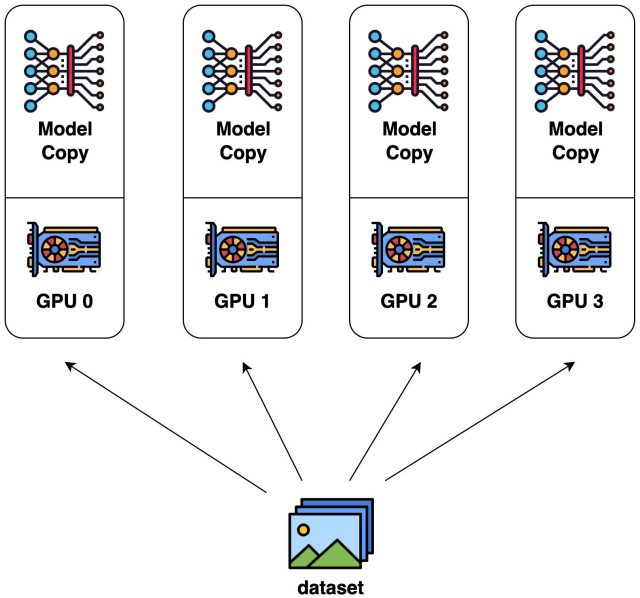
# Pathways: Asynchronous Distributed Dataflow for ML

Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Dan Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, Brennan Saeta, Parker Schuh, Ryan Sepassi, Laurent El Shafey, Chandramohan A. Thekkath, Yonghui Wu

MLSys Outstanding Paper Award

# Computation Models in Large-scale ML

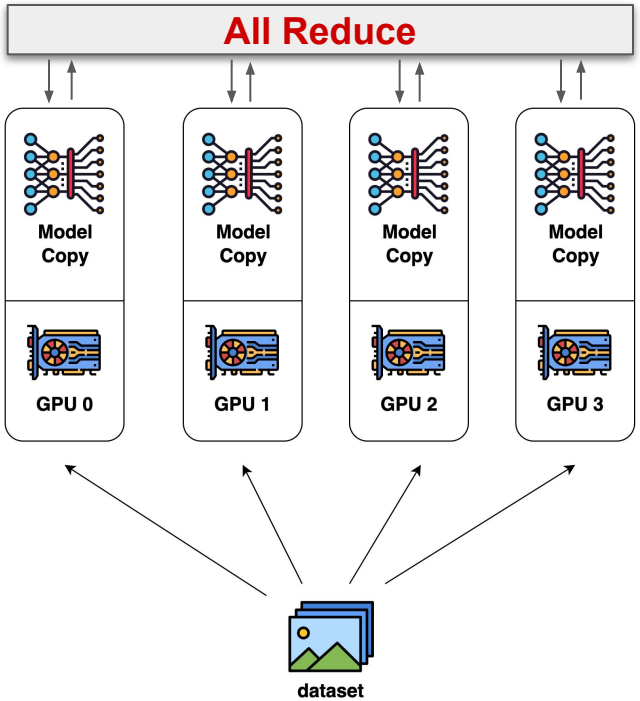
- Single Program Multiple Data (SPMD) – 1994, MPI




- The same program is run on every accelerator.

# Computation Models in Large-scale ML

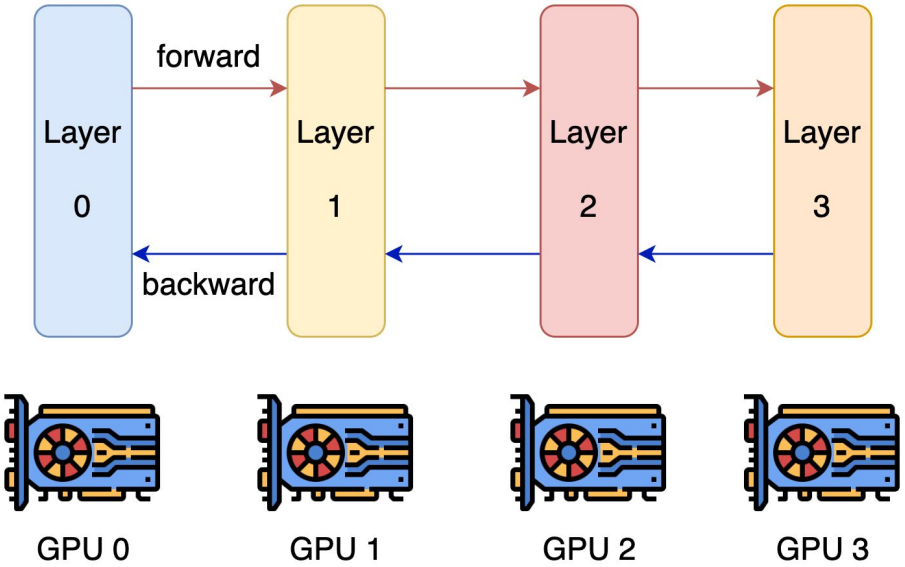
- Single Program Multiple Data (SPMD) – 1994, MPI




- The same program is run on every accelerator.
- Communication is ‘rare’ and done with standard collectives (e.g. **All Reduce**)

# Computation Models in Large-scale ML

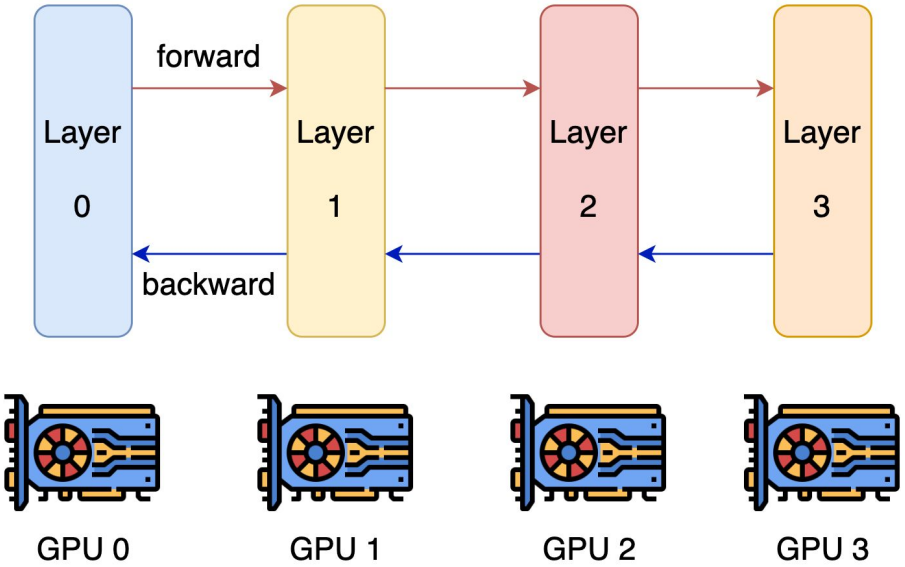
- Multi Program Multiple Data (MPMD)




- Accelerators run heterogeneous computation
- Communication is involved in the computation process

# Computation Models in Large-scale ML

- Multi Program Multiple Data (MPMD)




- Accelerators run heterogeneous computation
- Communication is involved in the computation process

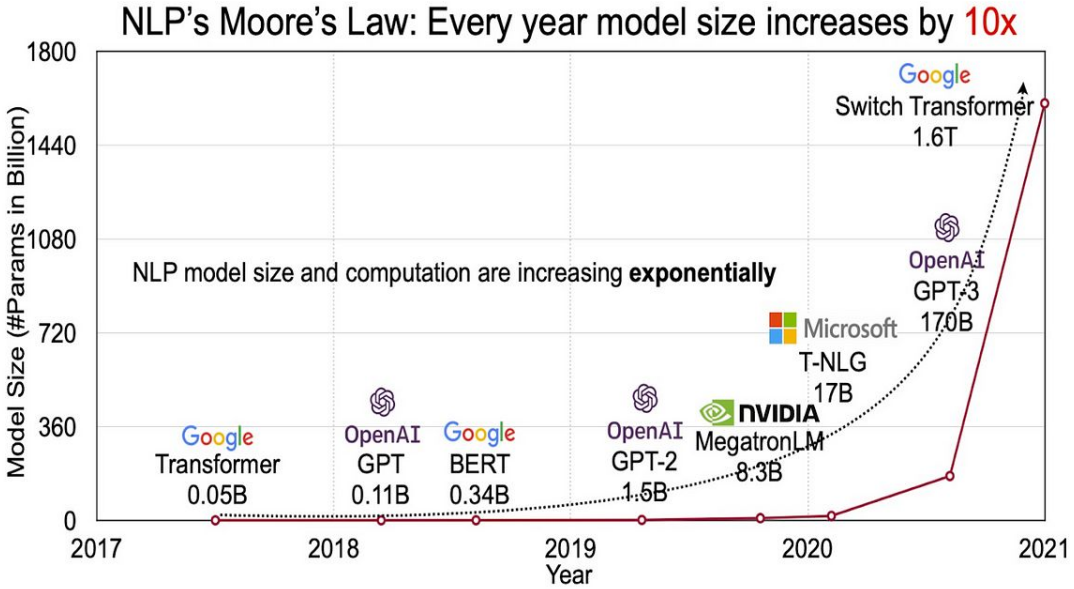
Examples:

- Mixture of Experts (MoE)
- Pipeline Parallelism
- Network Architecture Search (NAS)
- Computational Sparisity

…

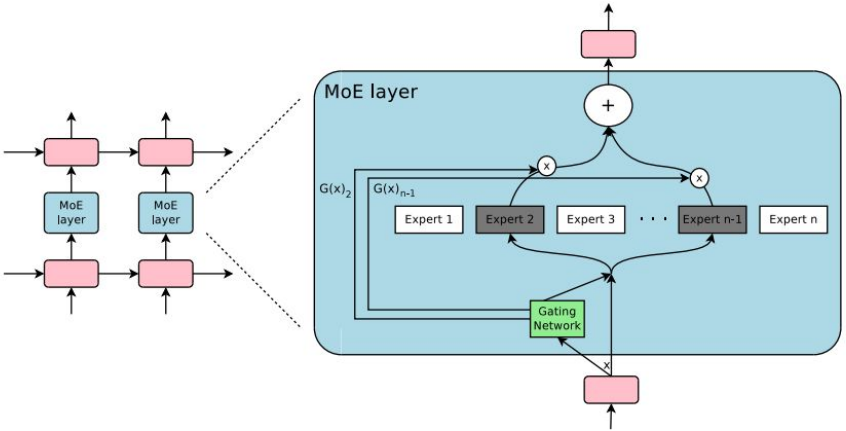
# We need heterogeneous computation (MPMD)

- Modern deep neural networks are orders of magnitude larger than the capacity of accelerator (HBM) memory

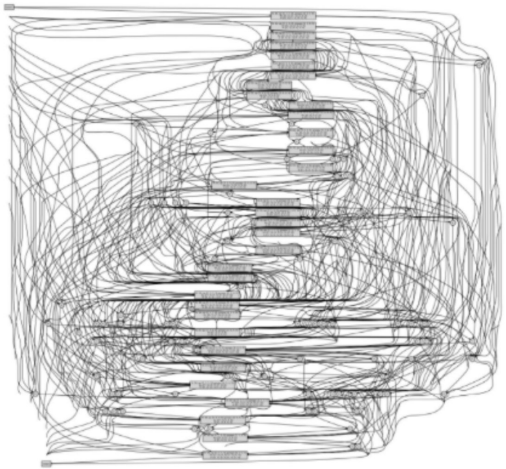


# We need heterogeneous computation (MPMD)

- Modern deep neural networks are orders of magnitude larger than the capacity of accelerator (HBM) memory
- The increasing popularity of heterogeneous models/computations




Mixture of Experts (MoE)

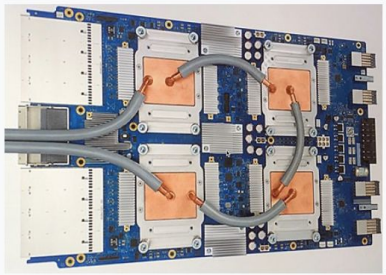


Network Architecture Search (NAS)

# We need heterogeneous computation (MPMD)

- The increasing scale of model sizes
- The increasing popularity of heterogeneous models/computations
- The increasing existence of heterogeneous hardware

**Tensor Processing Unit**



Tensor Processing Unit 3.0

| | |
|---|---|
| **Designer** | Google |
| **Introduced** | May 2016 |
| **Type** | Neural network<br>Machine learning |

# Existing Work: Multi-Controller v.s. Single-Controller

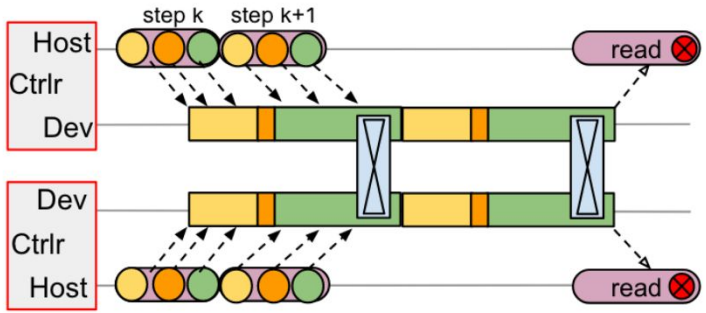
Concept – **Controller**:

- A central **coordinating** component responsible for
  - distribution of computation (scheduling)
  - execution of computation
  - resource allocation
  - optimizations
  - …
- Goal: To ensure complex computations are efficiently processed.

# Example: Multi-Controller

Multi-Controller Architecture (e.g. PyTorch, JAX):

- Controller is every accelerator itself (by replicating the same code to every accelerator)




(a) JAX/PyTorch SPMD

Pros

- Fast Workload Dispatch (PCI-E communication)

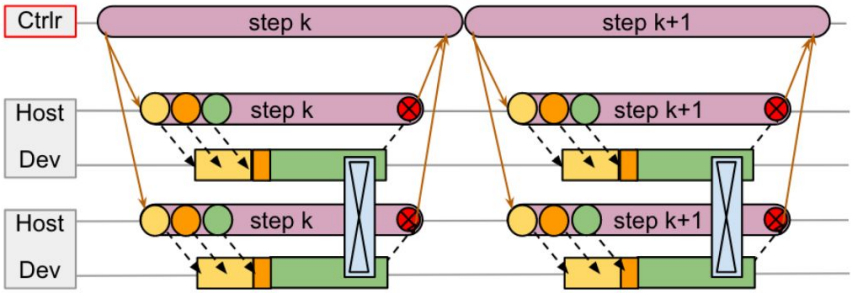
Cons:

- Needs non-trivial custom coordinators when executing MPMD

# Example: Single-Controller

Single-Controller Architecture (Tensorflow V1)

- One central controller that coordinates all worker nodes




(b) TF1 SPMD

Pros:

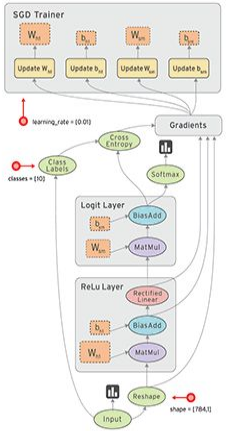
- Generality (not limited to SPMD)
- Global Optimizations

Cons:

- **Slow**: Single controller becomes the **performance bottleneck** (communication & graph complexity)

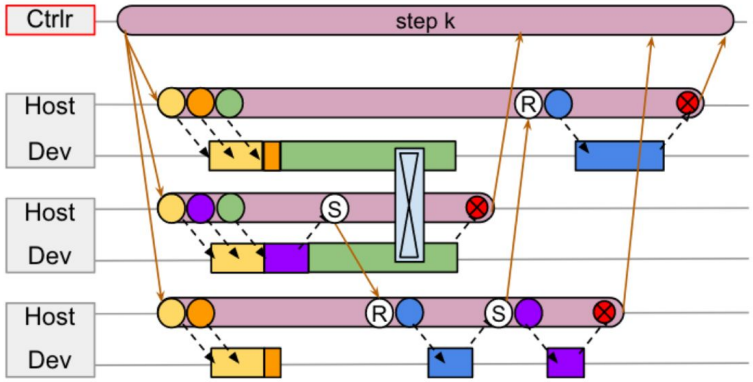
# Example: Single-Controller

User's Input: **Computation Graphs**




Single-Controller Architecture (Tensorflow V1)

- One central controller that coordinates all worker nodes




(c) TF1 non-SPMD

Pros:

- Generality (not limited to SPMD)
- Global Optimizations

Cons:

- **Slow**: Single controller becomes the **performance bottleneck** (communication & graph complexity)

# Existing Work: Multi-Controller v.s. Single-Controller

We want to express MPMD,

but Tensorflow is too slow, and Multi-Controller solutions are too limited…

What about building upon Multi-Controller solutions (e.g. JAX, PyTorch)?

→ **Megatron-LM**, DeepSpeed

# Existing Work: Multi-Controller v.s. Single-Controller

We want to express MPMD,

but Tensorflow is too slow, and Multi-Controller solutions are too limited…

What about building upon Multi-Controller solutions (e.g. JAX, PyTorch)?

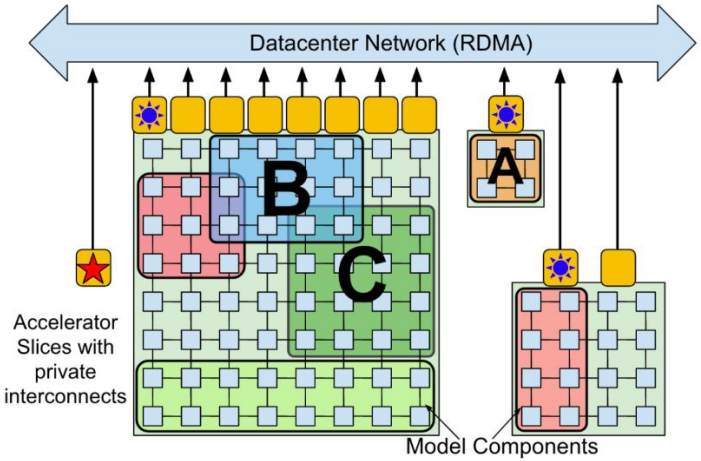
Google: No! We want to build a really general (single-controller) yet fast solution.
→ **Pathways**

# Google’s Data Center

**Islands**: a cluster (~1000) of all-to-all connected TPU devices, with ~100 hosts

**ICI**: high-speed interconnect between accelerators (e.g.NVLink)

**DCN**: slow connection between islands (e.g. Infiniband)

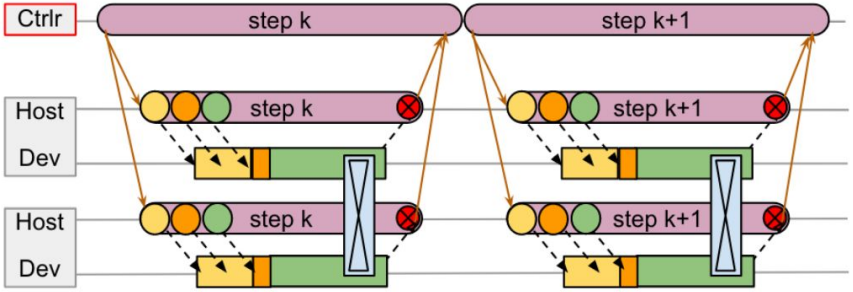


# Google’s Data Center

**Islands**: a cluster (~1000) of all-to-all connected TPU devices, with ~100 hosts

**ICI**: high-speed interconnect between accelerators (e.g. NCCL)

**DCN**: slow connection between islands (e.g. Infiniband)




(b) TF1 SPMD

TF1 is slow due to frequent communication over DCN

# Google’s Desire

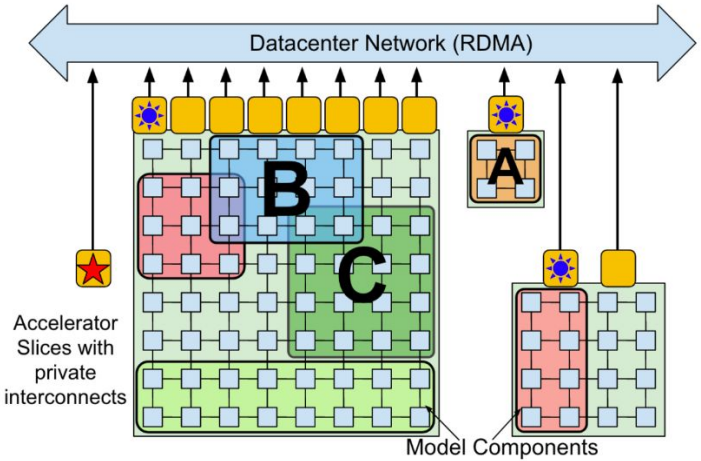
MPMD is the future.

**Generality** from single-controller with **comparable performance** to multi-controller**.**

# Resource Virtualization

“Virtual Slice”: A set of virtual devices with specified **topology requirements**

Physical devices can be added/removed dynamically to/from the slice.



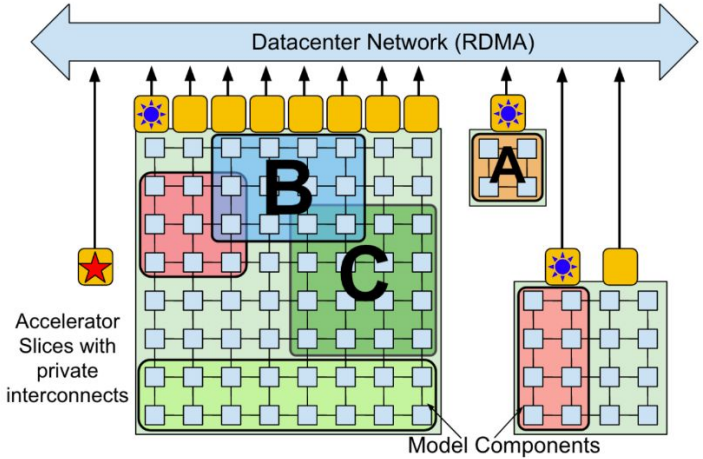
# Resource Virtualization

“Virtual Slice”: A set of virtual devices with specified **topology requirements**

Physical devices can be added/removed dynamically to/ the slice.

Benefit:

1. Transparent task suspend/resume, and migration
2. Higher device utilization by
   a. Enable multi-task resource sharing
   b. Decouples user program from resource management
   c. Alleviate user’s responsibility to monitor program failures and keep resource usage high

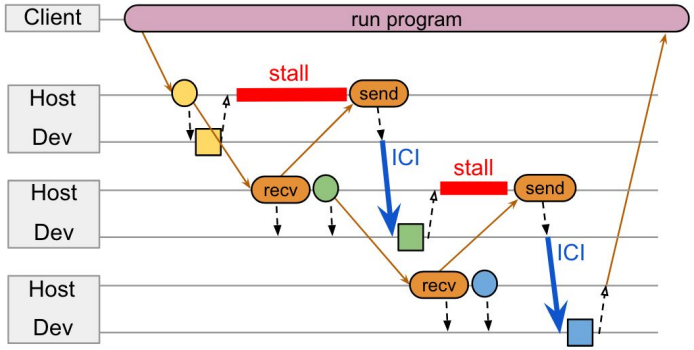


# Cross-host Coordination (based on PLAQUE)

Support efficient data transfer between hosts:

Requirements:

1. Scalability & Latency
2. Efficient Sparse Communication → Key to fast data-dependent control flow

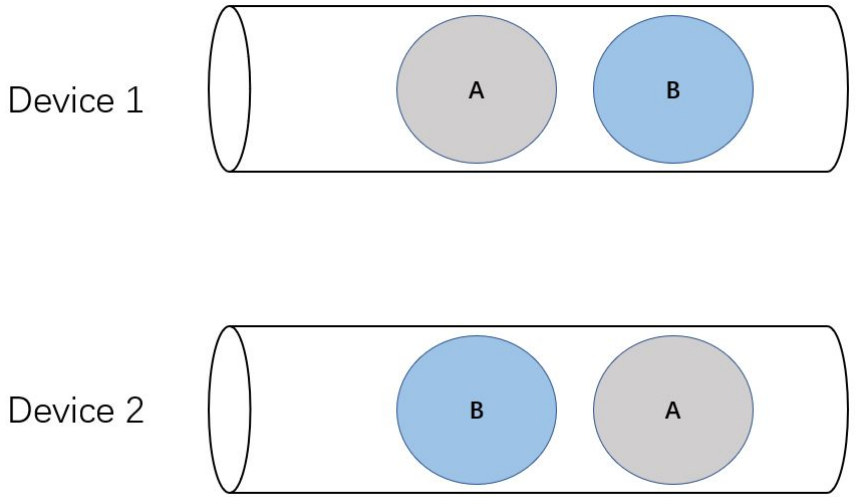



(a) Sequential dispatch

# Gang Scheduling

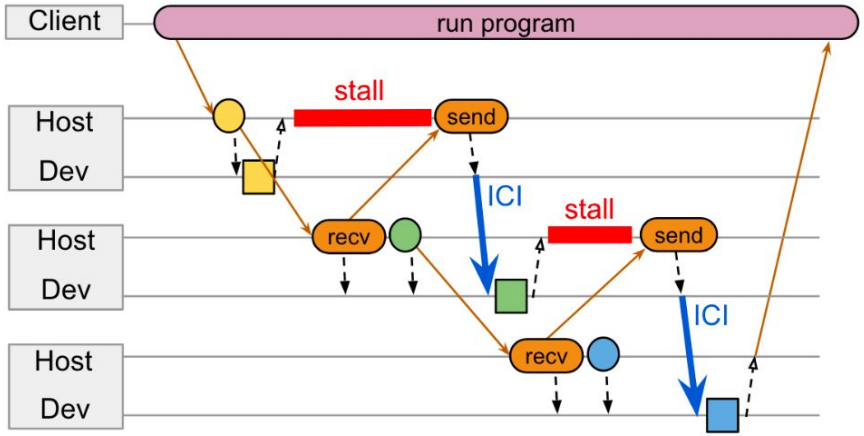
Schedule related tasks together

1. Deadlock Avoidance (TPU computation are not preemptible)
2. Higher Throughput (context switch is costly on accelerator)

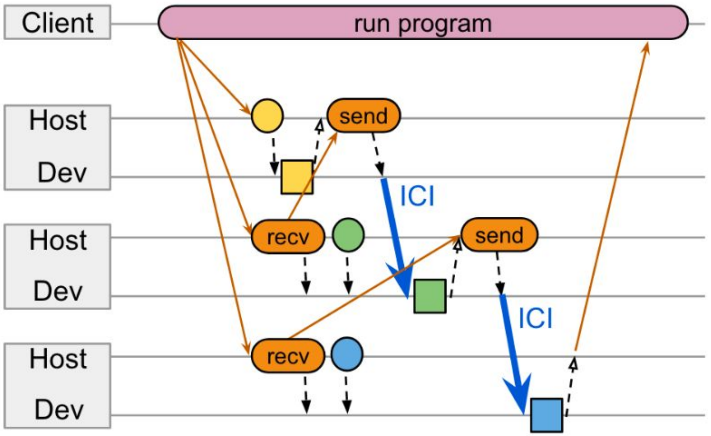



A and B are independent jobs, and not enforcing execution order might lead to deadlocks.

# Parallel Asynchronous Dispatch



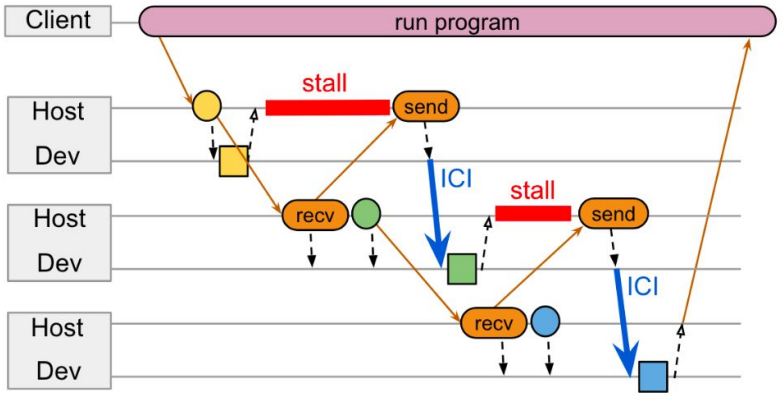

(a) Sequential dispatch



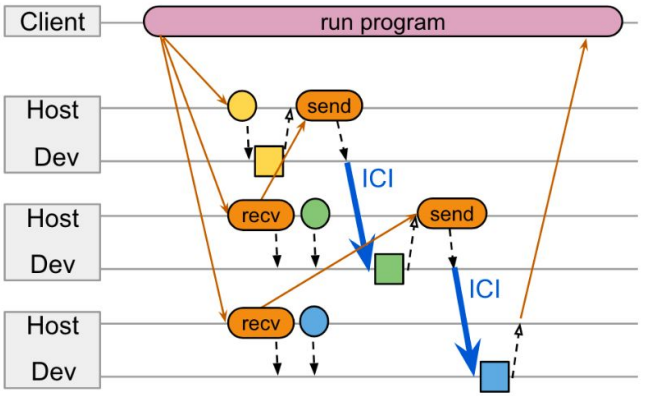

(b) Parallel dispatch

Pre-allocate resources for computations with known resource requirements (i.e. no data-dependent control flow)

# Parallel Asynchronous Dispatch




(a) Sequential dispatch




(b) Parallel dispatch

Pre-allocate resources for computations with known resource requirements (i.e. no data-dependent control flow)

TPUs have only limited support for dynamic shapes, making them a good fit for this technique.

# Evaluation of Pathways

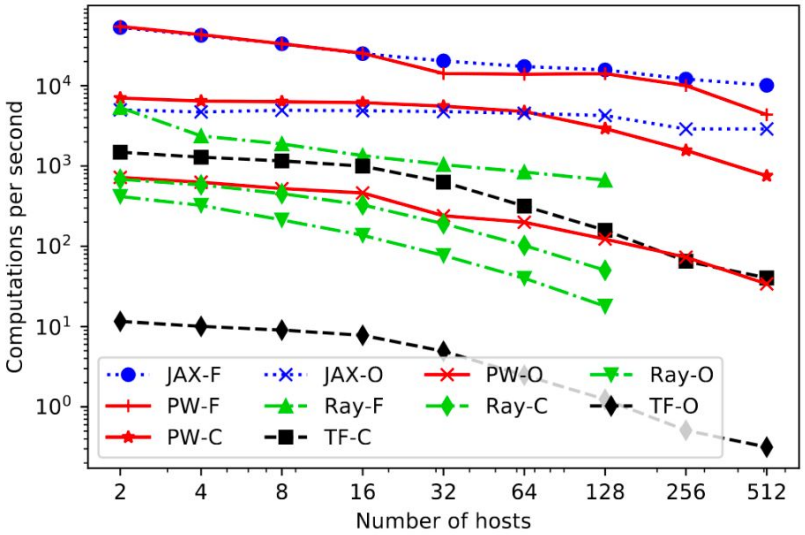
Micro Benchmarks

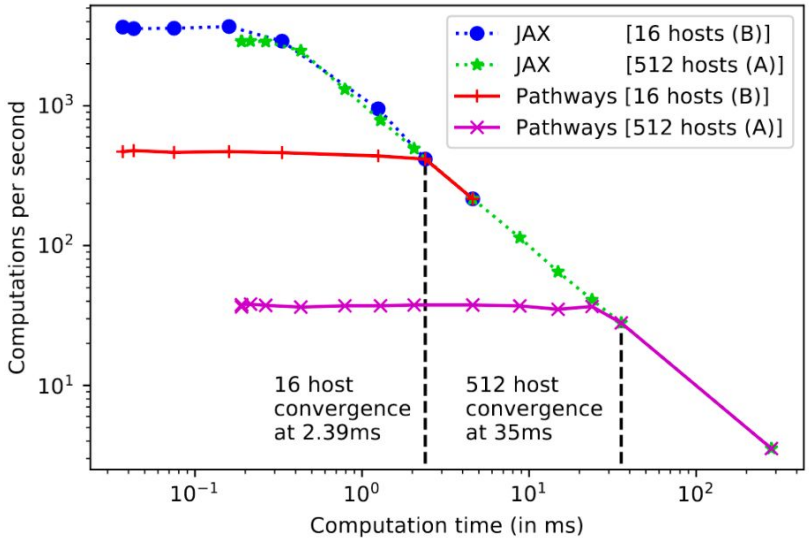
1. Single-controller dispatch overheads
2. Multi-tenancy

End-to-End Evaluation

3. Large-scale model training performance

# Single-Controller Dispatch Overheads




Little dispatch overhead when computation is not trivial

# Large-scale model training performance

*Table 1.* Training throughput (tokens/s) of Text-to-text Transformer model configurations from (Raffel et al., 2019) on JAX multi-controller and PATHWAYS.

| Model | Params | TPU cores | JAX | PATHWAYS |
|---|---|---|---|---|
| T5-Base | 270M | 32 | 618k | 618k |
| T5-Large | 770M | 32 | 90.4k | 90.4k |
| T5-3B | 3B | 512 | 282.8k | 282.8k |
| T5-11B | 11B | 512 | 84.8k | 84.8k |

# Large-scale model training performance

*Table 2.* Training throughput (tokens/s) of 3B Transformer language model, using SPMD or multiple pipeline stages, with $C$ TPU cores in PATHWAYS. For pipeline-parallel models, there are $S$ stages and each batch is split into $M$ $\mu$-batches.

| Model configuration | TPU cores | PATHWAYS |
|---|---|---|
| Model-parallel (SPMD) | 128 | 125.7k |
| Pipelining, S=4, M=16 | 128 | 133.7k |
| Pipelining, S=8, M=32 | 128 | 132.7k |
| Pipelining, S=16, M=64 | 128 | 131.4k |
| Pipelining, S=16, M=64 | 512 | 507.8k |

# Summary

- Pathways combines single-controller architecture's generality with multi-controller efficiency in ML.
- Comparable SPMD performance at large-scale operations.
- Parallel dispatch, gang scheduling, and coordination for enhanced efficiency and latency hiding.
- Centralized resource management supports multi-tenant sharing and virtualization, tailored for ML tasks.

# Limitations

- Primarily designed for TPUs, design choices might not hold for GPU
  - e.g. Gang Scheduling, Parallel Asynchronous Dispatch
- Ambiguity in distinct contributions, leveraging many existing systems & techniques.
- Incomplete evaluation
  - lacking end-to-end comparisons with frameworks like Ray.
  - performance benefit might lie in implementation (e.g. on-device object store) instead of design
- Performance benefit might not show for small-scaled workloads
- …


# 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

# Megatron-LM: Context

Can we use a single GPU for training an LLM?

- No — GPU memory too small!
  - Impossible to fit large models on even a A100 (80GB device memory)
- No — it would take too long!
  - Training GPT-3, with 175 billion parameters, takes 288 years with a single V100 NVIDIA GPU

Solution: 1D (Data) Parallelism

# 1D parallelism

Can we use **k** GPUs, and give each GPU data batches of **batch_size / k?**

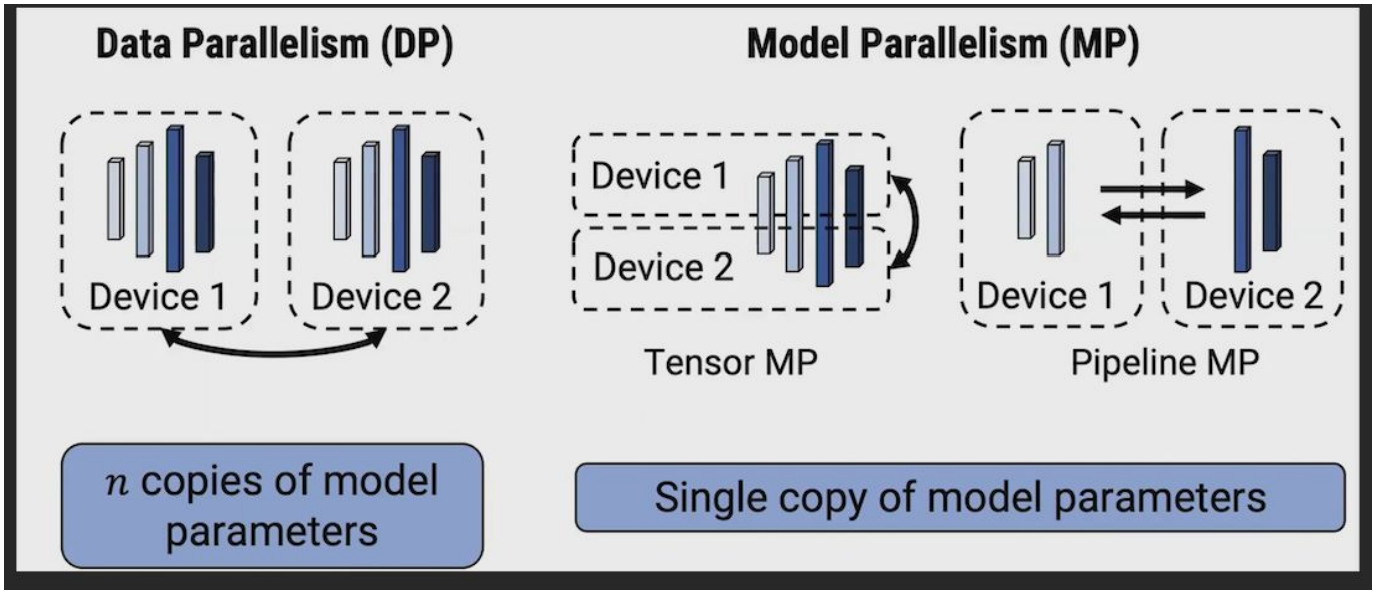
- Data-parallel scale out usually works well..
- For large **k**, per-GPU batch becomes too small
  - Reduced GPU utilization
  - Increased communication cost
- Can have at most **batch_size** GPUs
  - Batchsize == 1024 → at most 1024 data parallel instances
- **Too limited!**

Better Solution: 3D Parallelism

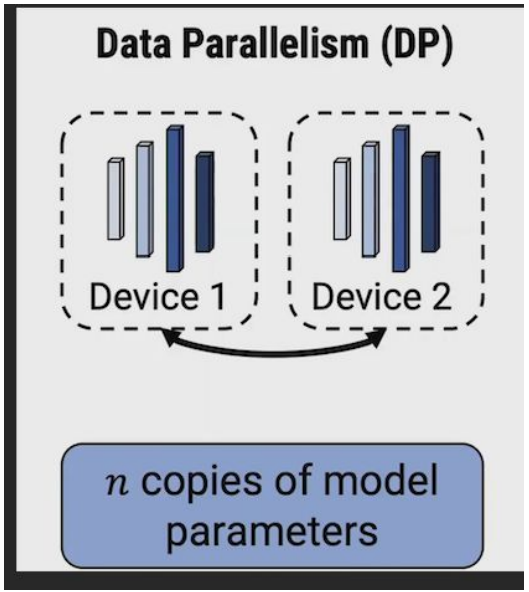
# Megatron-LM: Motivation

- “3D Parallelism” or “PTD-P” if you love acronyms
  - Three Different Methods of Parallelism
  - Combine the three methods based on training workload
  - Parallelizing the workflow adds overhead

# “3D Parallelism”



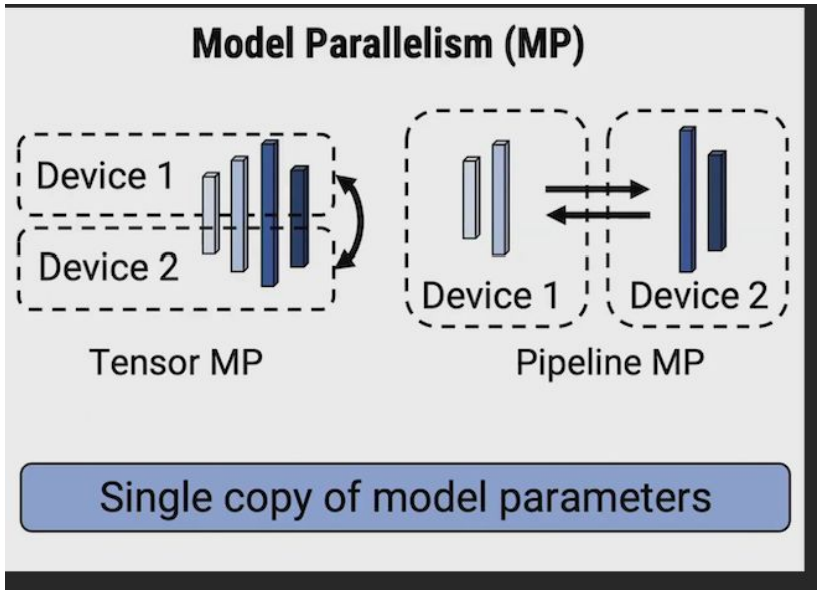
# “3D Parallelism”




- Data
  - Shard input data across multiple nodes
  - Each node processes its data with an entire copy of the model (or a shard of model weights)
  - weights periodically reaggregated

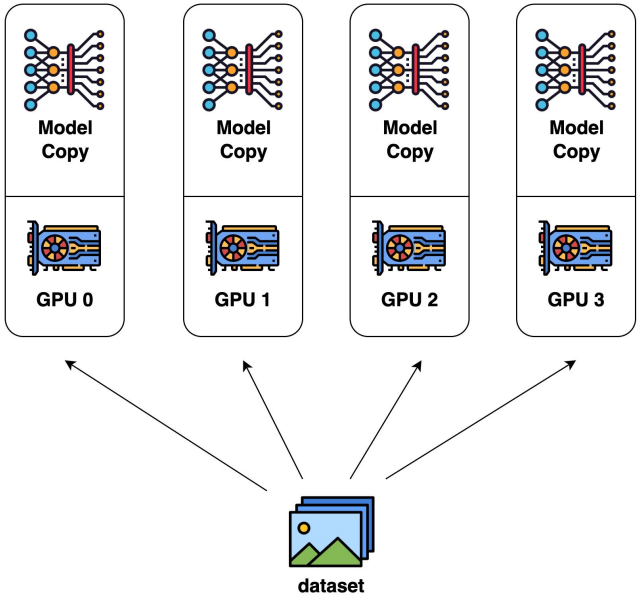
# “3D Parallelism”

- Model Parallelism
  - Split model across GPUs
  - Pipeline MP = split layers
  - Tensor MP = split weights



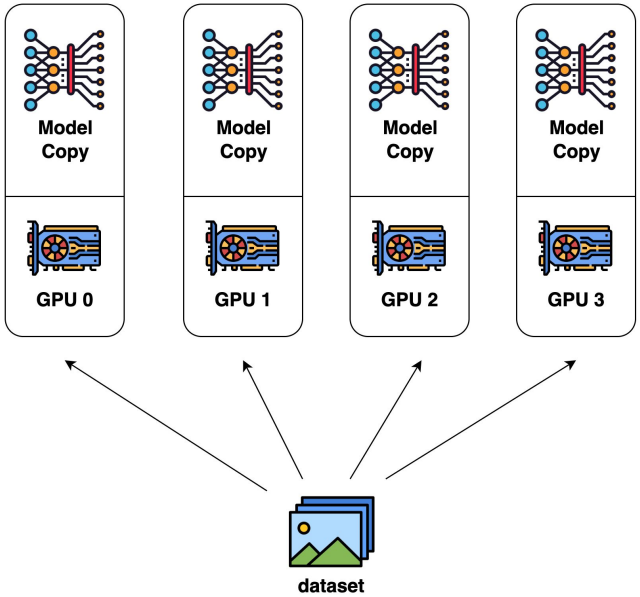
# Megatron-LM: Motivation

- “3D Parallelism”
-



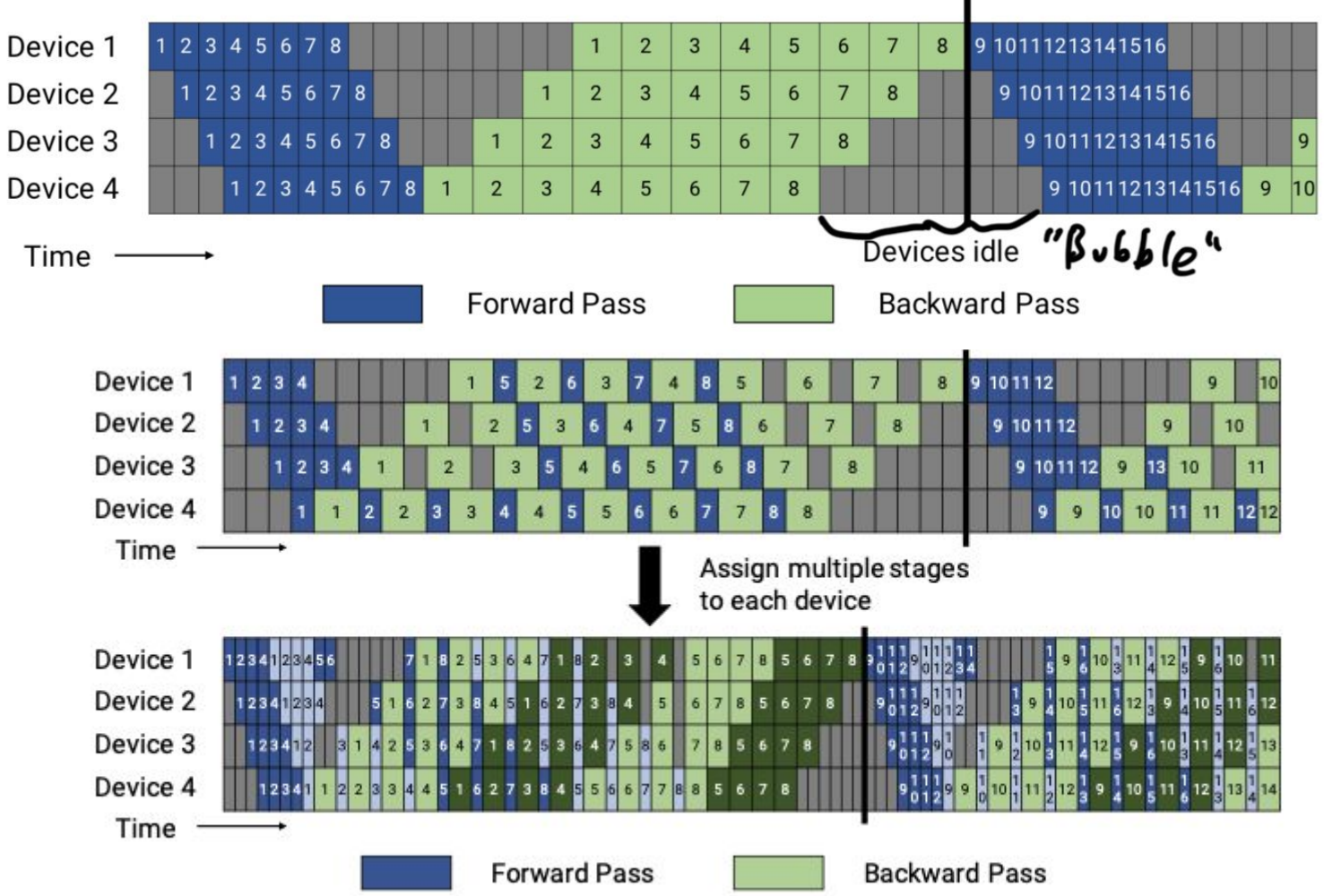
# Megatron-LM: Motivation

- “3D Parallelism”
- Data
  - Shard input data across multiple nodes
  - Each node processes its data with an entire copy of the model (or a shard of model weights)
  - weights periodically reaggregated



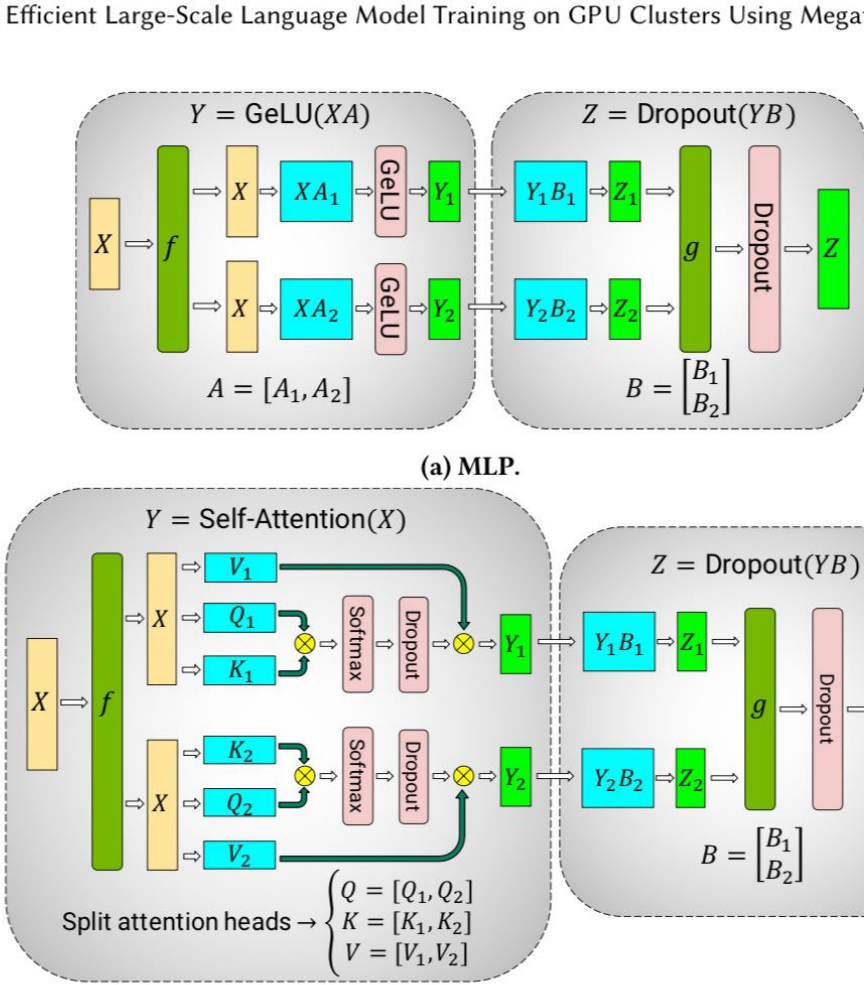
# Megatron-LM: Motivation

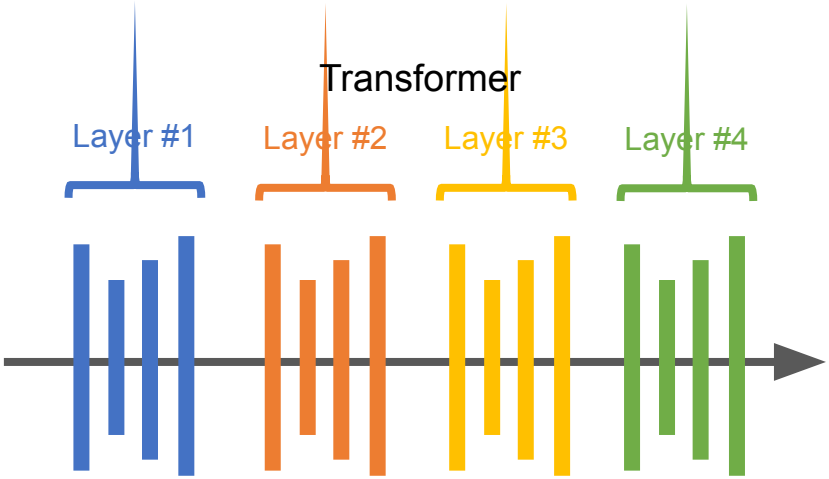
- “3D Parallelism”
- Pipeline Model Parallelism
  - Distribute different layers of model across multiple GPUs, each processing a part of the model
    - Megatron only considers sharding homogenous transformer layers
  - Megatron-LM wants to maintain optimizer correctness, which requires all of the model layers to finish before weights are updated and used
  - Clever pipeline interleavings cut down on the pipeline stalls/“pipeline bubble” time

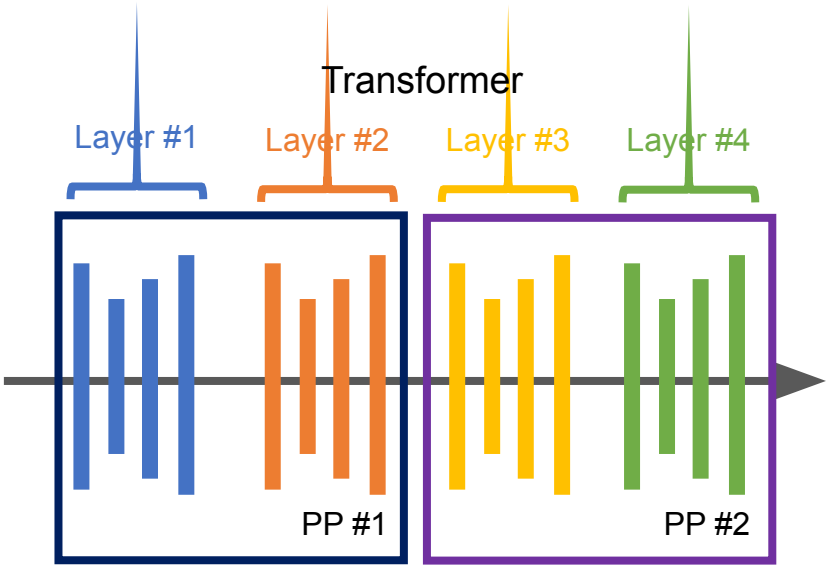

Device 1
Device 2
Device 3
Device 4
Time
Devices idle
"Bubble"
Forward Pass
Backward Pass
Device 1
Device 2
Device 3
Device 4
Time
Assign multiple stages
to each device
Device 1
Device 2
Device 3
Device 4
Time
Forward Pass
Backward Pass

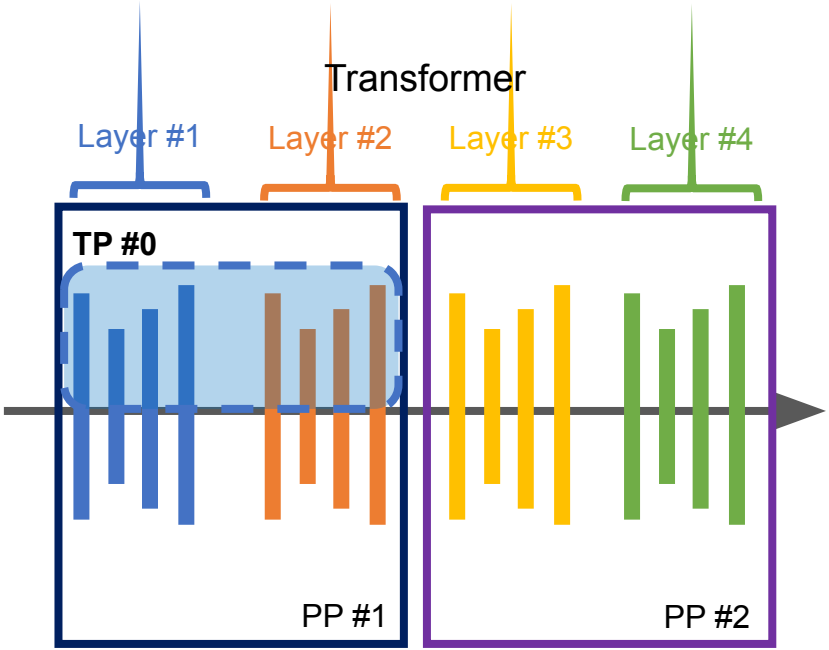
# Megatron-LM: Motivation

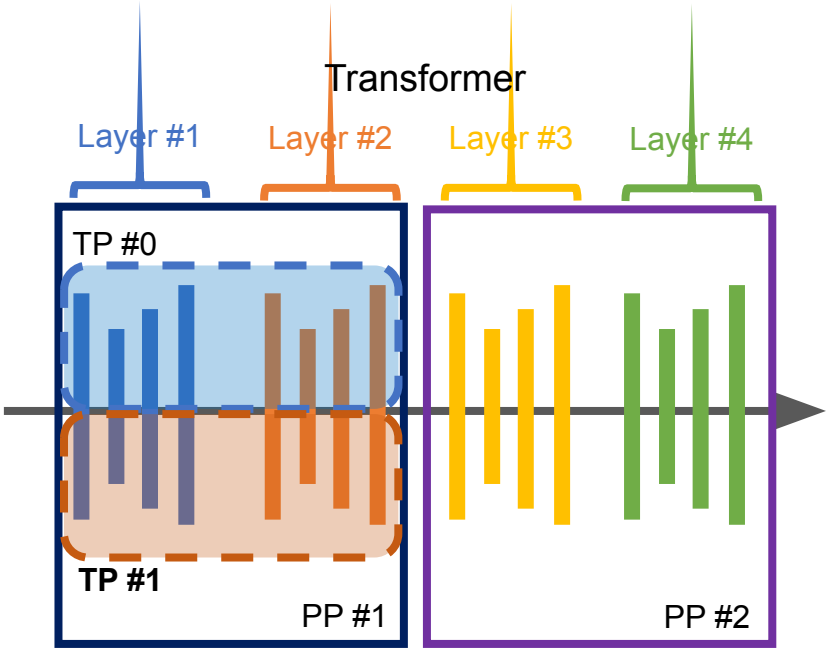
- “3D Parallelism”
- Tensor Model
  - Partition layers of the model and model weights across multiple devices
  - In Megatron, they split the post-attention calculation MLP update across multiple devices
  - Partitioning overhead: 4 all-reduce operations to split up matrices

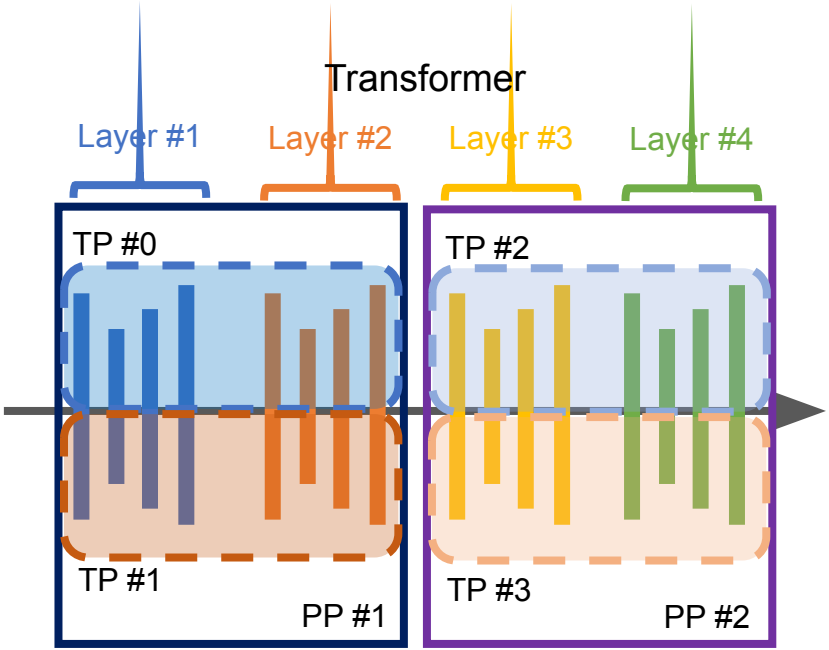



Transformer
Layer #1
Layer #2
Layer #3
Layer #4


Transformer
Layer #1
Layer #2
Layer #3
Layer #4
PP #1
PP #2


Transformer
Layer #1
Layer #2
Layer #3
Layer #4
TP #0
PP #1
PP #2


Transformer
Layer #1
Layer #2
Layer #3
Layer #4
TP #0
TP #1
PP #1
PP #2


Transformer
Layer #1
Layer #2
Layer #3
Layer #4
TP #0
TP #1
PP #1
TP #2
TP #3
PP #2

# Callback to Intro to Transformers presentation

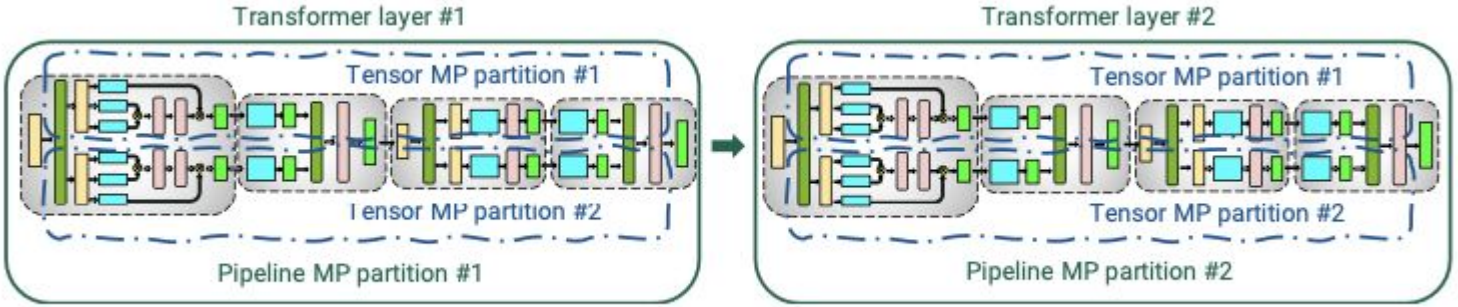
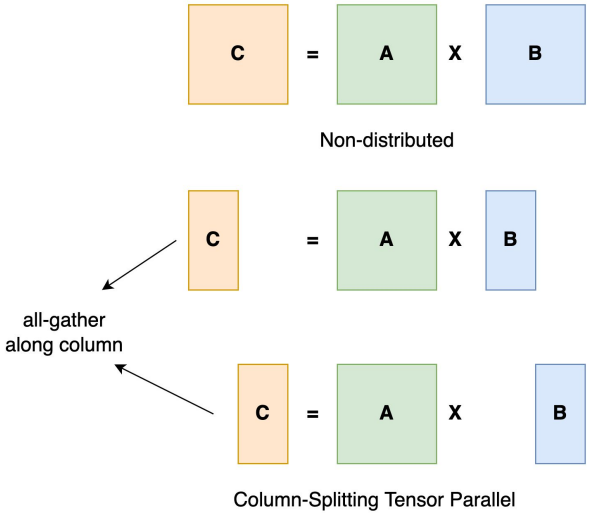



**Figure 2: Combination of tensor and pipeline model parallelism (MP) used in this work for transformer-based models.**

How can we minimize the overhead of parallelization?

# Implementation and Optimizations

- Written in PyTorch
  - github.com/NVIDIA/Megatron-LM
- Communication optimization
  - Scatter-gather (11% speedup)
- Fuse together kernels
  - Also 11% speedup



# Megatron-LM: Contributions

- Combine several parallelism techniques
  - Pipeline parallelism across multi-GPU servers
  - Tensor parallelism within multi-GPU server
  - Data parallelism
- Reason about performance, performance *interactions* of modes of parallelism

# DELETE DELETE Moved to Marwa's Optimization Section Megatron-LM: Contributions

- Describe some heuristics for when each part of parallelism should be applied
  - Tensor + Pipeline Parallelism
    - Reduce "bubble" via tensor parallelism
    - If you have *g* GPUs/server, create *g* tensor partitions, and then apply pipeline model paralleism
  - Data + Model Parallelism
    - Use t (tensor-model-parallel size) * p (pipeline-model-parallel-size) should be used
  - Optimal Microbatch size
    - Depends on everything.

# Megatron-LM: PTD-P

- Combine the “good” of parallelism techniques
  - Pipeline parallelism across multi-GPU servers
  - Tensor parallelism within multi-GPU server
  - Data parallelism
- Along with compute-bound kernel, smart partitioning, good hardware, communication optimization…
  - These are additional optimizations that elevate PTD-P (the core) into Megatron-LM
- And along with “careful” engineering

# Megatron-LM: PTD-P

- Combine the “good” of parallelism techniques
  - Pipeline parallelism across multi-GPU servers
  - Tensor parallelism within multi-GPU server
  - Data parallelism
- Along with compute-bound kernel, smart partitioning, good hardware, communication optimization…
  - These are additional optimizations that elevate PTD-P (the core) into Megatron-LM
- And along with “careful” engineering
- What we get:
  - Graceful scaling
  - Optimized cluster environment
  - High bandwidth links between GPUs on same server and across

# Performance of Different Parallization Configurations

Tensor + Pipeline Model Parallelism

If you have g GPUs, first do g tensor model partitions, then scale out with pipeline model parallelism

Todo add the takeaways from the other combinations

# Megatron-LM: Evaluation

- Experiment setup
  - Mixed precision on Selene Supercomputer
  - For each node/cluster
    - 8 NVIDIA 80-GB A100 GPUs
    - NVIDIA Mellanox HCAs for application communication
- Three-level fat-tree topology
  - Fast all-reduce communication

# Megatron-LM: Evaluation

- Experiment setup
  - Mixed precision on Selene Supercomputer
  - For each node/cluster
    - 8 NVIDIA 80-GB A100 GPUs
    - NVIDIA Mellanox HCAs for application communication
- Three-level fat-tree topology
  - Fast all-reduce communication
- Goal

# Megatron-LM: Evaluation

- Experiment setup
  - Mixed precision on Selene Supercomputer
  - For each node/cluster
    - 8 NVIDIA 80-GB A100 GPUs
    - NVIDIA Mellanox HCAs for application communication
- Three-level fat-tree topology
  - Fast all-reduce communication
- Goal
  - Take large-scale GPT models (mostly just GPT-3)

# Megatron-LM: Evaluation

- Experiment setup
  - Mixed precision on Selene Supercomputer
  - For each node/cluster
    - 8 NVIDIA 80-GB A100 GPUs
    - NVIDIA Mellanox HCAs for application communication
- Three-level fat-tree topology
  - Fast all-reduce communication
- Goal
  - Take large-scale GPT models (mostly just GPT-3)
  - Incorporate Megatron framework

# Megatron-LM: Evaluation

- Experiment setup
  - Mixed precision on Selene Supercomputer
  - For each node/cluster
    - 8 NVIDIA 80-GB A100 GPUs
    - NVIDIA Mellanox HCAs for application communication
- Three-level fat-tree topology
  - Fast all-reduce communication
- Goal
  - Take large-scale GPT models (mostly just GPT-3)
  - Incorporate Megatron framework
  - See how it performs in model training

# Megatron-LM: Evaluation

How well does PTD-P Perform?

# Megatron-LM: Evaluation

How well does PTD-P Perform?

| 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 teraFIOP/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 |

# Megatron-LM: Evaluation

How well does PTD-P Perform?

| 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 teraFIOP/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 |

End-to-end training time ~three months for trillion parameter model

# Megatron-LM: Evaluation

How well does PTD-P Perform?

| 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 teraFIOP/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 |

End-to-end training time ~three months for trillion parameter model

- Before Megatron: Training GPT-3, with 175 billion parameters, takes 288 years with a single V100 NVIDIA GPU

# Megatron-LM: Evaluation

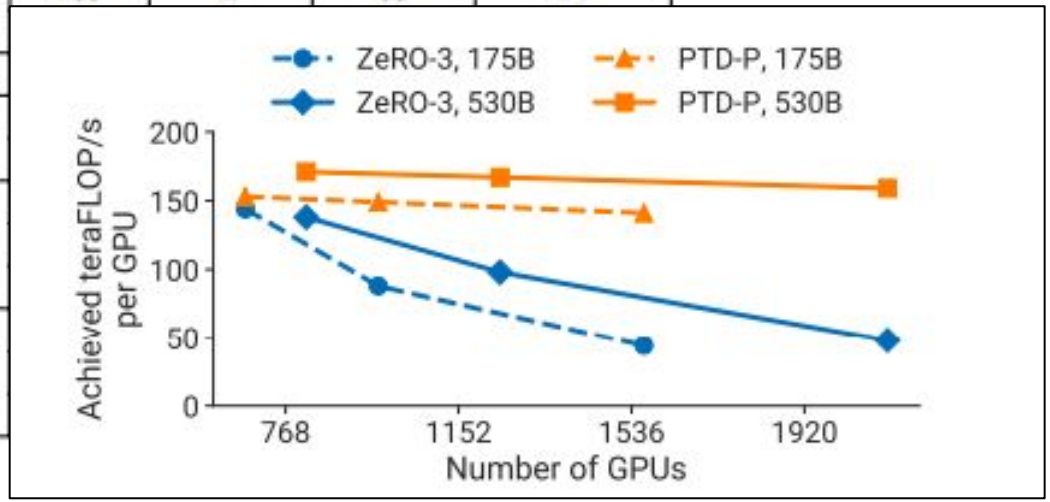
How well does PTD-P Perform against other models?

| Scheme | Number of parameters (billion) | Model-parallel size | Batch size | Number of GPUs | Microbatch size | Achieved teraFlOP/s per GPU | Training time for 300B tokens (days) |
|---|---|---|---|---|---|---|---|
| ZeRO-3 without Model Parallelism | 174.6 | 1 | 1536 | 384 | 4 | 144 | 90 |
| | | | | 768 | 2 | 88 | 74 |
| | | | | 1536 | 1 | 44 | 74 |
| | 529.6 | 1 | 2560* | 640 | 4 | 138 | 169 |
| | | | 2240 | 1120 | 2 | 98 | 137 |
| | | | | 2240 | 1 | 48 | 140 |
| PTD Parallelism | 174.6 | 96 | 1536 | 384 | 1 | 153 | 84 |
| | | | | 768 | 1 | 149 | 43 |
| | | | | 1536 | 1 | 141 | 23 |
| | 529.6 | 280 | 2240 | 560 | 1 | 171 | 156 |
| | | | | 1120 | 1 | 167 | 80 |
| | | | | 2240 | 1 | 159 | 42 |

# Megatron-LM: Evaluation

How well does PTD-P Perform against other models?

| Scheme | Number of parameters (billion) | Model-parallel size | Batch size | Number of GPUs | Microbatch size | Achieved teraFlOP/s per GPU | Training time for 300B tokens (days) |
|---|---|---|---|---|---|---|---|
| ZeRO-3 without Model Parallelism | 174.6 | 1 | 1536 | 384 | 4 | 144 | 90 |
| | | | | 768 | 2 | 88 | 74 |
| | 529.6 | 1 | 2560* | | | | |
| | | | 2240 | | | | |
| PTD Parallelism | 174.6 | 96 | 1536 | | | | |
| | 529.6 | 280 | 2240 | | | | |

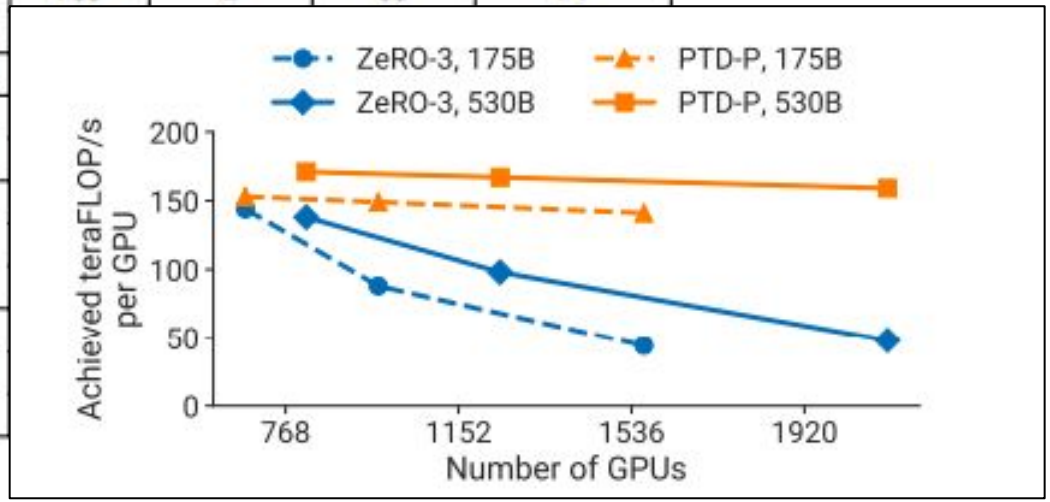


# Megatron-LM: Evaluation

How well does PTD-P Perform against other models?

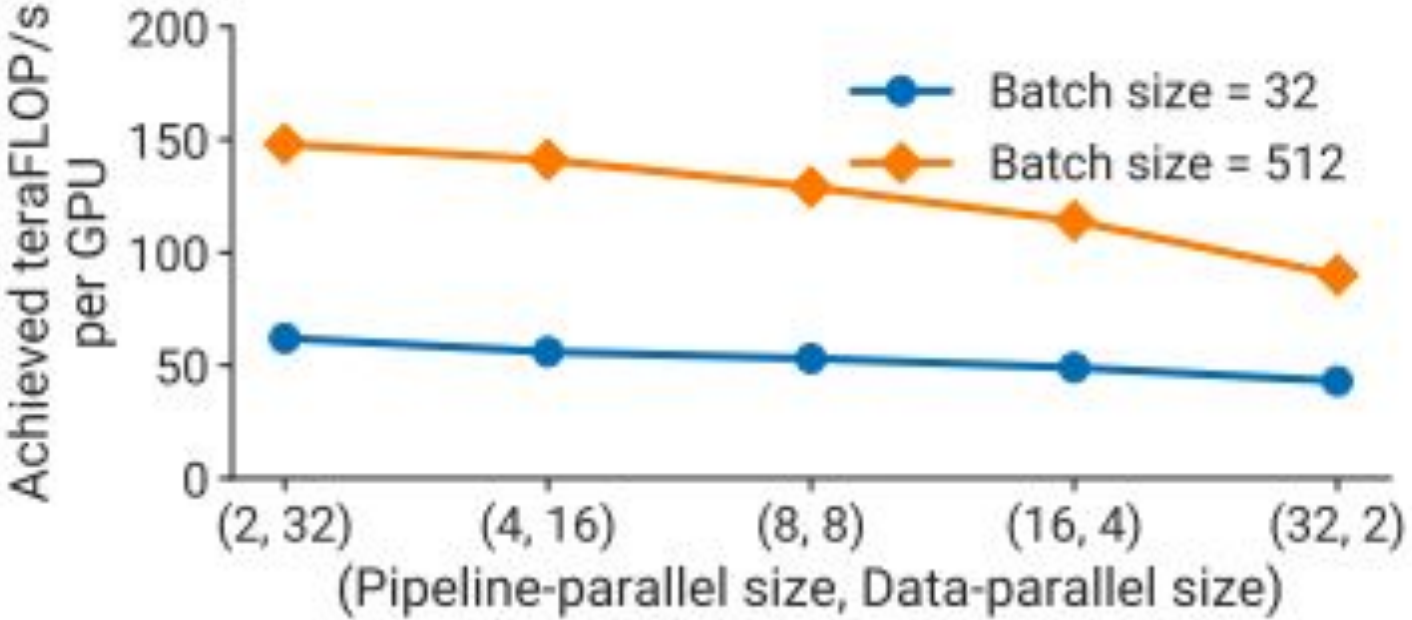
| Scheme | Number of parameters (billion) | Model-parallel size | Batch size | Number of GPUs | Microbatch size | Achieved teraFlOP/s per GPU | Training time for 300B tokens (days) |
|---|---|---|---|---|---|---|---|
| ZeRO-3 without Model Parallelism | 174.6 | 1 | 1536 | 384 | 4 | 144 | 90 |
| | | | | 768 | 2 | 88 | 74 |
| | 529.6 | 1 | 2560* | | | | |
| | | | 2240 | | | | |
| PTD Parallelism | 174.6 | 96 | 1536 | | | | |
| | 529.6 | 280 | 2240 | | | | |

PTD-P scales more gracefully than ZeRO-3 in isolation

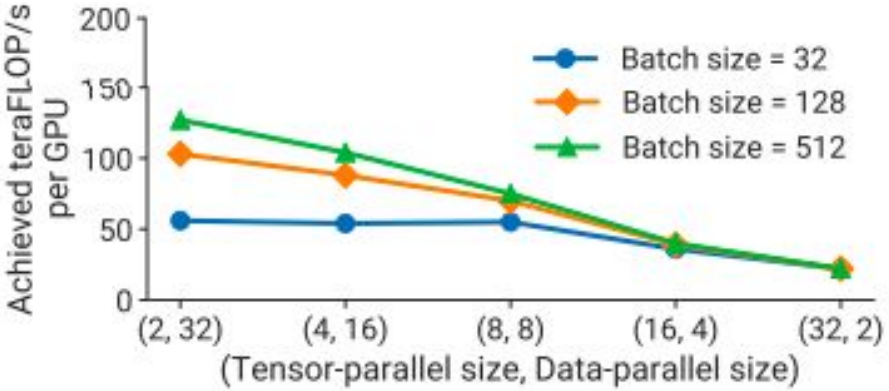


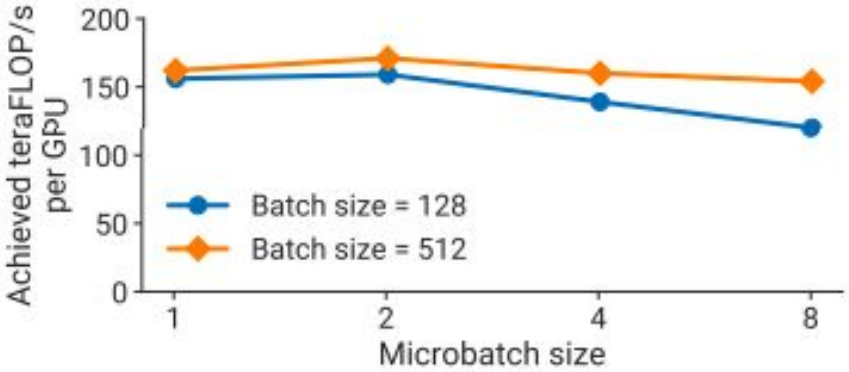
# Megatron-LM: Comparison of Parallel Configurations

# Megatron-LM: Comparison of Parallel Configurations

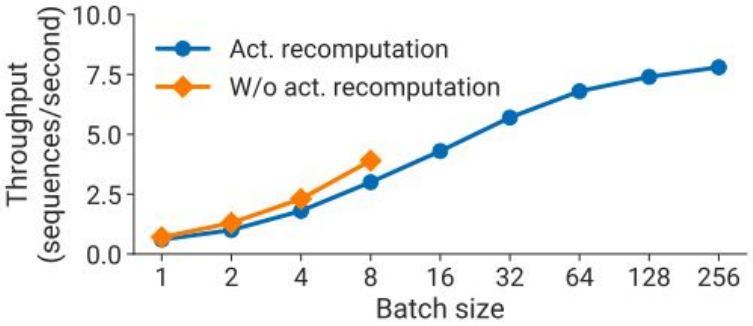


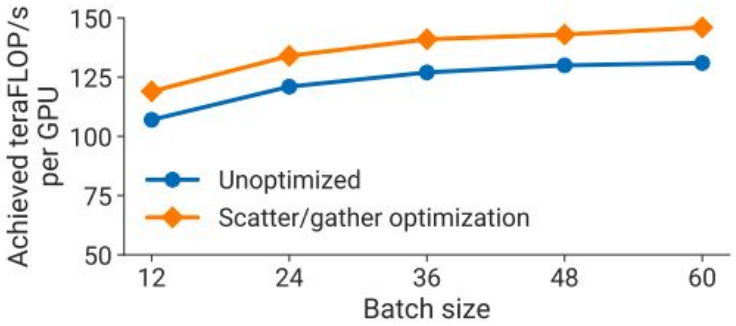
# Comparison of Parallel Configurations, Microbatch Size



# Comparison of Activation Recompution, Scatter/Gather



# Megatron-LM: Discussion

- PTD-P := “3D Parallelism”
  - High aggregate throughput (502 petaFLOP/s) even with exponential parameter growth
  - Reasonable end-to-end training times (3 months for a trillion-parameter model)

# Megatron-LM: Discussion

- PTD-P := “3D Parallelism”
  - High aggregate throughput (502 petaFLOP/s) even with exponential parameter growth
  - Reasonable end-to-end training times (3 months for a trillion-parameter model)
- Each method of parallelism has its own trade-offs
  - Ex: Data Parallel Scale-out
  - But when combined, with “careful” engineering (and some cherry picking), very powerful

# Megatron-LM: Discussion

- PTD-P := “3D Parallelism”
  - High aggregate throughput (502 petaFLOP/s) even with exponential parameter growth
  - Reasonable end-to-end training times (3 months for a trillion-parameter model)
- Each method of parallelism has its own trade-offs
  - Ex: Data Parallel Scale-out
  - But when combined, with “careful” engineering (and some cherry picking), very powerful
- Megatron-LM is very [Nvidia A100] GPU-Centric
  - How can we translate to other accelerators (i.e. TPUs)?

# Megatron-LM: Future Directions

- Future Directions: translate to other types of accelerators

# Megatron-LM: Future Directions

- Future Directions: translate to other types of accelerators
- Accelerator agnostic:
  - Smart partitioning to minimize communication and maintain activity

# Megatron-LM: Future Directions

- Future Directions: translate to other types of accelerators
- Accelerator agnostic:
  - Smart partitioning to minimize communication and maintain activity
  - Minimize number of memory-bound kernels with operator fusion

# Megatron-LM: Future Directions

- Future Directions: translate to other types of accelerators
- Accelerator agnostic:
  - Smart partitioning to minimize communication and maintain activity
  - Minimize number of memory-bound kernels with operator fusion
  - Other domain-specific optimizations  (scatter-gather)

# Megatron-LM: Future Directions

- Future Directions: translate to other types of accelerators
- Accelerator agnostic:
  - Smart partitioning to minimize communication and maintain activity
  - Minimize number of memory-bound kernels with operator fusion
  - Other domain-specific optimizations  (scatter-gather)
- No quantitative measuring of communication overhead (complex network topology, different speeds for intra-server and between-server transfers, etc)

# Megatron-LM: Related Works

- Other techniques to train models at scale
- Pipeline model parallelism
  - Megatron — flushes to ensure strict optimizer semantics
  - TeraPipe — auto-regressive, fine-grained parallelism across tokens
  - PipeTransformer — pipe and data parallelism
  - PipeDream — relaxed semantics; 1-stale weight update w/out expensive flushes
- Sharded Data Parallelism
- Automatic Partitioning
  - Flexflow
- HPC for Model Training
  - ImageNet

# Future of Parallel Computing: Pathways & Megatron-LM

- New types of parallelism often require a constant coordinator, like **Megatron**

# Future of Parallel Computing: Pathways & Megatron-LM

- New types of parallelism often require a constant coordinator, like **Megatron**
- **Pathways** architecture wants a very general computation graph to allow for optimizations **without Megatron**

# Future of Parallel Computing: Pathways & Megatron-LM

- New types of parallelism often require a constant coordinator, like **Megatron**
- **Pathways** architecture wants a very general computation graph to allow for optimizations **without Megatron**
- Two divergent approaches:
  - Pathways: universal and adaptable computation graph
  - Megatron: coordinator-reliant parallelism upon existing SPMD systems

# Future of Parallel Computing: Pathways & Megatron-LM

- New types of parallelism often require a constant coordinator, like **Megatron**
- **Pathways** architecture wants a very general computation graph to allow for optimizations **without Megatron**
- Two divergent approaches:
  - Pathways: universal and adaptable computation graph
  - Megatron: coordinator-reliant parallelism upon existing SPMD systems
- Two approaches to the future of parallel computing in ML

# Future of Parallel Computing: Pathways & Megatron-LM

- New types of parallelism often require a constant coordinator, like **Megatron**
- **Pathways** architecture wants a very general computation graph to allow for optimizations **without Megatron**
- Two divergent approaches:
  - Pathways: universal and adaptable computation graph
  - Megatron: coordinator-reliant parallelism upon existing SPMD systems
- Two approaches to the future of parallel computing in ML

Add layers of complexity to current architecture or start from scratch?

# Q&A