EECS 6895 Adv. Big Data and AI

Lecture 4: Distributed Training

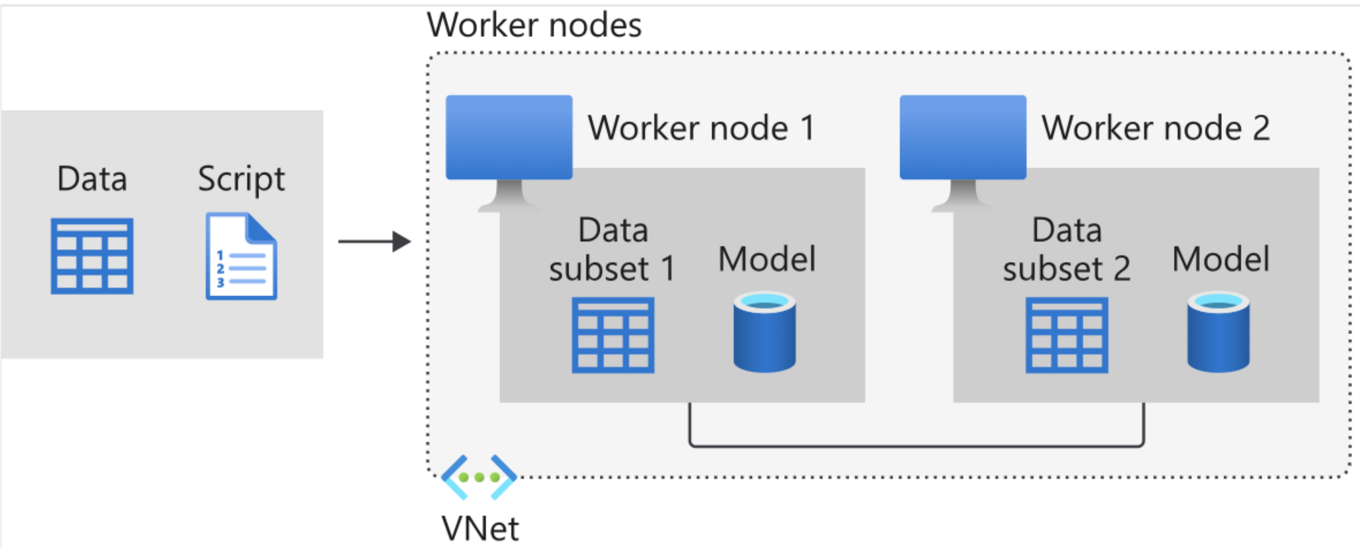
Prof. Ching-Yung Lin

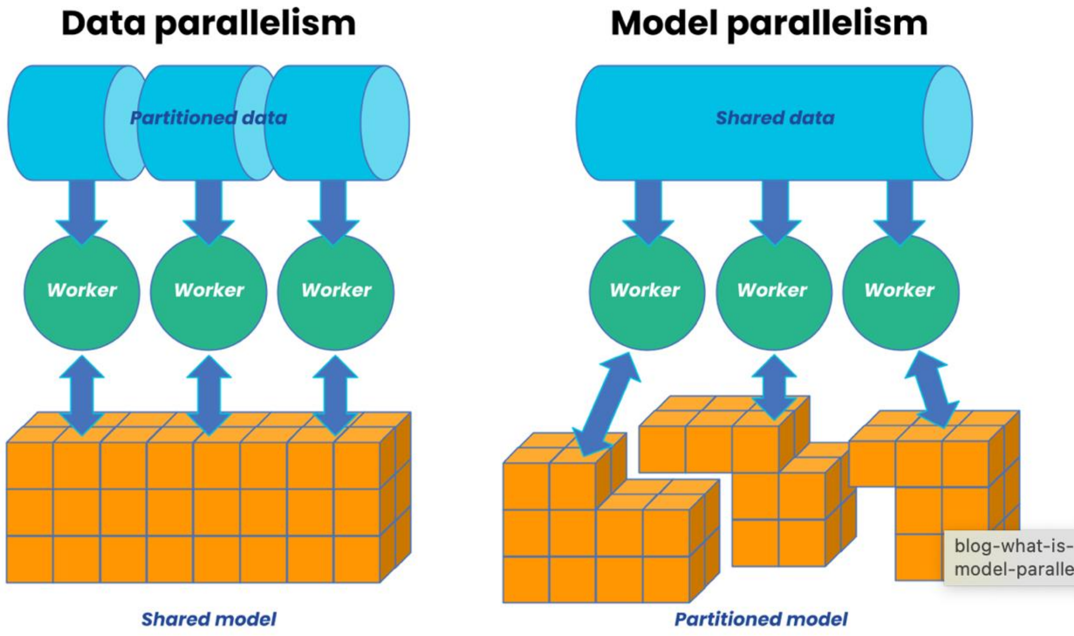
Columbia University

February 11th, 2025

Distributed Training

Distributed Training is to execute Machine Learning or Deep Learning tasks into subtasks run on multiple parallel machines.



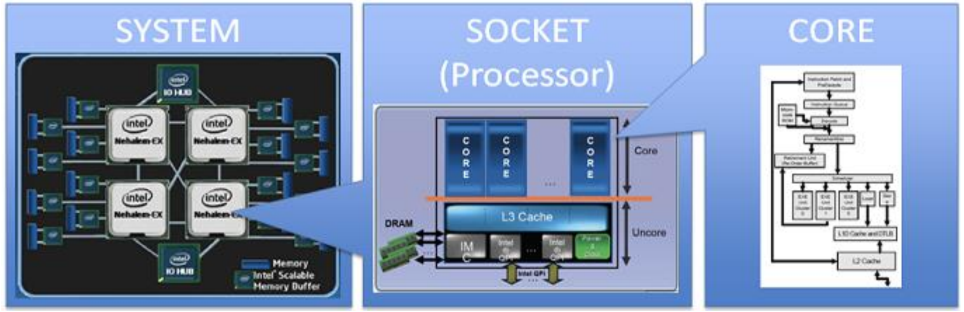


https://learn.microsoft.com/en-us/azure/machine-learning/concept-distributed-training?view=azureml-api-2

https://www.anyscale.com/blog/what-is-distributed-training

Distribution Training Speed

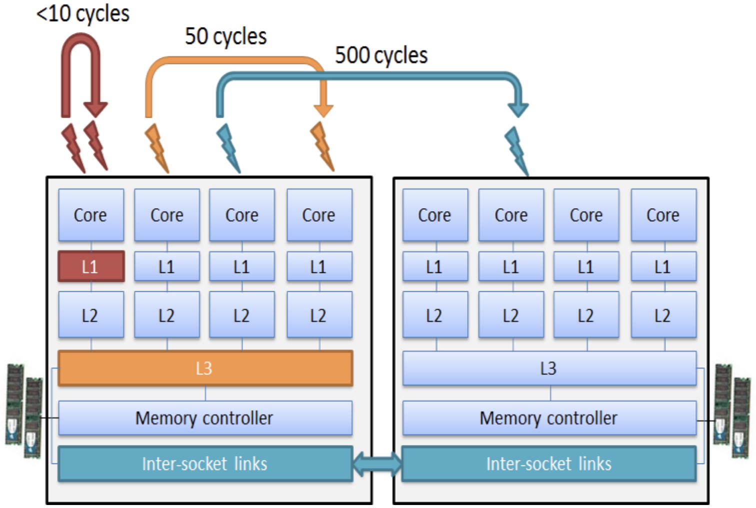
Training Speed is proportional to *speed of single device* x *quantity of devices* x *speed up of multi-devices*

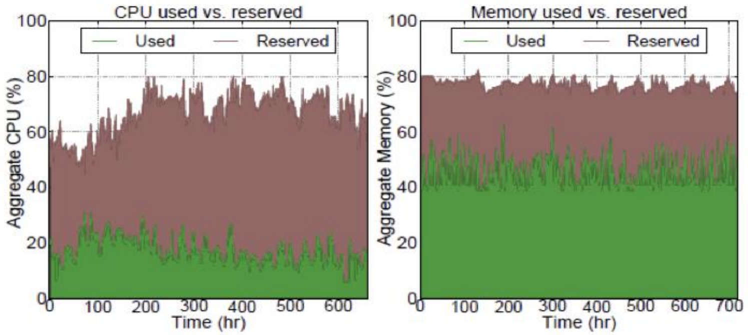


- Speed of Single Device:
  - Single Chip speed
  - Data I/O speed
    - Hybrid Precision Training
    - Computational Fusion
    - Gradient Addition

- Quantity of Devices:
  - Depending on the communication o devices

- Multi-device acceleration:
  - Combining algorithm and network topology





Training Time and Device Comparisons

- GPT-3: Nvidia V100 GPU
- OPT:
  - 992 Nvidia A100 80GB GPU
  - Fully Sharded Data Parallel
  - Megatron-LM Tensor Parallelism
  - 2 months

- BLOOM:
  - 384 GPU → 48 x 8 Nvidia A100 80GB GPU
  - Using 4 x NVLink for GPU internal communications
  - Using 4 Omni-Path 100GBps card for enhanced 8-dim communication
  - 3.5 months

- Llama:
  - Nvidia A100 80GB GPU
  - Llama-7B 82,432 GPU hours
  - Llama-13B 135,168 GPU hours
  - Llama-33B 530,432 GPU hours
  - Llama-65B 1,022,362 GPU hours

Hardware Computational Challenges

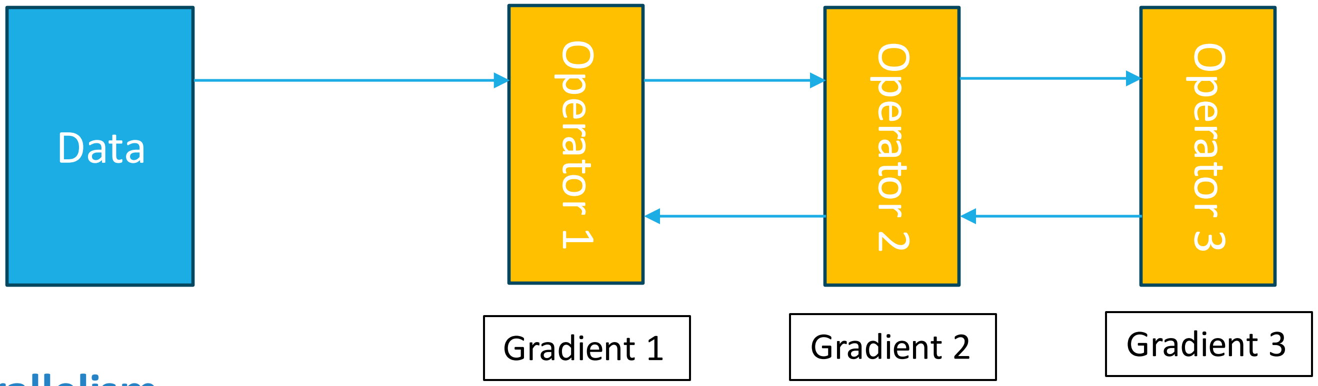
- Computational Wall:
  - Nvidia H100 SXM (March 2022): FP 16 -> 2000 TFLOPS (Floating Point Operations per Second)
  - GPT-3 needs 314 ZFLOPS

- Memory Wall:
  - GPT-3 uses 175B parameters
  - Using FP32, it needs 700GB
  - However, H100 GPU has only 80GB memory

- Communication Wall:
  - GPT-3 if using 128 epoch, then each iteration needs to transmit 89.6 TB.
  - However, InfiniBand link provides less than 800Gbps.

Parallelization Strategy

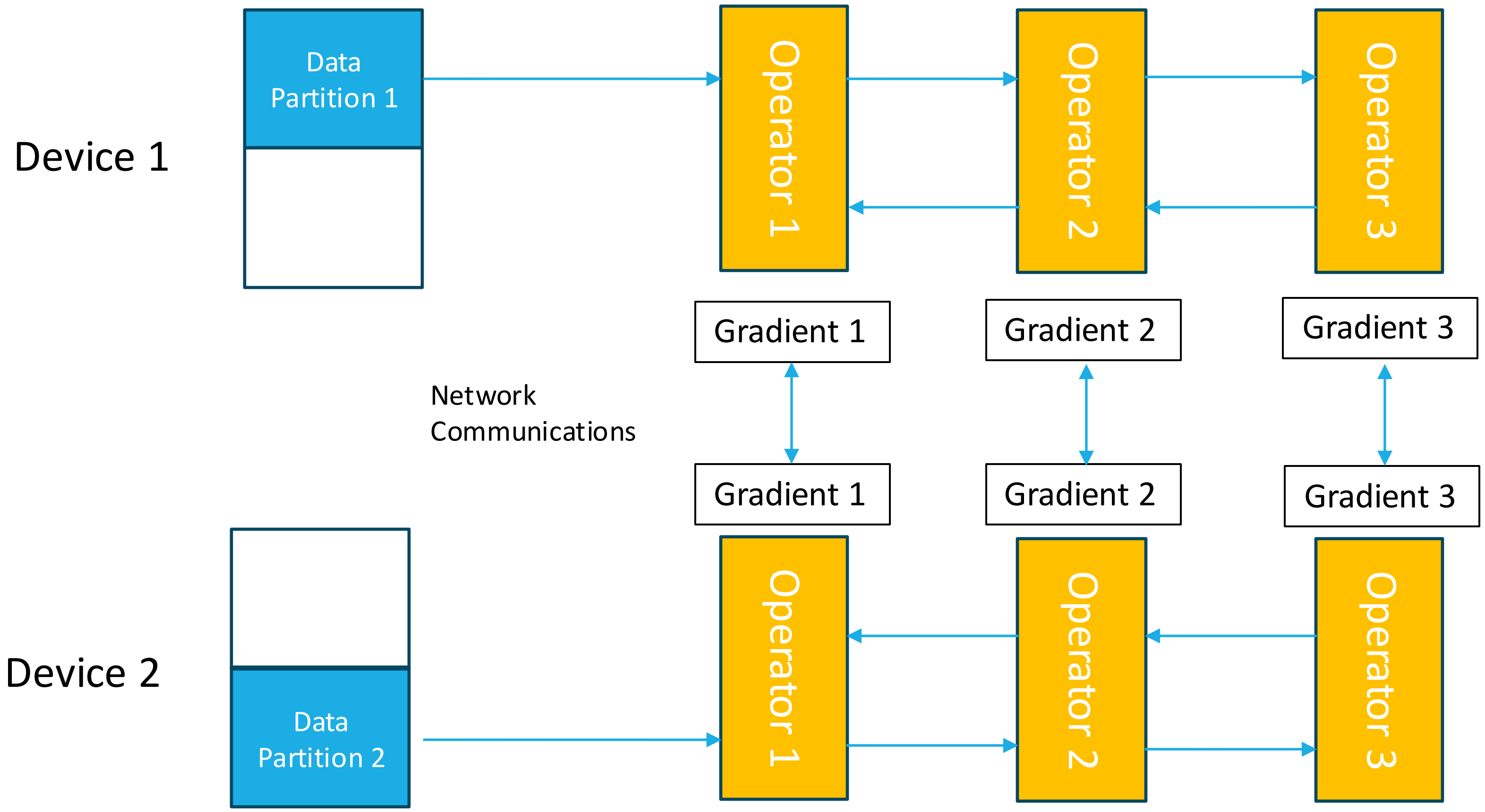
- Computation is based on Data and Model
- Data are divided into Mini-batch.
- Training system uses mini-batch’s loss function and optimization to modify the parameters
- Execution of LLM multi-layer neural network can be described by a Computational Graph.
- This graph has multiple connected Operations.
- Each operator executes a Neural Network Lawyer.
- Parameters represents the trained weights.



Data Parallelism, Model Parallelism, or Hybrid Parallelism

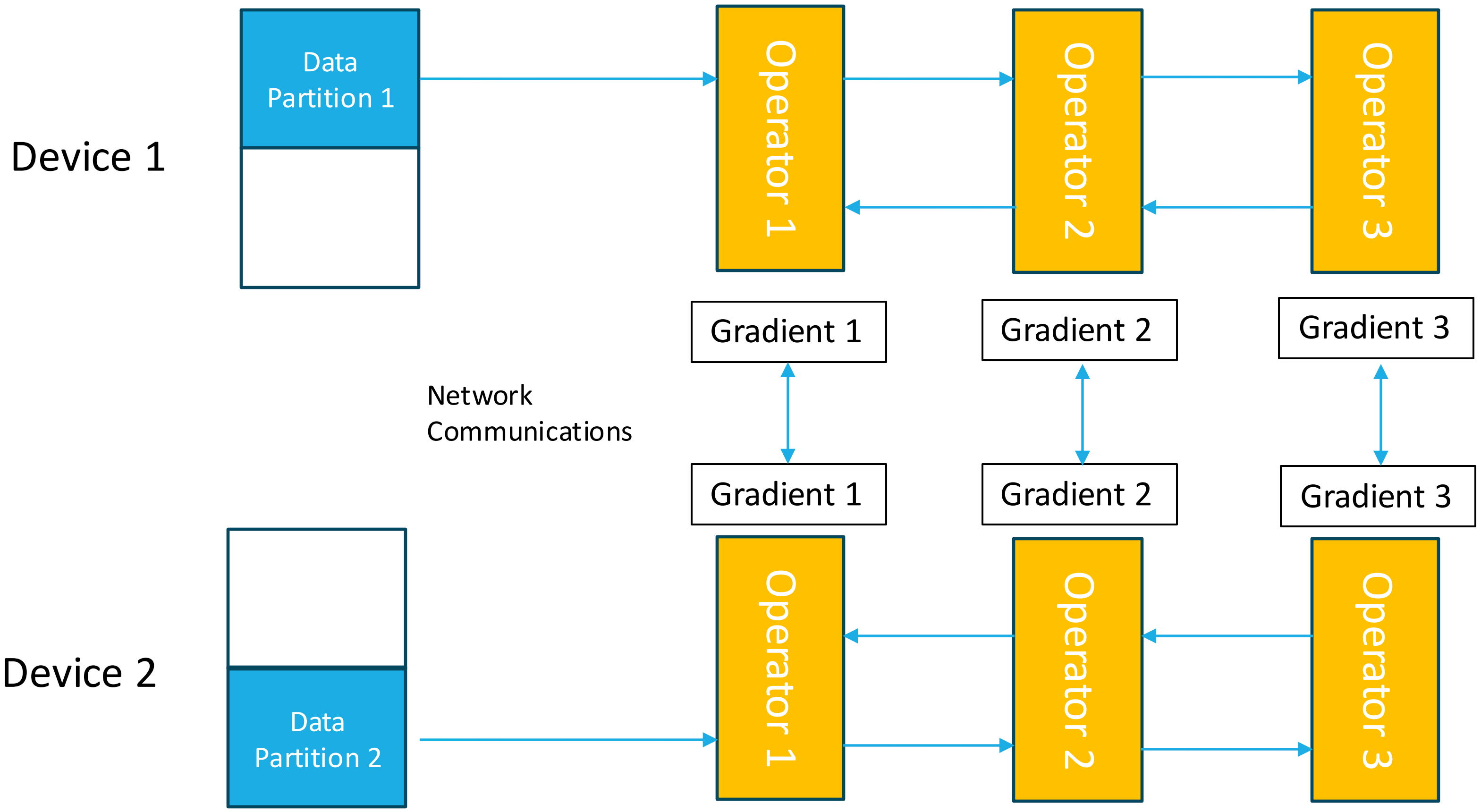
Data Parallelism

- Every device has a Model Replica of the entire neural network
- In each iteration, each device only process a subset of a mini-batch.
- Using this data for forward computation
- Each local ***Gi*** propagates its result to all devices
- All devices combine all new ***Gi***, and use the average to update the model



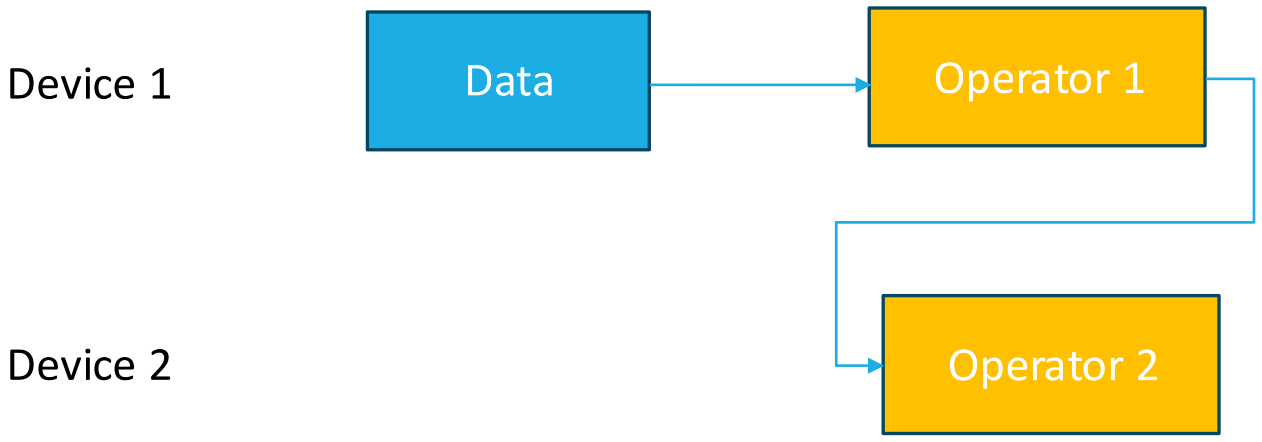
Data Parallelism – Global Batch Size Per Second

- Synchronized computation of all devices' gradient computations at the backward computation.
- Make sure all devices get the average of gradients.
- Usual strategies include:
  - Tensor Flow Distributed Strategy
  - PyTorch Distrubted
  - Horovod Distributed Optimzer
- Pros: data are parallelized. Each computation is relative independent.
- Cons: each device has a backup of the whole model. Requires more memory

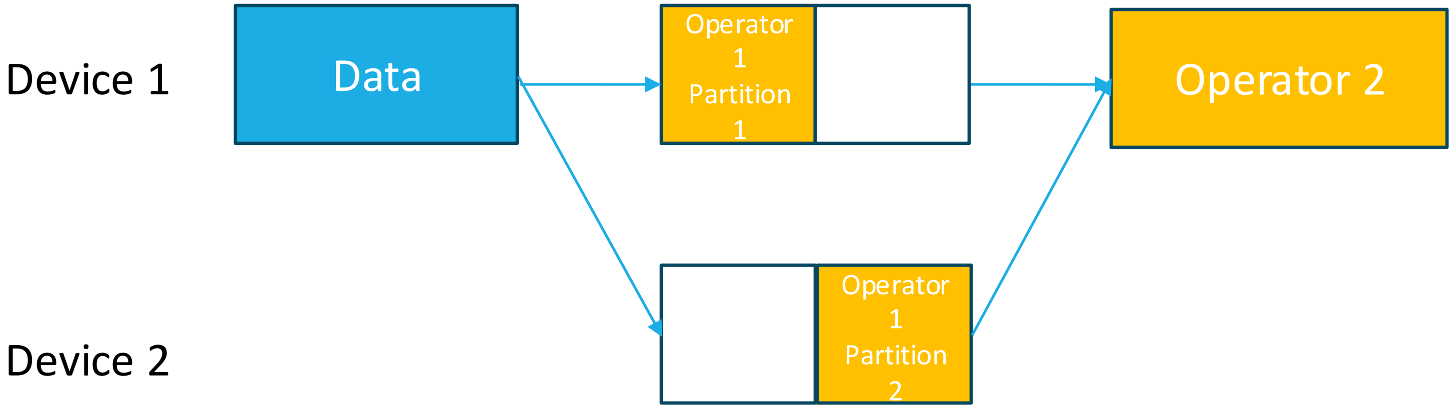


Model Parallelism

- Inter-operator Parallelism a.k.a. Pipeline Parallelism
- Intra-operator Parallelism, a.k.a. Tensor Parallelism
- E.g.: GPT-3 has 175B parameter.
  - If each parameter uses 32 FP ➔ 700GB memory
  - If each parameter uses 16 FP ➔ 350GB memory
- However, H100 only supports 80GB memory

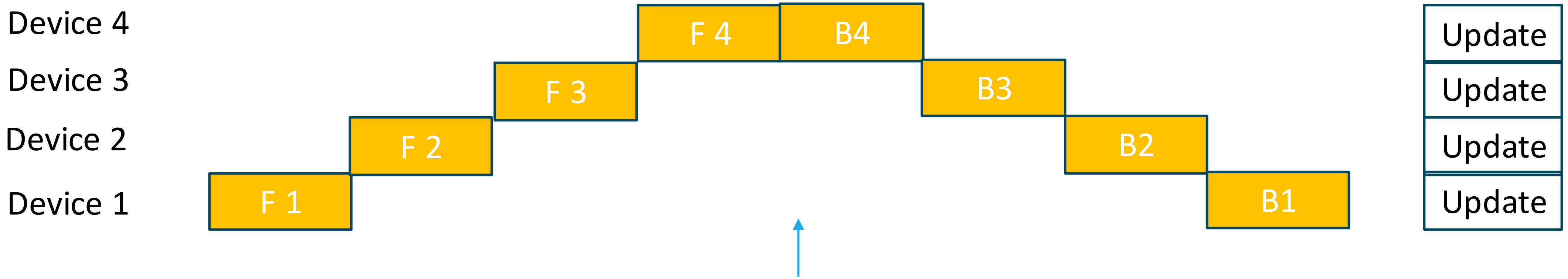


Pipeline Parallelism

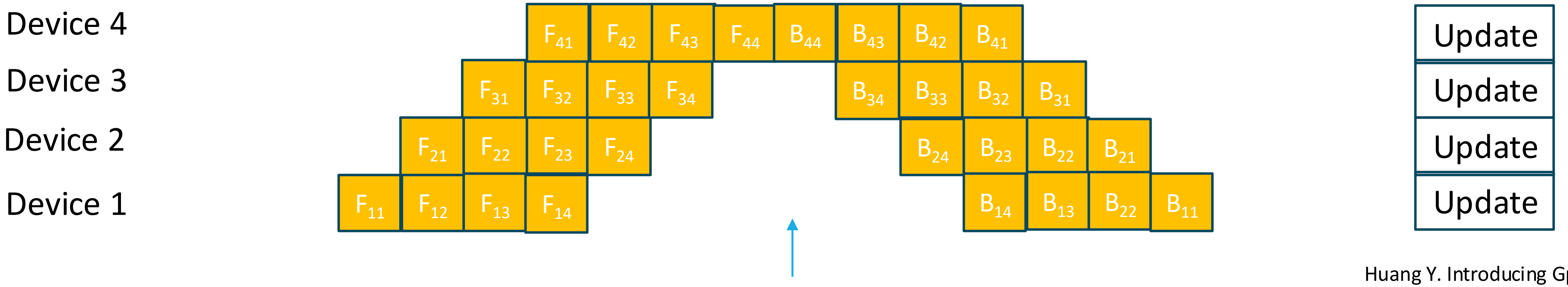


Tensor Parallelism

Pipeline Parallelism challenge - Pipeline Bubble



Model Parallelism Bubble / Pipeline Bubble



Reducing Pipeline Bubble by the Gpipe Micro-batch Strategy

Huang Y. Introducing Gpipe, an open source library for efficient training large scale neural network models. Google AI Blog, March 2019.

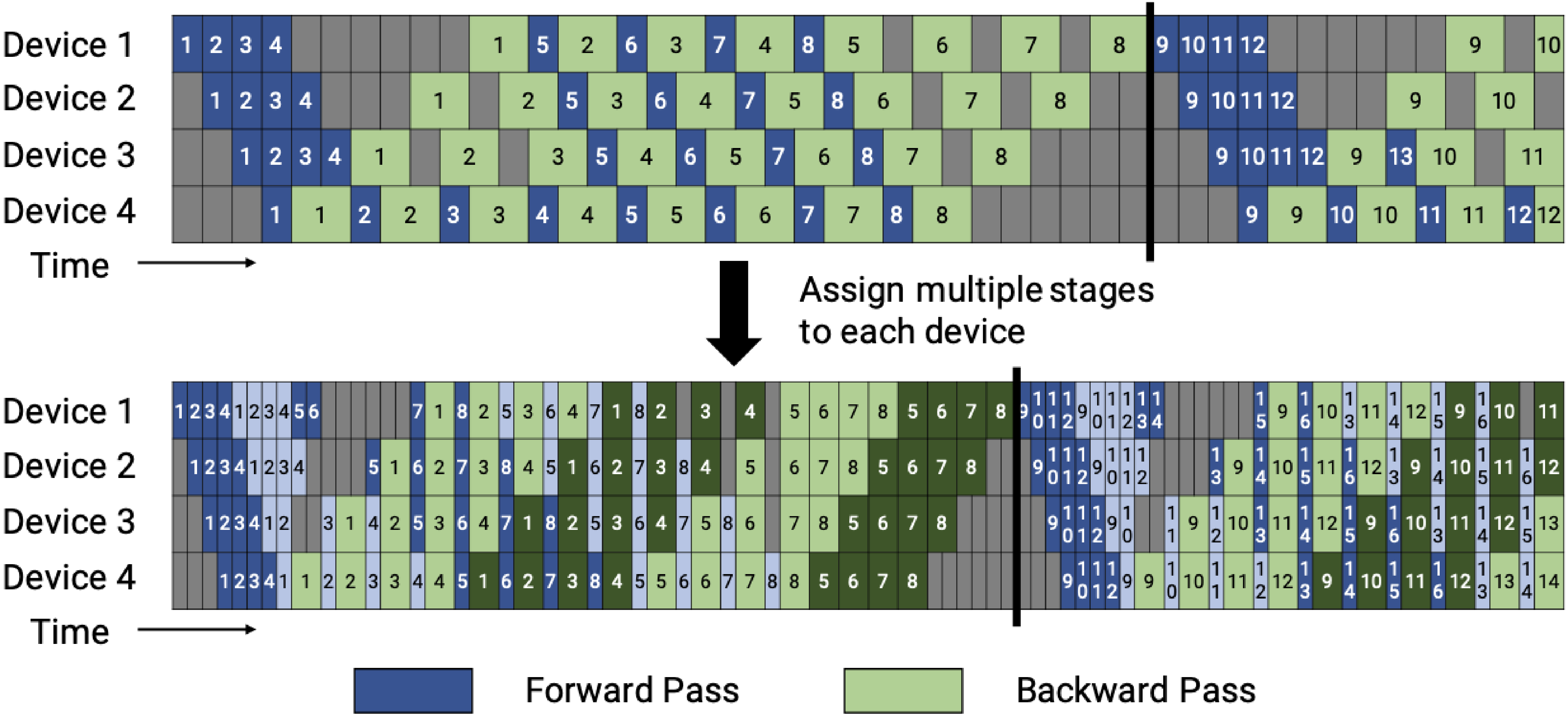
1F1B Pipeline Scheduling

Default 1F1B Pipeline Schedules

Saves memory

Interleaved 1F1B Pipeline Schedules

Saves memory & increase computational efficiency



Dark color shows the first chunk of a layer. Light color shows the second chunk of a layer.

Narayanan etc.. Efficient Large-Scale Language Model Training on GPU Clusters using Megatron-LM., Prof. of Int. Conf. on High Performance Computing, Networking, Storage and Analysis 2021..

Tensor Parallelism

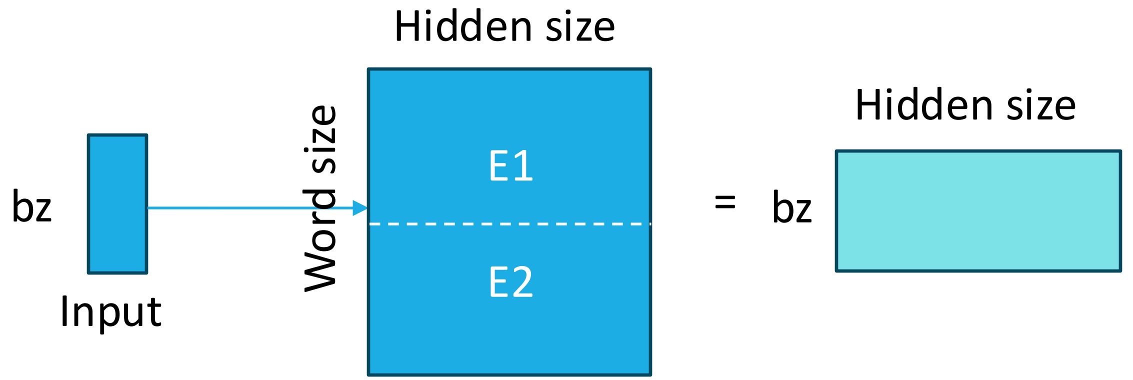
- Tensor Parallelism divides parameters to different devices based on model structure and operators.

- LLMs are based on Transformers which mainly include three major computation modules:
  - Embedding
  - Matrix Multiplication (MatMul)
  - Cross Entropy Loss

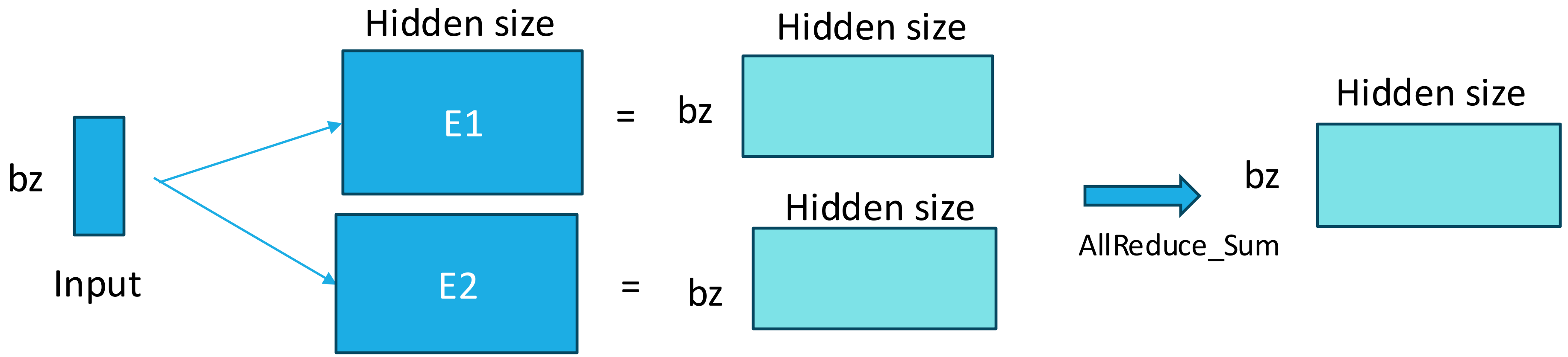
Tensor Parallelism -- Embedding

- If the total number of words are big, then memory cannot handle embedding parameters.
  - E.g.: 64,000 words with dimension of 5120, with 32-bit FP ➔ 64000 x 5120 x 4/1024/1024 = 1250 MB
  - Backward gradients also need 1250MB.
  - ➔ Needs 2.5GB to store

Single-Node Embedding Tensor Parallelism

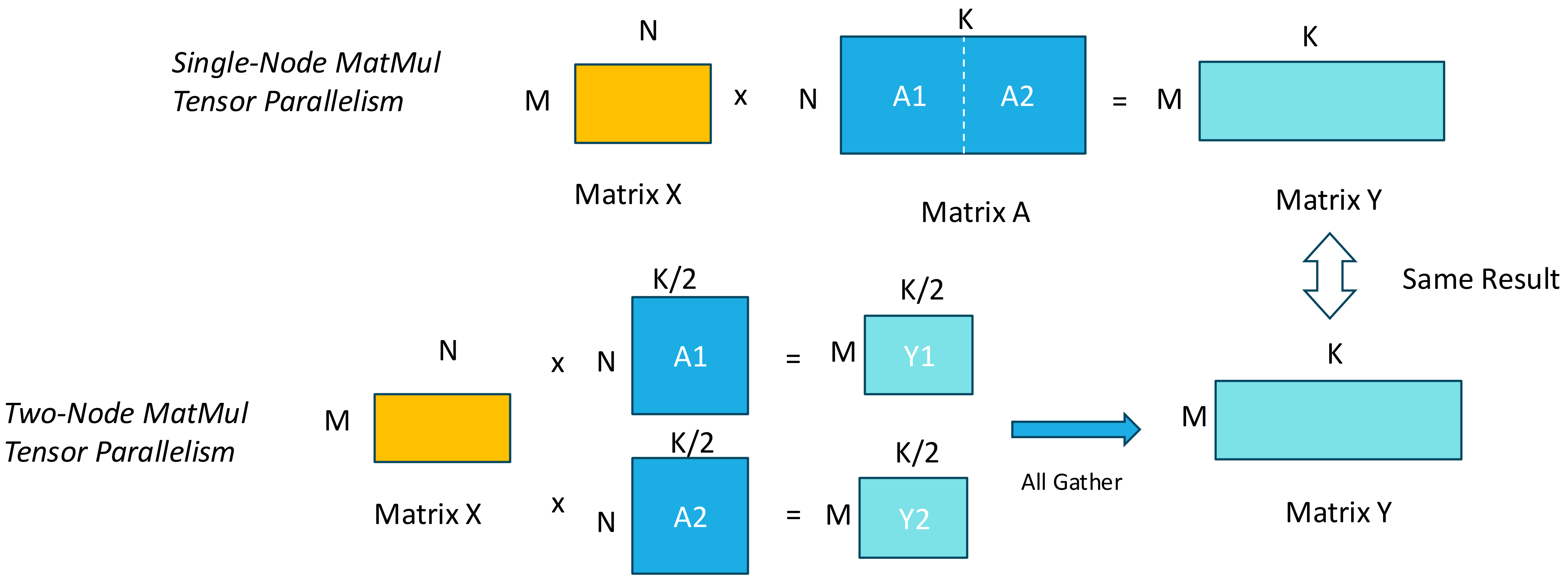


Two-Node Embedding Tensor Parallelism



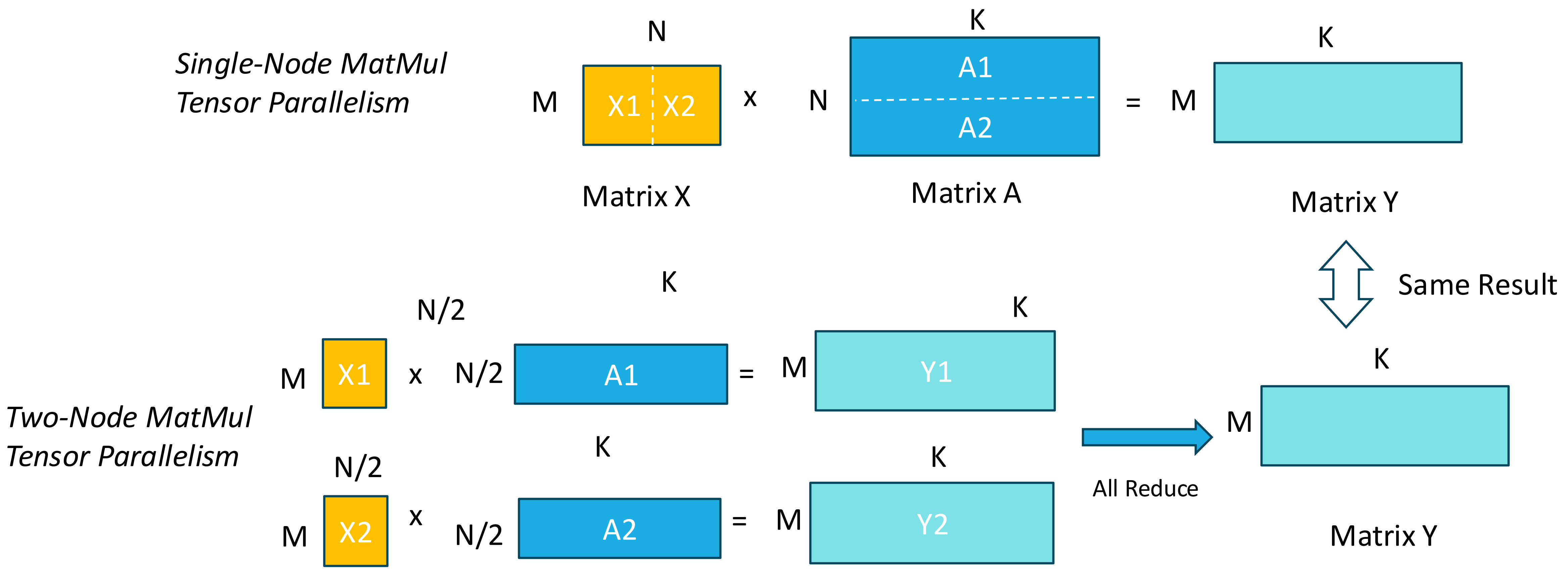
Tensor Parallelism – Matrix Multiplication I

- Partition Matrix
- Divided into multiple devices to accommodate the memory constraints
- Results are the same

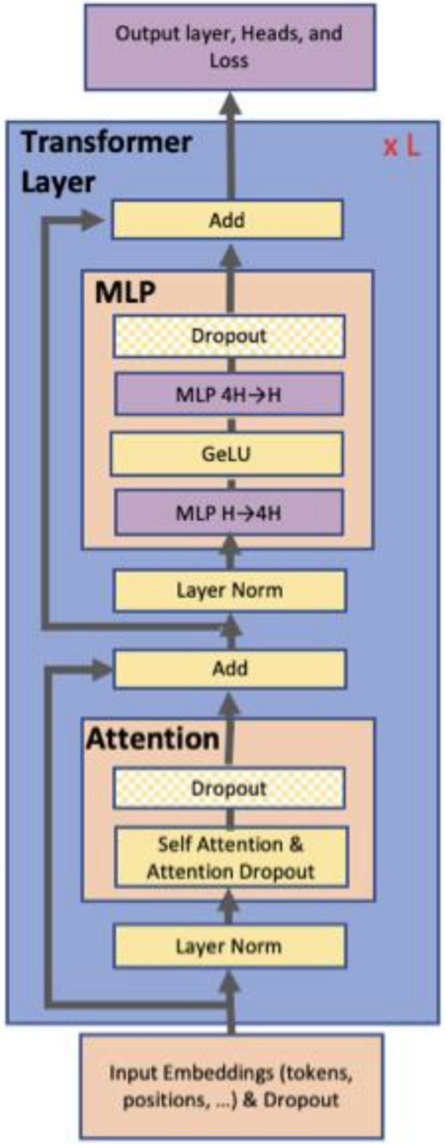


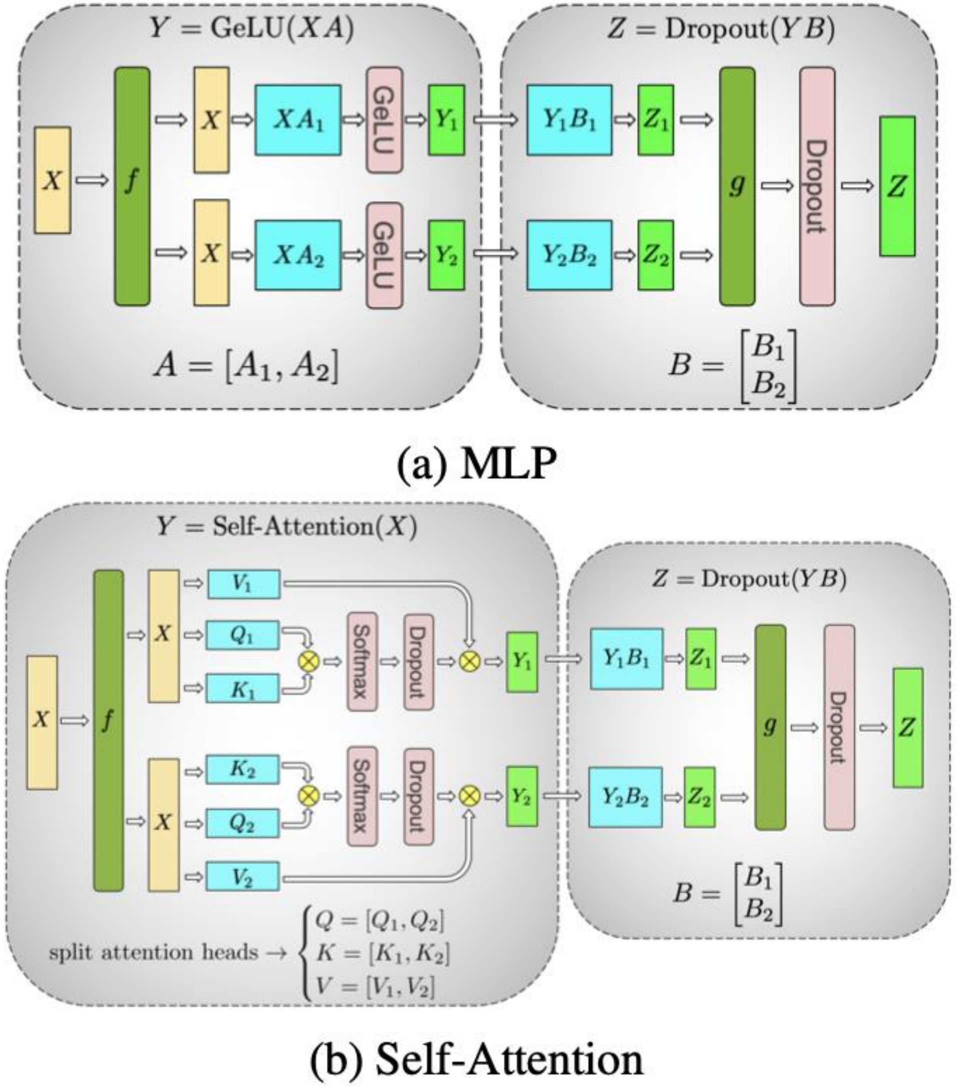
Tensor Parallelism – Matrix Multiplication II

- Different Partitions



Transformer's Tensor Parallelism I





(a) MLP

(b) Self-Attention

Sheoybi, Patwary, Puri, et. Al. Megatron-lm: Training multibillion parameter language models using model parallelism. ArXiv: 1909.08053, 2019.

Transformer's Tensor Parallelism II

Table 1. Parameters used for scaling studies. Hidden size per attention head is kept constant at 96.

Hidden Size	Attention heads	Number of layers	Number of parameters (billions)	Model parallel GPUs	Model +data parallel GPUs
1536	16	40	1.2	1	64
1920	20	54	2.5	2	128
2304	24	64	4.2	4	256
3072	32	72	8.3	8	512

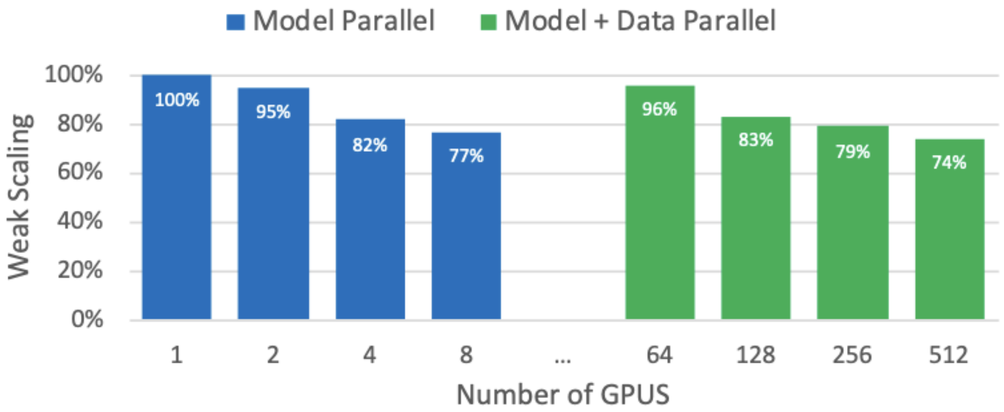


Figure 5. Model and model + data parallel weak scaling efficiency as a function of the number of GPUs.

Table 2. Model configurations used for GPT-2.

Parameter Count	Layers	Hidden Size	Attn Heads	Hidden Size per Head	Total GPUs	Time per Epoch (days)
355M	24	1024	16	64	64	0.86
2.5B	54	1920	20	96	128	2.27
8.3B	72	3072	24	128	512	2.10

Table 3. Zero-shot results. SOTA are from (Khandelwal et al., 2019) for Wikitext103 and (Radford et al., 2019) for LAMBADA.

Model	Wikitext103 Perplexity ↓	LAMBADA Accuracy ↑
355M	19.31	45.18%
2.5B	12.76	61.73%
8.3B	**10.81**	**66.51%**
Previous SOTA	15.79	63.24%

Sheoybi, Patwary, Puri, et. Al. Megatron-lm: Training multibillion parameter language models using model parallelism. ArXiv: 1909.08053, 2019.

Tensor Parallelism – Softmax / Cross Entropy Loss

- If the computational categories are big, Softmax / Cross Entropy Loss layer will make the results too big to store.

➔ Calculate Softmax values based on partitioning dimensions:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j (e^{x_j})} = \frac{e^{x_i - x_{max}}}{\sum_j (e^{x_j - x_{max}})} = \frac{e^{x_i - x_{max}}}{\sum_N \left(\sum_j (e^{x_j - x_{max}})\right)}$$

$$x_{max} = \max_p(\max_k(x_k))$$

p : Device

Hybrid Parallelism

- Hybrid Parallelism combines different parallelism strategies including Data Parallelism, Pipeline Parallelism, and Tensor Parallelism.

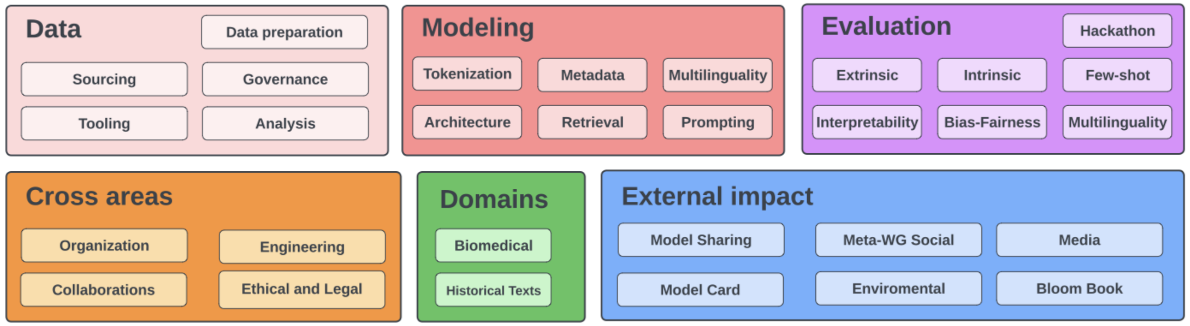
- Requires high speed communication bandwidth.

- Steps:
  - Use Pipeline Parallelism, divide models into different stages using different machines.
  - Use Aggregation Data Parallelism to include training efficiency.

- Example of BLOOM:
  - Betatron-LM provides Tensor Parallelism and Data Input
  - DeepSpeed provides ZeRO optimizer, Pipeline Parallelism, and Distributed Training Components.

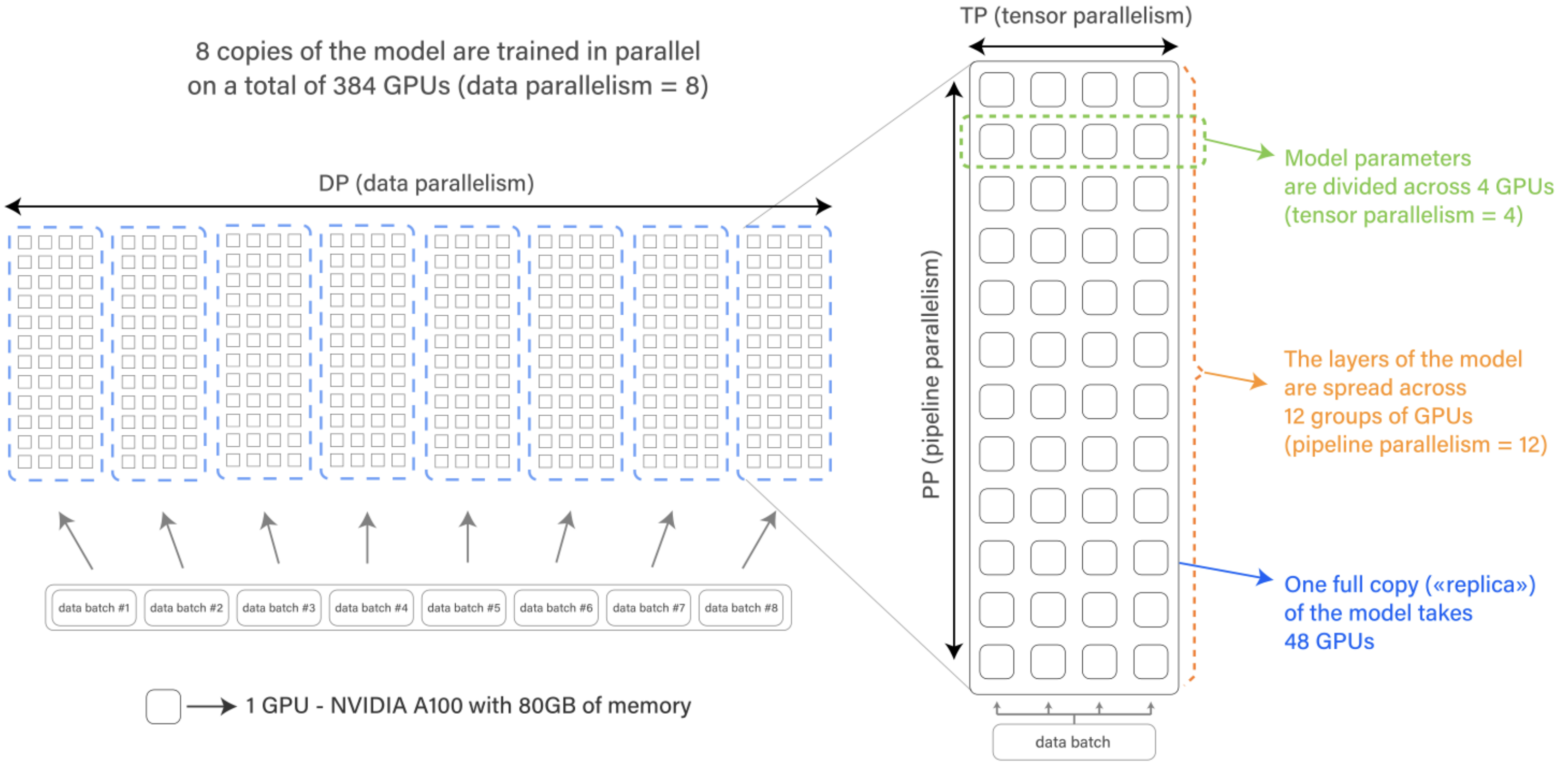
  ➔ Realize all Data, Pipeline, and Tensor Parallelism.

BigScience Large Open-science Open-access Multilingual Language Model (BLOOM) => more than 1200 contributors



Hybrid Parallelism – BLOOM example

- Bloom training uses 48 Nvidia DGX-A100 clusters. Each cluster includes 8 Nvidia A100 80GB GPU ➔ 384 GPUs.
- Data Parallelism is divided into 48 groups.
- Each Model is divided into 12 steps, using Pipeline Parallelism.
- Each Step is divided into 4 GPUs to do Tensor Parallelism.
- Using ZeRO to reduce the usage of memory.



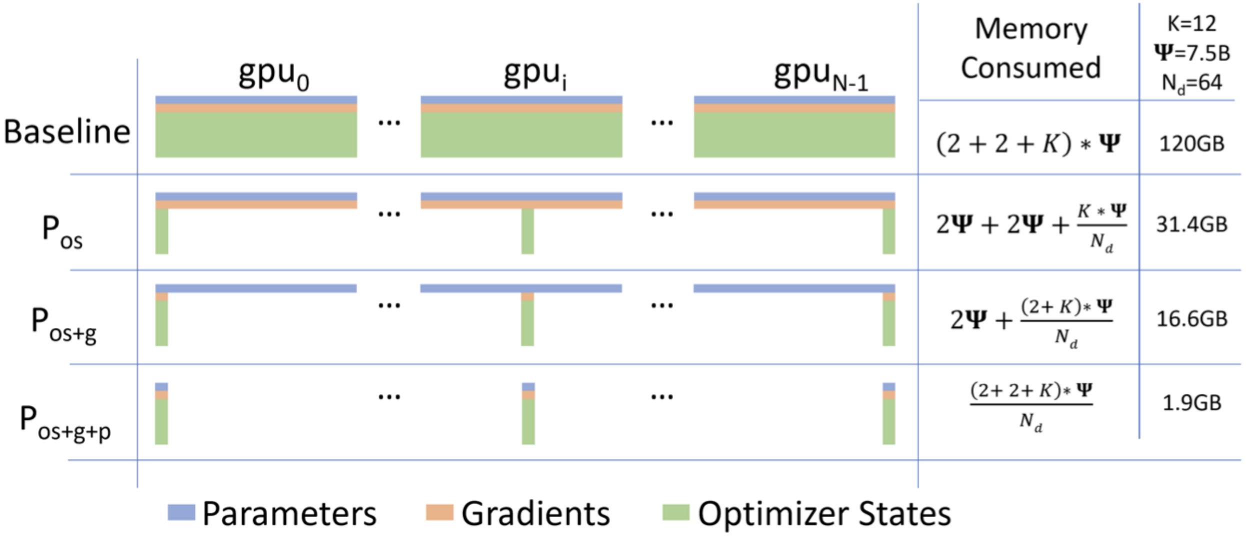
Scao, Fan, Akiki, et al. BLOOM: A 176B-parameter open access multilingual language model. ArXiv: 2211.05100, 2022.

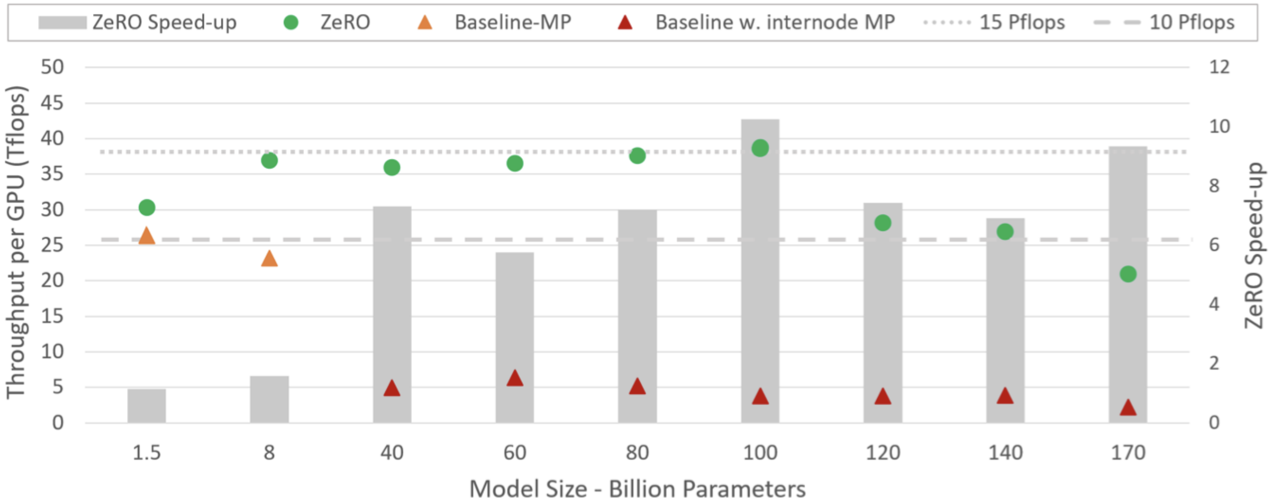
Computational Memory Optimization

- Most LLM training uses Adam Optimization Algorithm.
- Needs 1-dim Momentum and 2-dim Variance.
- Although Adam optimization algorithm is better than SGD and more stable, it increases the need for memory.
- To reduce the memory requirements, most system uses Mixed Precision Training ➔ Save FP32 and FP16 or BF16 simultaneously.
- BF16 has bigger range but fewer accuracy.
- Use some technologies to handle gradient loss and model not stable ➔ Dynamic Loss Scaling and Mixed Precision Optimizer.
- Example:
  - For a 75B parameter model, it needs 15GB computational memory using FP16.
  - But, at training, it needs 120GB for:
    - Model Sates
    - Residual States, including Activation, Buffer, and Memory Fragmentation.
    - ➔ Using ***Activation Checkpoinging*** to reduce the memory usage.

ZeRO: Zero Redundancy Data Parallelism

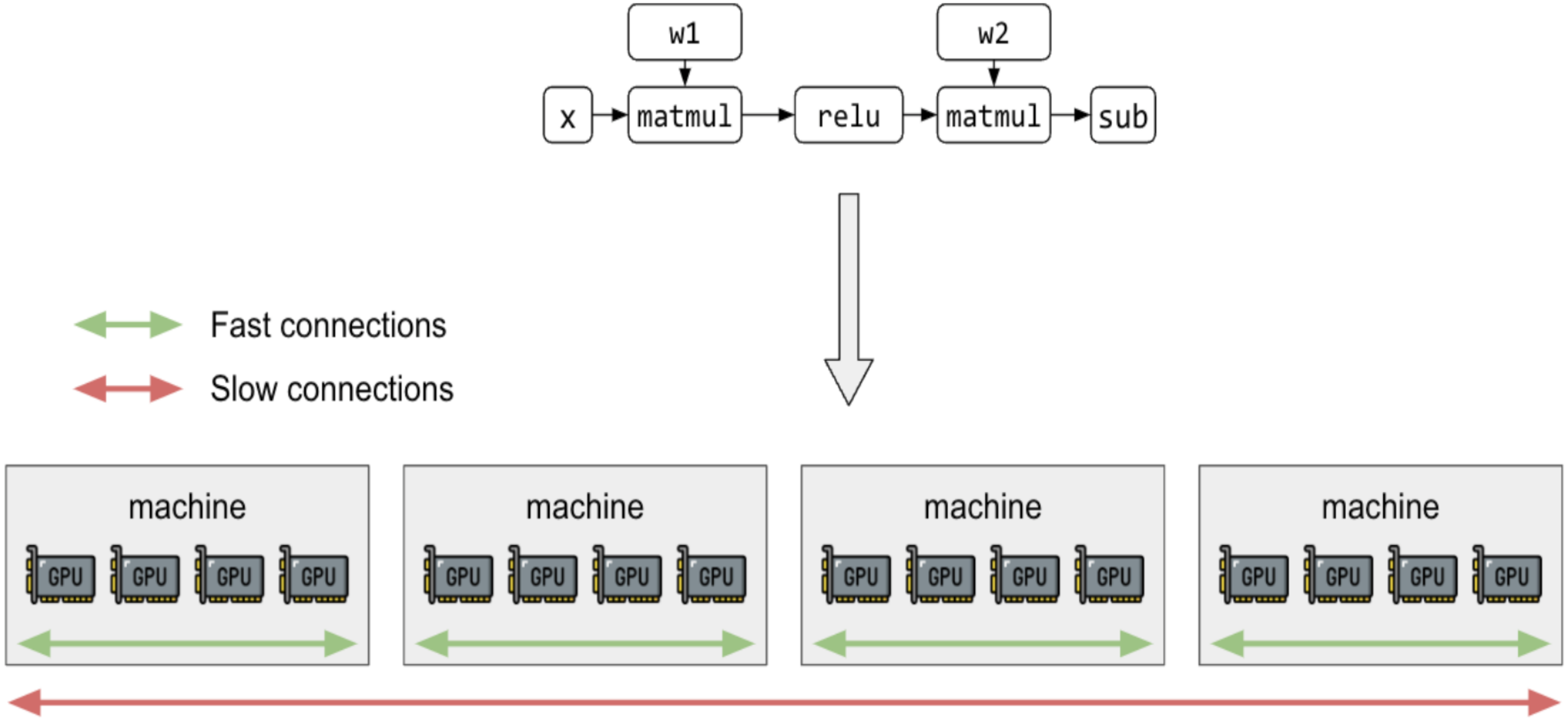
- ZeRO reduces memory needs and communication need, including these three methods:
  - Partitioning Adam Optimizer
  - Partitioning Model Gradients
  - Partitioning Model Parameters

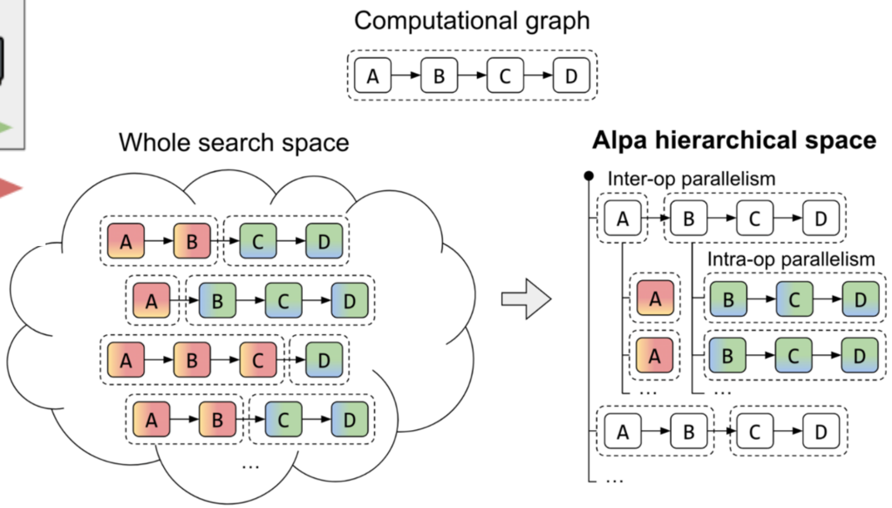




ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. Rajbhandari et. al. Prof. of Intl. Conf. for High Performance Computing, Networking, Storge and Analysis. IEEE 2020

Computing Cluster for Distributed Training

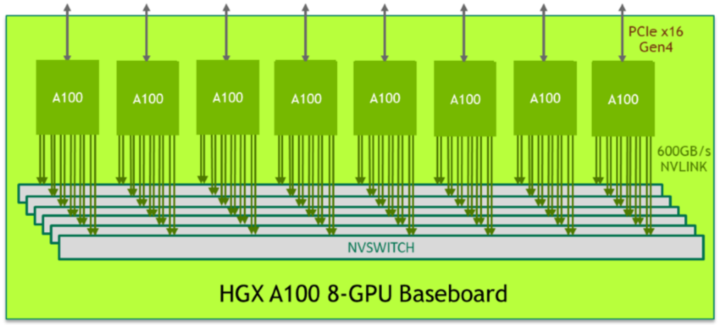




- Multiple servers in a Rack
- Racks communicated with Top of Rack Switch (ToR)
- Spine Switch can be added
- Multi-Level Tree

Communication Speed in Cluster

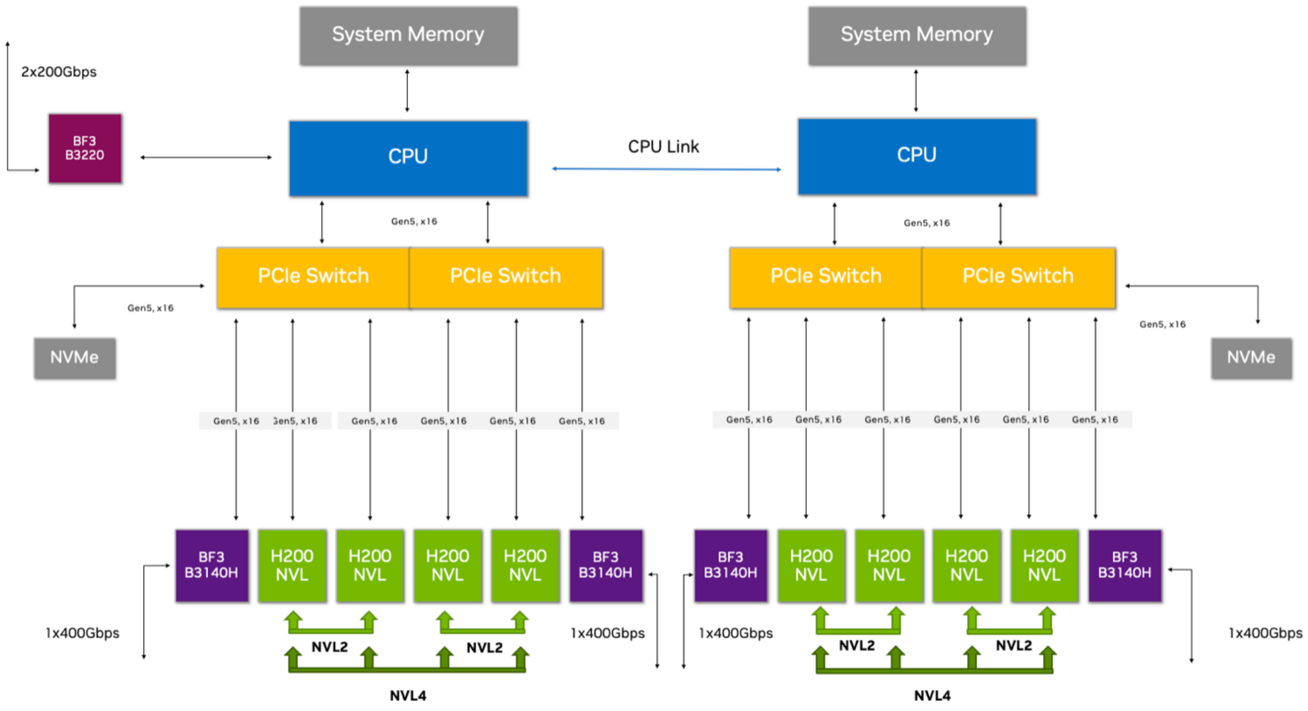
- GPT-3 as an example, each model copy has 700GB local data.
- If using 1024 GPUs having 128 Model Copies, then it needs to transmit 700GB x 128 = 89.6 TB gradient data.
- Therefore, for LLM distributed training, usually Fat-Tree Topology is used.
- InfiniBand (IB) technology is used for High Speed Network. Each IB can provide 200 Gbps or 400 Gbps bandwidth.
  - Nvidia's DGX server provides each machine of 1.6 Tbps bandwidth.
  - Nvidia's HGX server provides each machine of 3.2 Tbps bandwidth.
- Each server is usually composed of 2-16 computational units.
- If using traditional PCIe, which can only provide 128 GB / second.
- Nvidia H100 uses HBM which provides 3350 GB / second.
- Nvidia HGX H100 GPU uses NVSwitch, which has NV Port. Each NVSwitch is links 8 H100 cards. It makes any H100 card has 900 GB/s two-way speed.

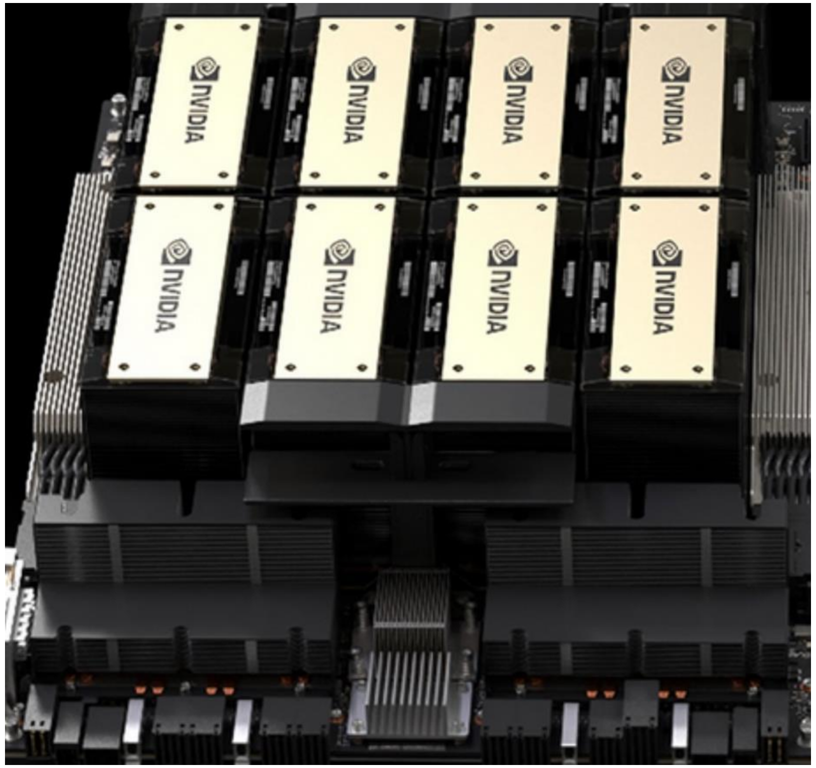


New cluster specs

Feature	NVIDIA H100 NVL	NVIDIA H200 NVL	Improvement
Memory	94 GB HBM3	141 GB HBM3e	1.5x capacity
Memory Bandwidth	3.35 TB/s	4.8 TB/s	1.4x faster
Max NVLink (BW)	2-way (600 GB/s)	4-way (1.8 TB/s)	3x faster
Max Memory Pool	188 GB	564 GB	3x larger

Table 1. Specification comparison between H100 NVL and H200 NVL





Parameter Server

- A distributed system has two types of servers: Training Server and Parameter Server
- Parameter server needs to provide enough memory and communications.
- When training, parameter server is responsible to parameter synchronization.
- Each training server sends the computed gradient values to the corresponding parameters.
- Each Parameter server can be either synchronized training or non-synchronized training.

Decentralized Network

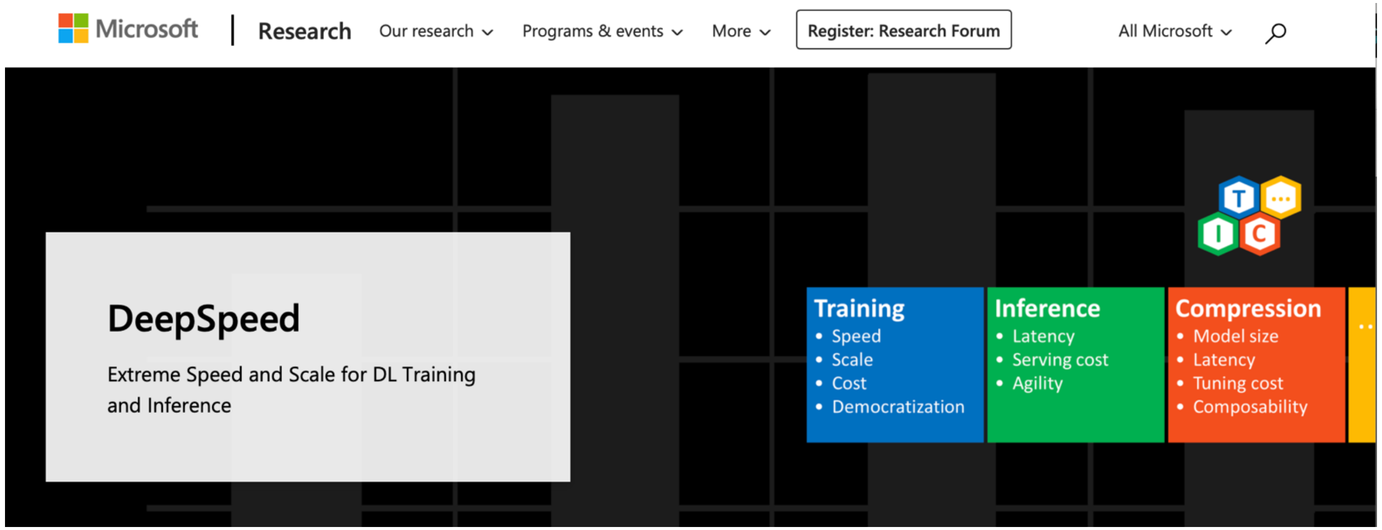
- Decentralized Network can communicate based on Collective Communication.

- Basic communications include:
  - Broadcast
  - Scatter
  - Reduce
  - AllReduce
  - Gather
  - AllGather
  - ReduceScatter
  - AlltoAll

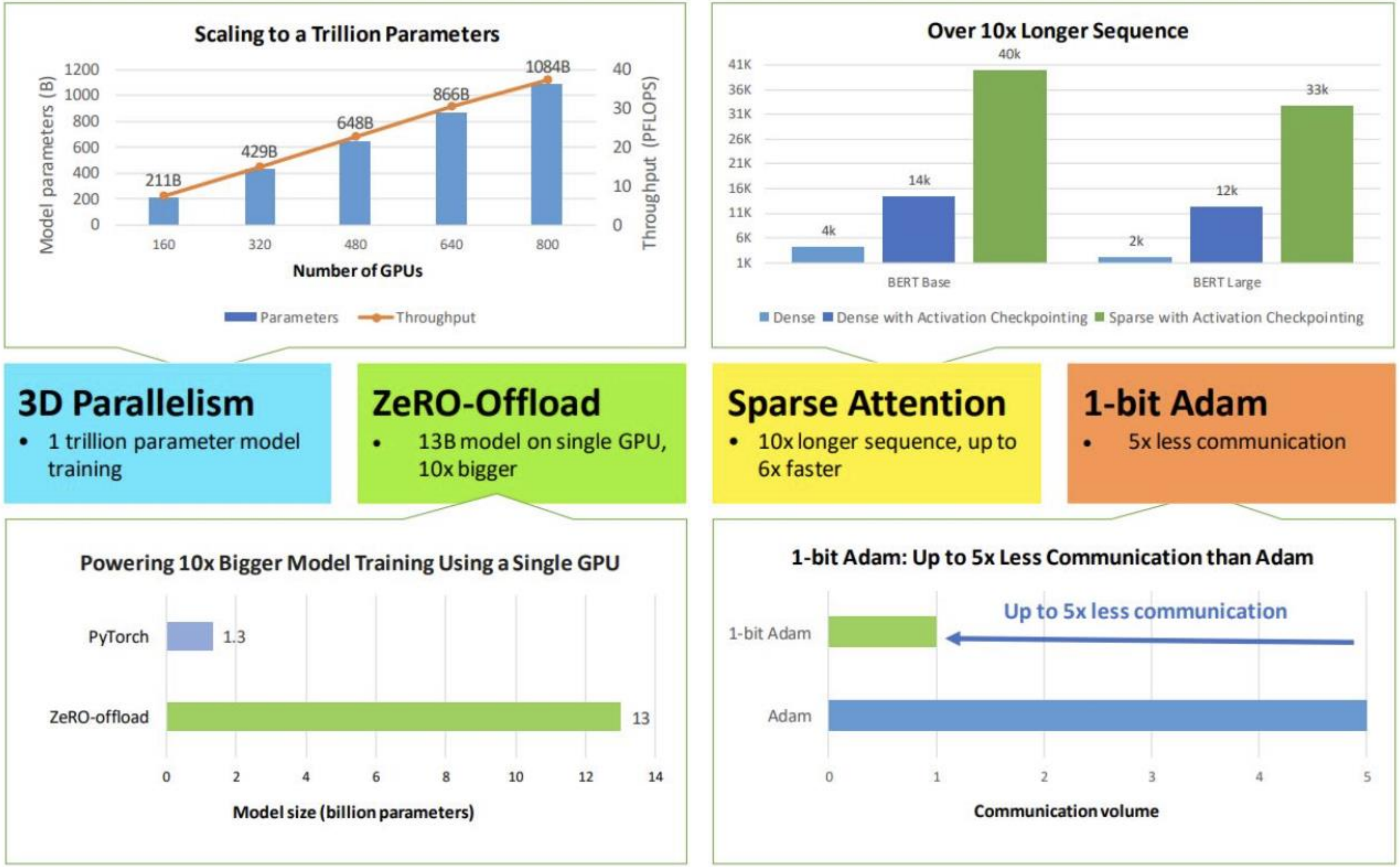
- Popular Libraries include MPI, GLOO, NCCL, etc.
  - Message Passing Interface (MPI) is usually used in multiple process communication and coordination.
  - GLOO is an MPI provided by Facebook, providing Collective Communications Library. It supports CPU and GPU distributed Learning.
  - Nvidia Collective Communication Library is a GPU communication library issued by Nvidia specifically for GPU.

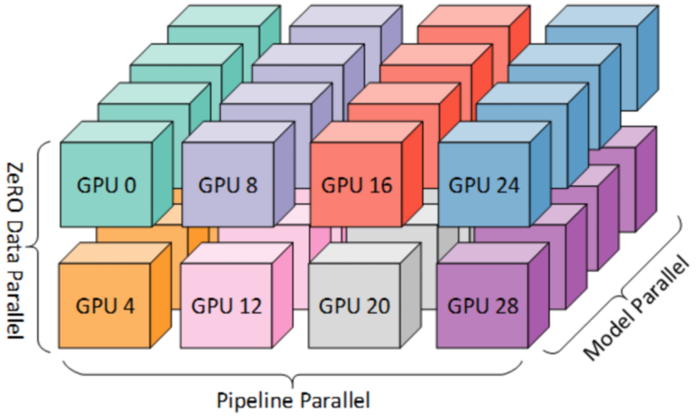
DeepSpeed

- DeepSpeed is an open-source deep learning optimization library created by Microsoft.
- It is mainly for LLM training speed and scalability.
- It helps researchers being able to quickly explore iteration and new models and algorithms.
- It includes many speedup algorithms.
- It also includes many management tools, such as distributed training management, memory optimization, and model compression.



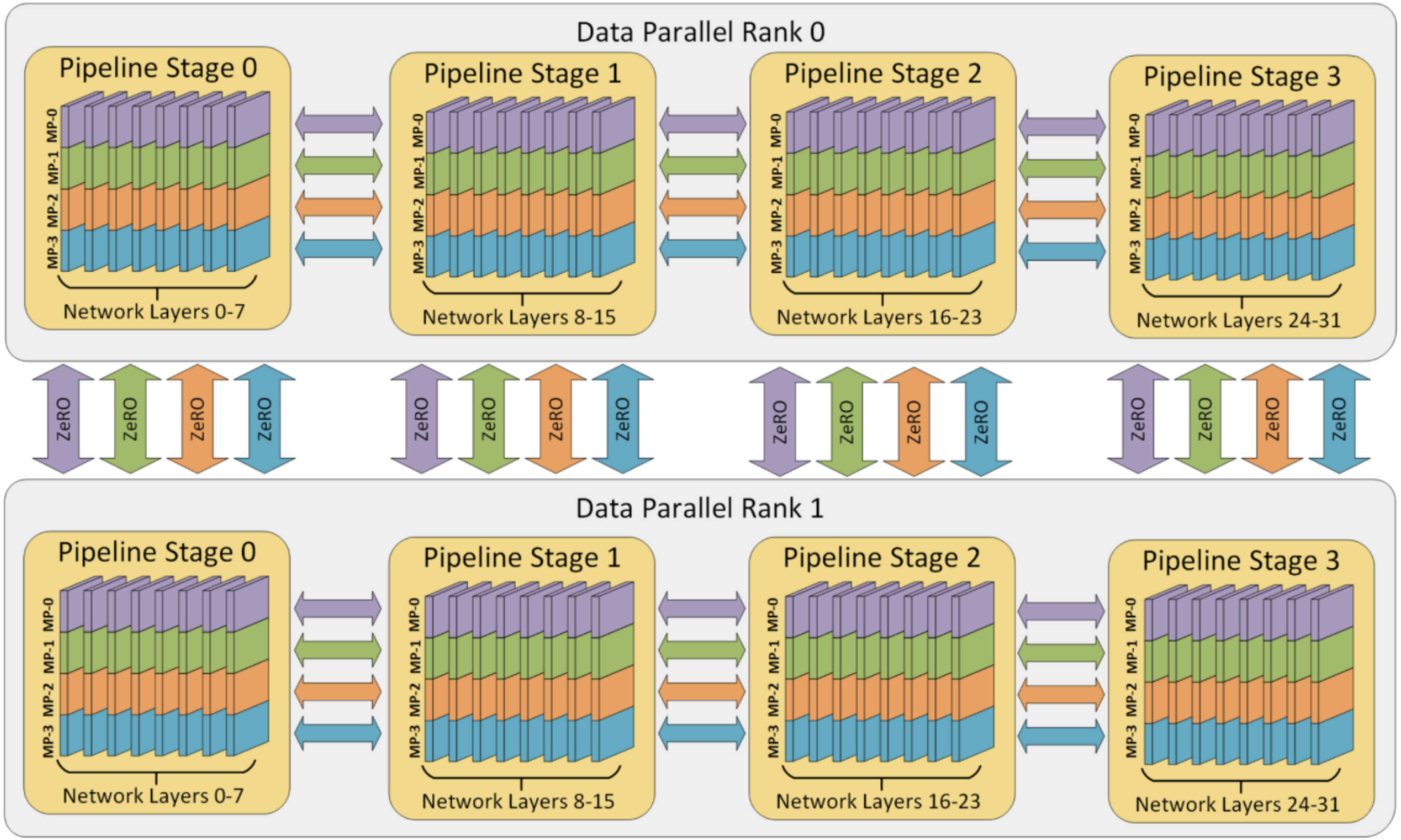
DeepSpeed mechanisms





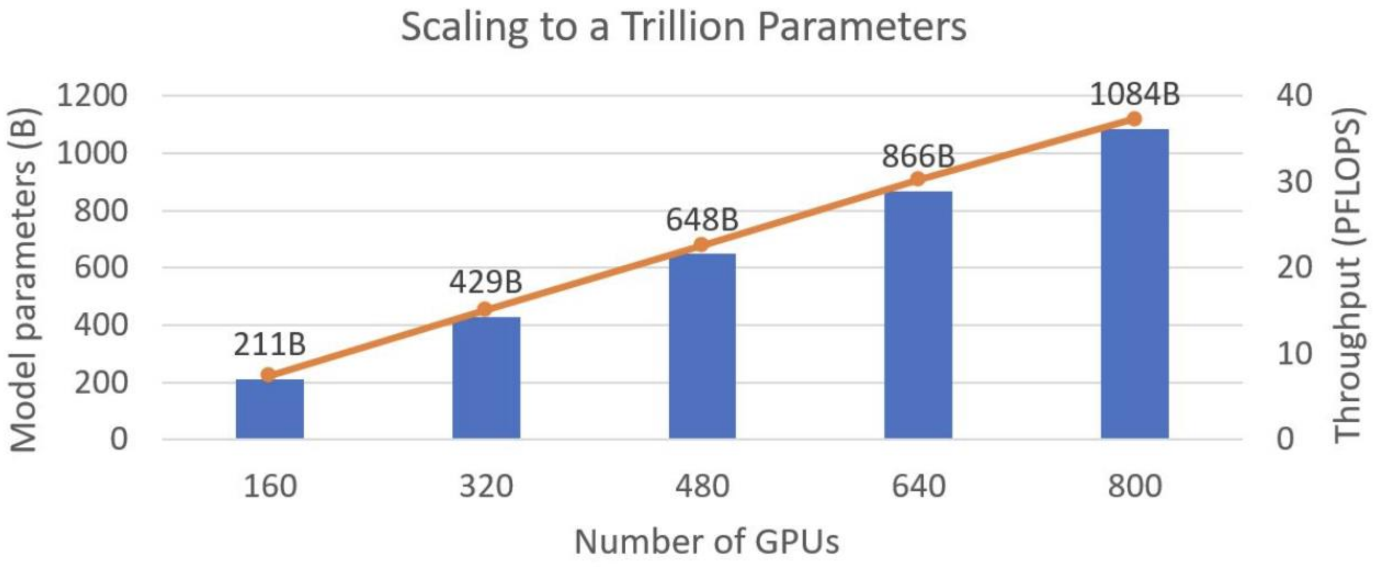
https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

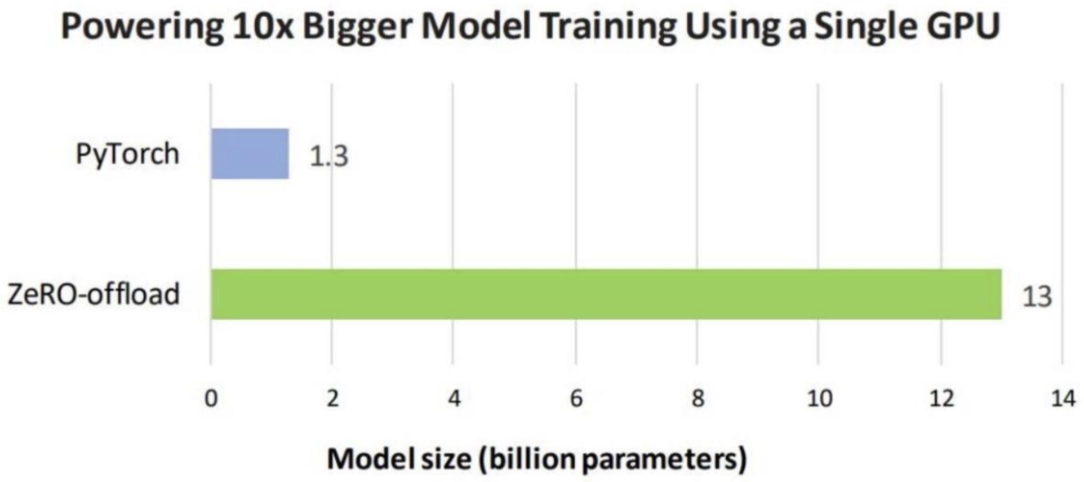
DeepSpeed 3D Parallelism Strategy



https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

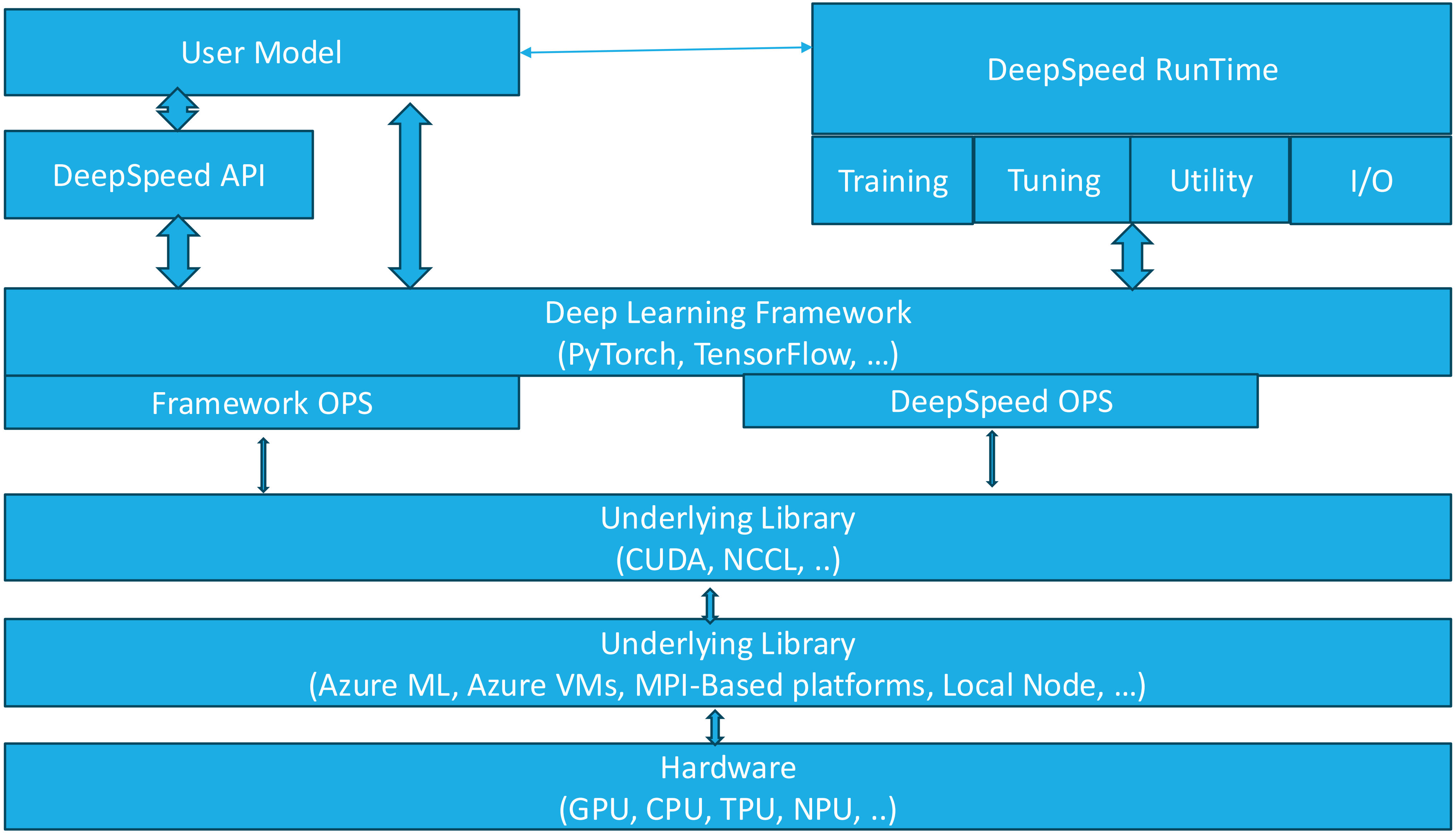
DeepSpeed Scaling





https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

DeepSpeed Software Architecture



Llama Distributed Training - I

- Using DeepSpeed to train Llama model.

- Step 1: Training data setting:
  - DataLoader
  - RandomSampler and SequentialSampler are samplers from PyTorch
  - DistributedSample as data sampler for distributed training
  - Default_date_collator: data collector for transformers
  - Create_pretrain_dataset: for setting pre-train dataset.

- Step 2: Model loading:
  - Using transformers library to load and set Llama model and related Tokenizer
  - Use From-Pretrained to load pretrained Llama model, tokenizer, and model setting.
  - Padding may be used if necessary.

- Step 3: Set up optimizer:
  - Using DeepSpeedCPUAdam and FusedAdam to speedup.
  - Use get_optimizer_grouped_parameters
  - Choose best optimizers
  - Scheduling of the learning rate

Llama Distributed Training - II

- Step 4: DeepSpeed Set Up:
  - Set up Global_Batch_Size and Micro_Batch_size
  - Set up get_train_ds_config:
    - ZeRO optimization setting
    - Hybrid precision training (e.g., FP16)
    - Gradient Clipping
    - Hybrid Engine setting
    - TensorBoard setting
    - Get Evaluation DS Config

- Step 5: DeepSpeed Initialization:
  - Check local GPU (using CUDA)
  - DeepSped Init Distributed() for each process's synchronization.
  - Get Torch.Distributed>get_rank()
  - Based on parameters (e.g., offload, Zero Stage, etc) to set up a DeepSpeed Dictionary
  - Sync all procedures using torch.distributed.barrier()
  - Use Deepspeed.initialize to initiate
  - Use Gradient checkpointing to find ways to save memory.

Llama Distributed Training - III

- Step 6: Model training
  - Preparation before training.
    - Use print_rank_0 to print out the training states. Make sure all proceses print info.
  - Training loop:
    - In each iteration, it prints current loop and all loops.
    - Data batch is moved to related GPU
    - Execute model
  - Storing Model:
    - Models can be saved in different format:
      - HuggingFace's model format
      - DeepSpeed's Zero Stage 3 format

Summary of Speeding Up

- DeepSpeed, Megatron-LM, Colossal-AI’s training models can be used for LLM model training.
- Most open source LLM models are developed based on HuggingFace transformers.
- If < 30B parameters, it’s possible to not using Tensor Parallelism.
- It’s important for hyper parameters – batch size, learning rate, optimizer, etc.
- Important for the stability of models.
  - Llama-2 uses batch size of 4M tokens.
  - GPT-3 uses batch size of 32K to 3.2M tokens.
- Many current LLMs use Warm-up and Decay Learning Rate. Gradually increase Learning rate to the maximum number.
- LLMs training usually uses Adam or AdamW optimizers.