

DSCI 565: COMPUTATIONAL PERFORMANCE

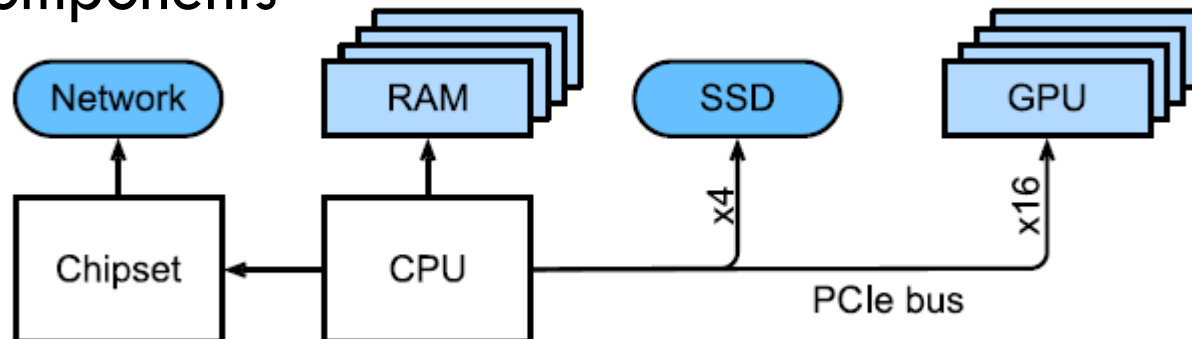
Ke-Thia Yao

Lecture 18: 2025 November 5

Hardware

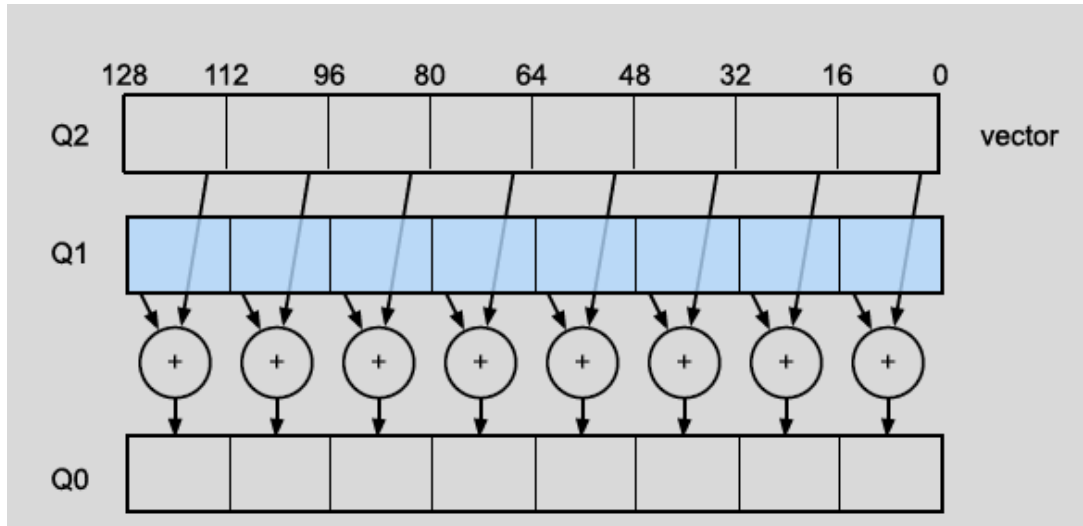
2

- The hardware components and the connectivity between them can have a large effect on the computational performance
- Section 13.4 of Zhang *et al.* has lots of interesting details
- Hardware include:
 - ▣ Components: Processor (CPU, GPU), memory (RAM), storage (SSD, hard drive), network
 - ▣ Connectivity between components



Vectorization

3



- Vector unit enable CPU to perform many operation in one clock cycle
- Perform SIMD (Single Instruction Multiple Data) operations
- Example: vector unit performing 8 additions in one cycle
- Other types can perform multiply-add operations

Connectivity

4

- Connectivity is typically measured by
 - ▣ Bandwidth, the amount of data transferred per second
 - ▣ Latency, the time delay before the transfer starts

CPU-Memory Connectivity

5

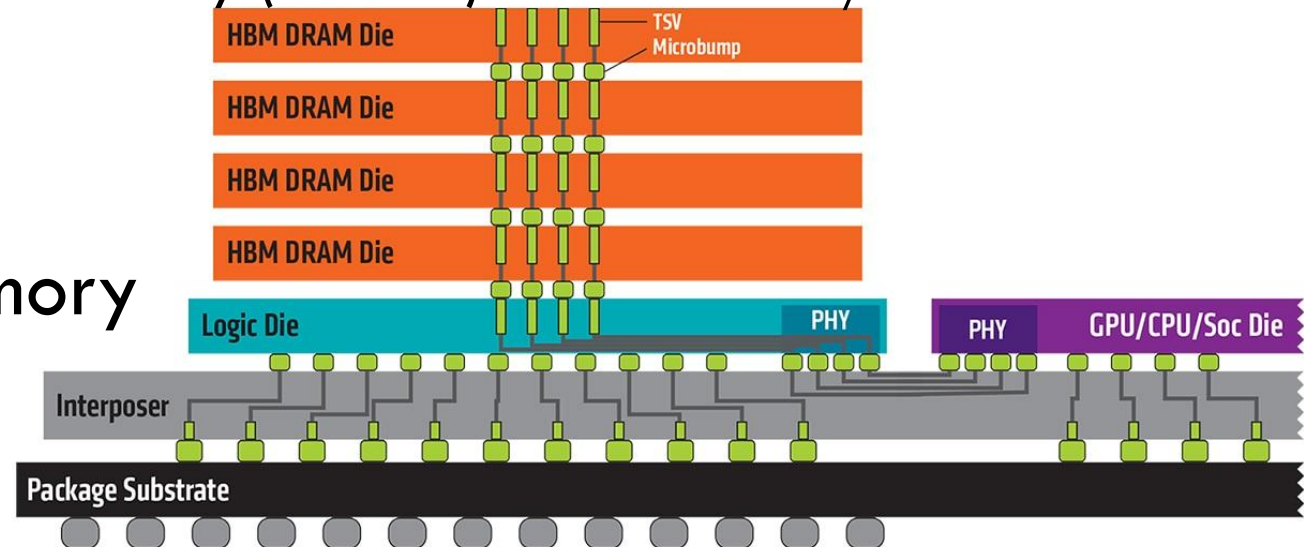
- A typical DDR4 RAM offers 20-25 GB/s bandwidth per module, where each module has a 64-bit-wide bus
- A CPU have between 2 and 4 memory channels, i.e., peak memory bandwidth between 40GB/s to 100GB/s
- Read a 64-bit record takes just 0.2ns at 40GB/s
- But setting up the transfer (send address to RAM then wait for start of transfer) is 100ns (500 times!)

GPU-Memory Connectivity

6

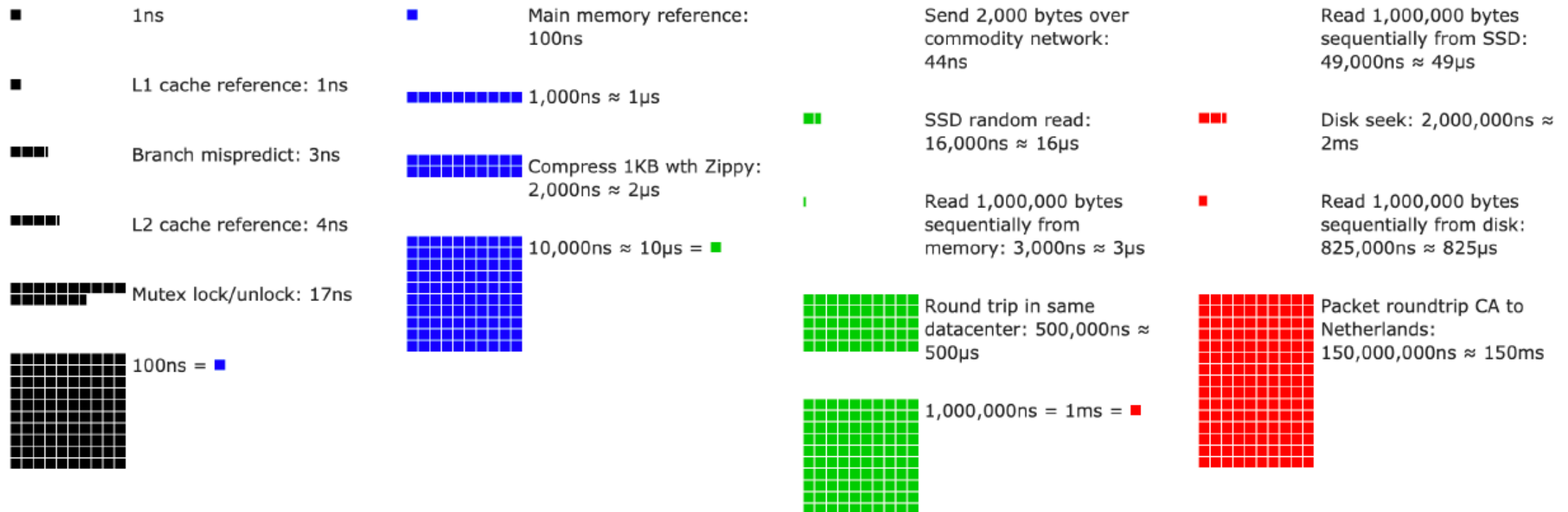
- GPUs require much higher bandwidth
- Use wider bus, e.g., NVIDIA B100 has 2X 4096-bit-wide bus
- Use higher speed memory
 - ▣ NVIDIA V100 using High Bandwidth Memory (HBM2) has 900 GB/s
 - ▣ NVIDIA H100 using High Bandwidth Memory (HBM3) has 3300 GB/s
 - ▣ NVIDIA B200 using High Bandwidth Memory (HBM3e) has 7700 GB/s

High Bandwidth Memory



Latency

7



GPU

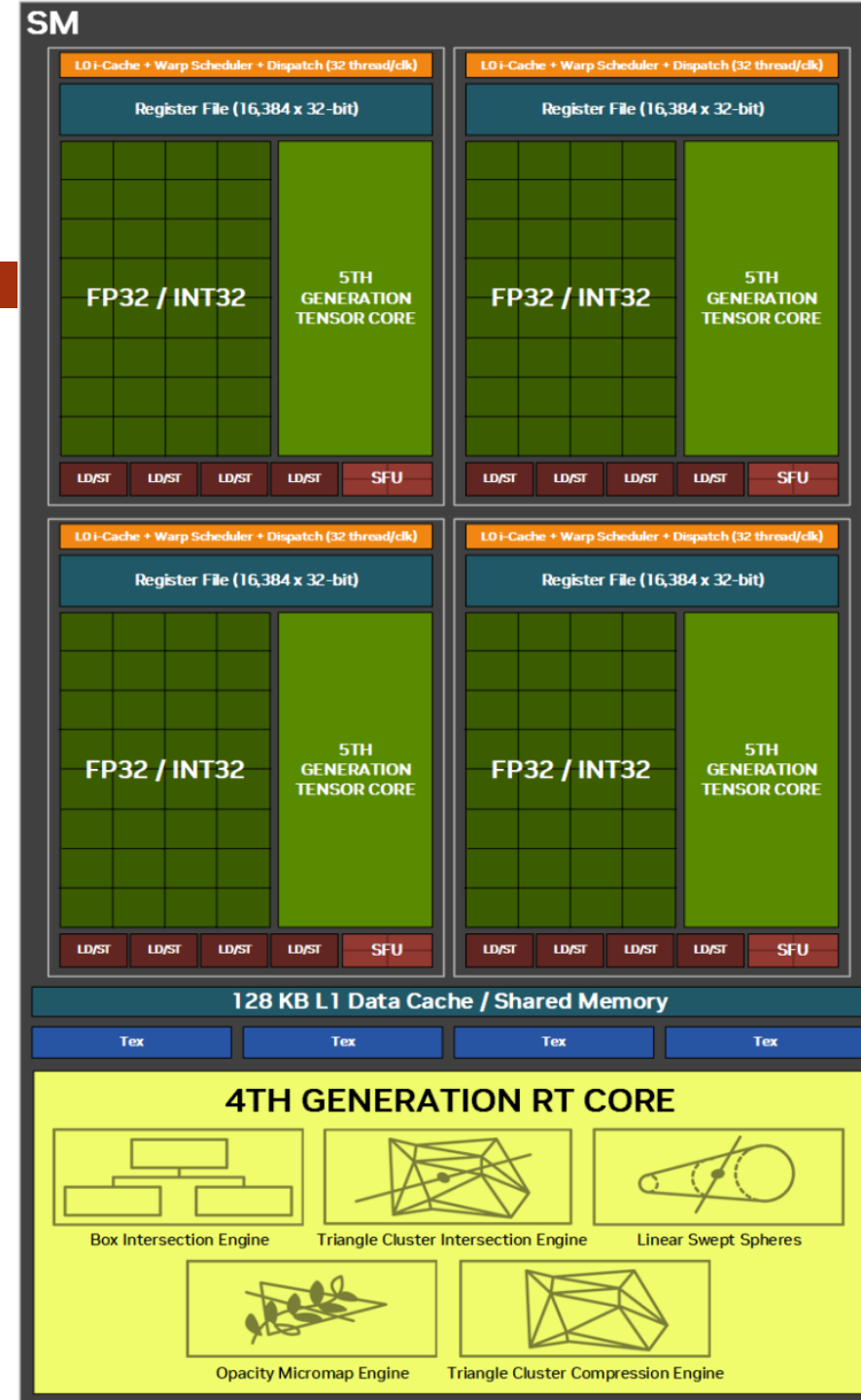
8

- Deep learning would not have been successful without the GPUs
- GPU design strategy
 - ▣ Many more core
 - ▣ Support matrix operations, not just vector operations (tensor cores)

Blackwell GB202 GPU Architecture

9

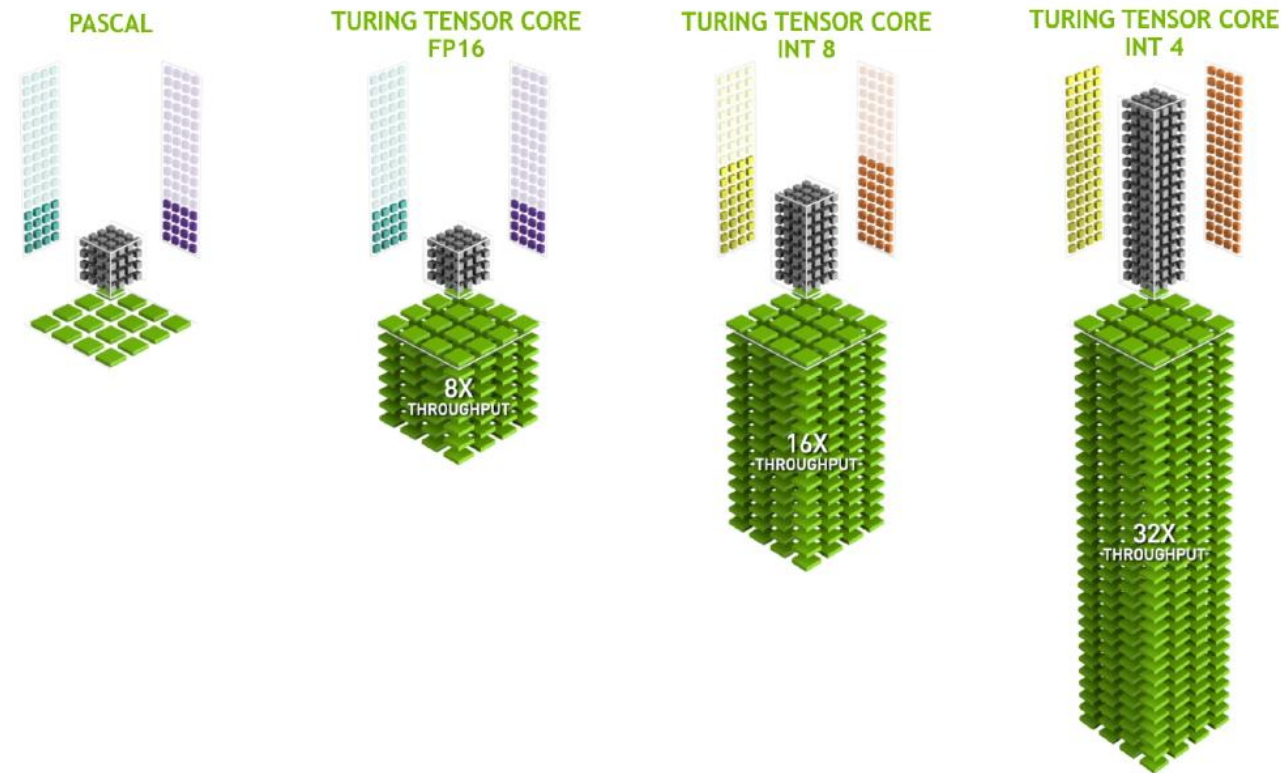
- Basic Processing Block
 - ▣ 32 FP32/INT32 CUDA cores
 - ▣ 1 5th generation tensor core
 - ▣ 64K register file (16K x 32-bit)
- Streaming multiprocessor
 - ▣ 4 processing blocks
 - ▣ 128KB L1 Cache
 - ▣ 1 Ray Tracing core
 - ▣ Single instruction multiple threads (SIMT)



Tensor Core

10

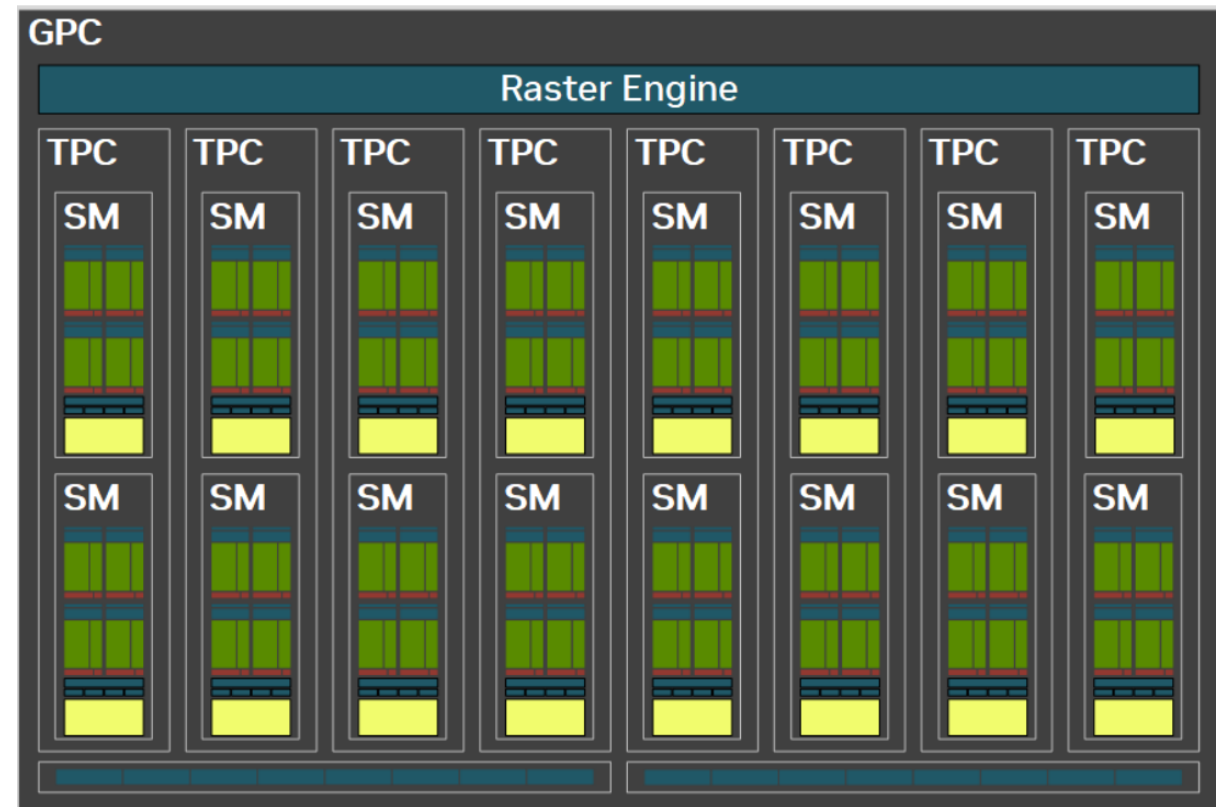
- Performs matrix multiplication and accumulate
- Mixed precision calculation: multiple in FP16 and accumulate in FP32
- Optimized for 4×4 and 16×16 matrix operations



Blackwell GB202 GPU Architecture

11

- Graphics Processing Clusters (GPC) contains
 - ▣ 16 Streaming multiprocessors (SMs)
 - ▣ 8 Texture processing cores (TPCs)
 - ▣ 1 Raster engine



Blackwell GB202 GPU Architecture

12

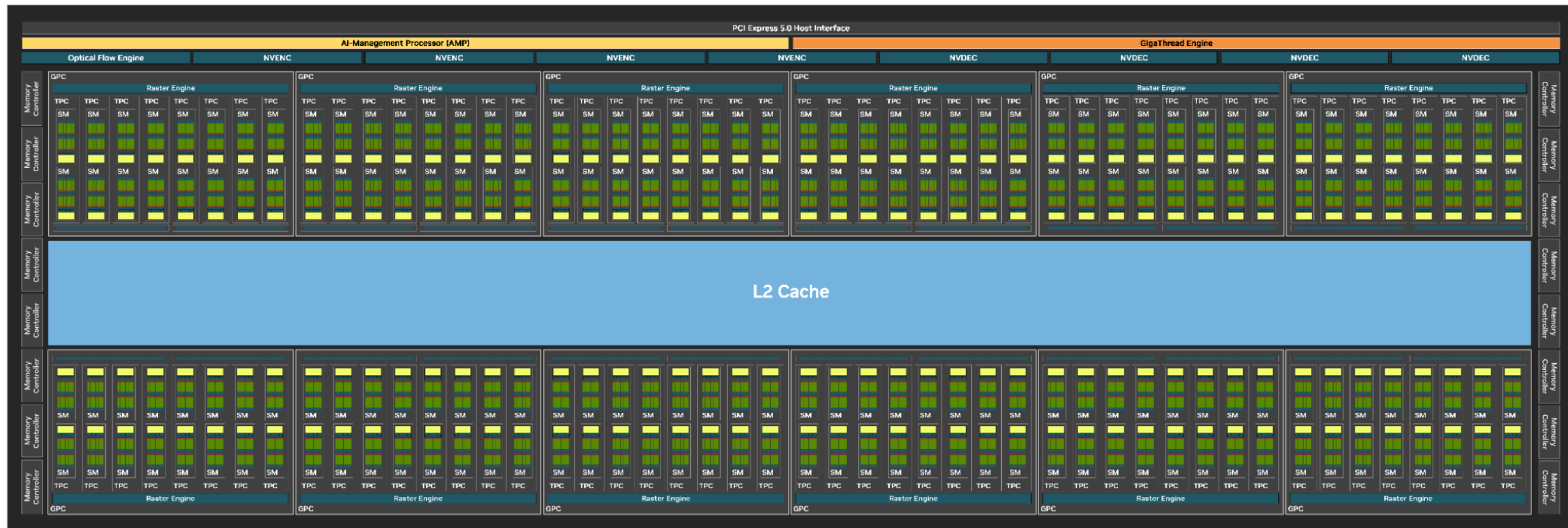
□ 12 GPCs

□ 128MB L2 Cache



□ 24576 CUDA Cores

□ 768 Tensor Cores



Blackwell GB202 GPU Performance

13

- Peak FP32 (non-Tensor): 126 TFLOPs
- Peak FP16 Tensor with FP32 Accumulate: 503.8 TFLOPs
- Peak FP8 Tensor with FP32 Accumulate: 1007.6 TFLOPs
- Peak FP4 Tensor with FP32 Accumulate: 2015.2 TFLOPs

GPU Data Center

14



Grace Blackwell Ultra "Superchip"
1 Grace CPU + 2 Blackwell GPUs



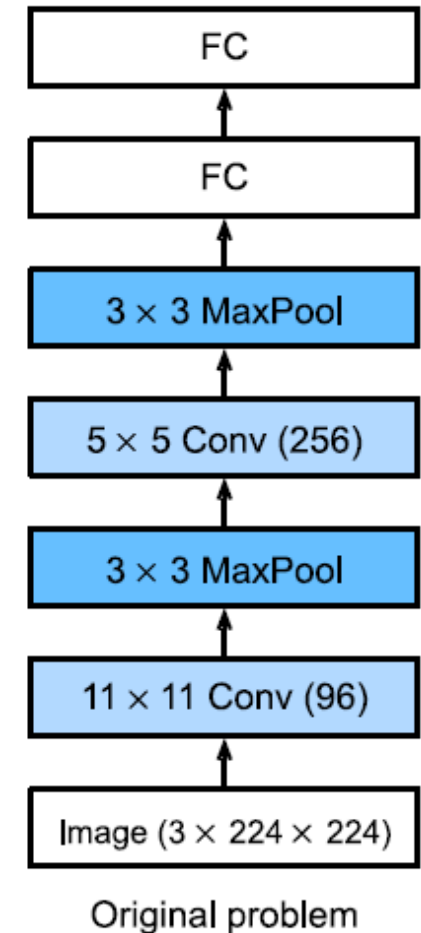
Rows and rows of racks in a data center

1 Rack = 36 Grace CPUs
+ 72 Blackwell GPUs

Training on Multiple GPUs

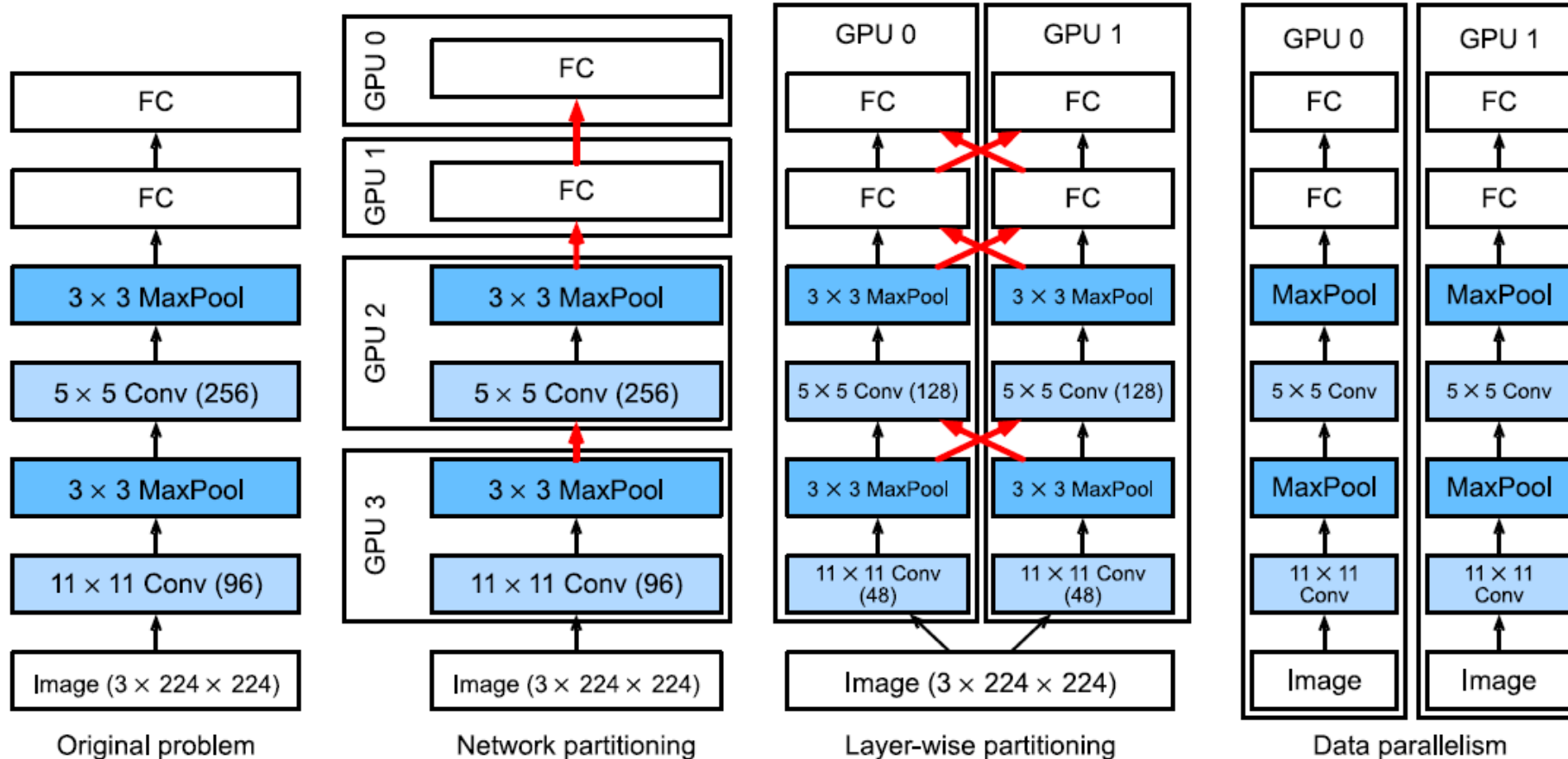
15

- How to split up the problem to train on multiple GPUs
- Pipeline parallelism / network partitioning
 - ▣ Partition the network by layers, i.e., place one or more layers in a GPU
- Tensor parallelism / layer-wise partitioning
 - ▣ Tensor in each layer is partitioned among multiple GPUs
- Data parallelism
 - ▣ Partition the minibatch. Each GPU holds a complete copy of the network



Parallelization on multiple GPUs

16



Pipeline Parallelism / Network Partitioning

17

□ Advantages

- ▣ Memory footprint per GPU is smaller (only a few layers of the network)

□ Disadvantages

- ▣ Balance workload across GPUs can be tricky. Some layers are more computationally intensive than other layers
- ▣ Require large amount of data transfer between GPUs
- ▣ Difficult to achieve linear scaling, i.e., compute time decreases linearly with number of GPUs

Tensor Parallelism / Layer-wise Partitioning

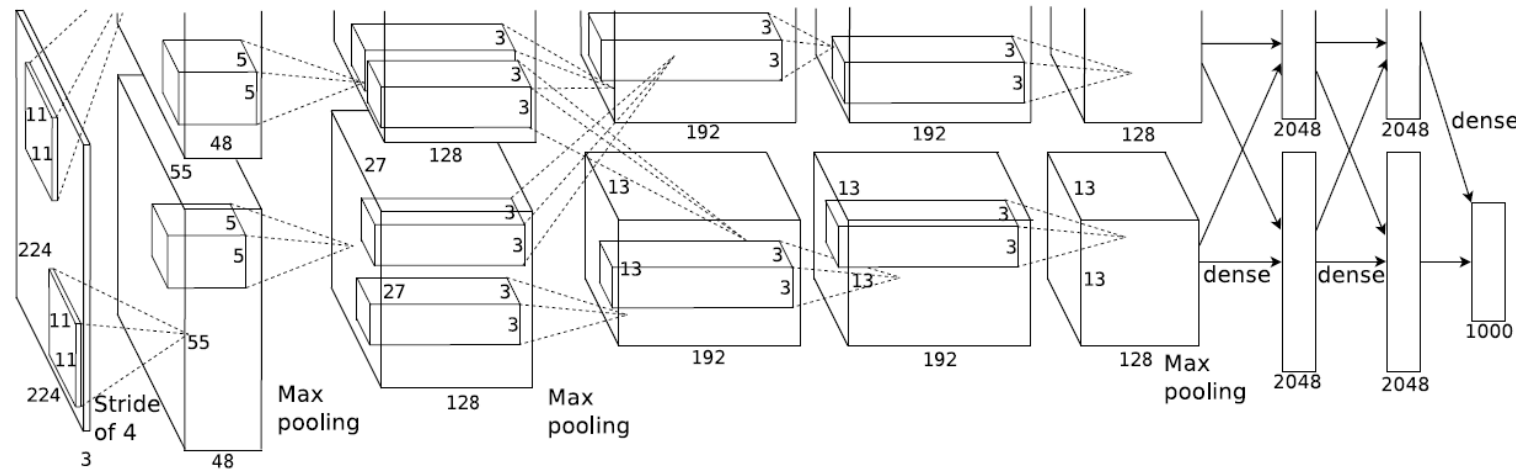
18

□ Advantages

- ▣ Memory footprint per GPU is smaller, e.g., with 4 GPUs and a layer with 64 channels each GPU gets 16 channels

□ Disadvantages

- ▣ Very large number of synchronization/barrier operations
- ▣ Require large amount of data transfer between GPUs



AlexNet

Data Parallelism

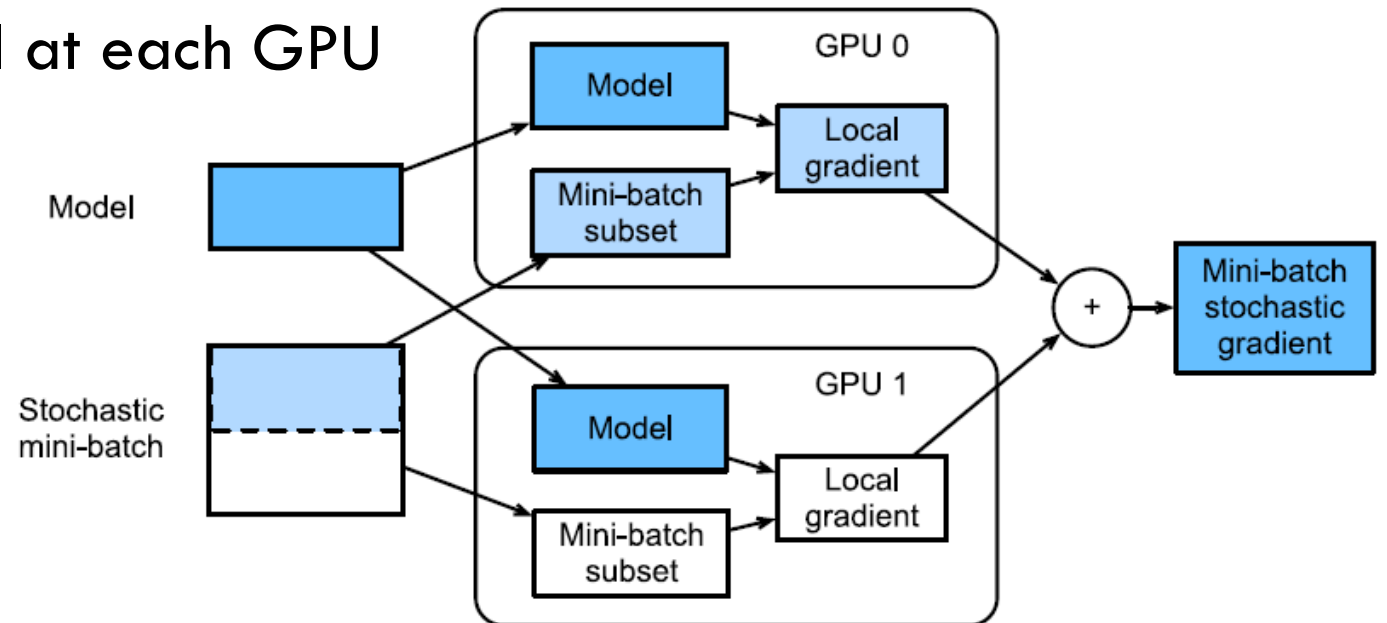
19

□ Advantages

- ▣ Easy to implement
- ▣ Near linear scaling

□ Disadvantages

- ▣ Need to store entire model at each GPU



Data Parallelism Notebook

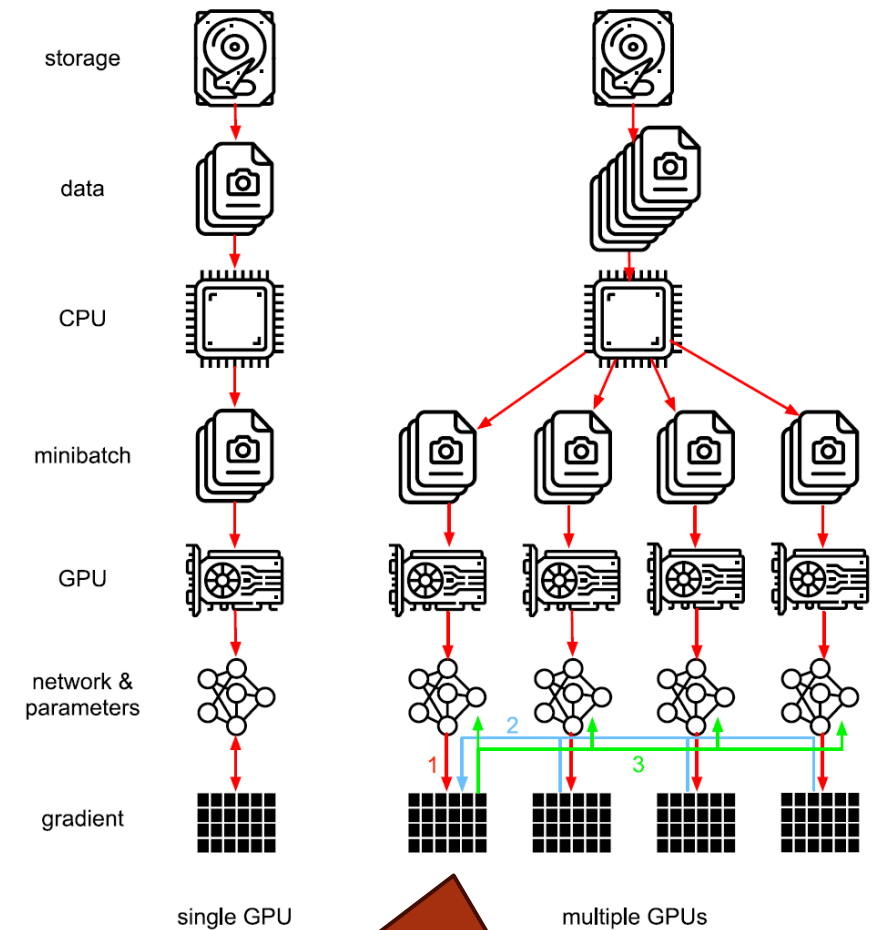
20

- `chapter_computational-performance/multiple-gpus.ipynb`
 - ▣ Toy network
 - ▣ Data synchronization: `get_param()`, `allreduce()`
 - ▣ Distributing data
 - ▣ Training
- `chapter_computational-performance/multiple-gpus-concise.ipynb`
 - ▣ `net = nn.DataParallel(net, device_ids=devices)`

Parameter Server

21

- A parameter server stores, collects and distributes parameters/values needed by the deep learning model
- For example, for the data parallel approach a parameter server would
 - ▣ Aggregate gradients from all GPUs
 - ▣ Update the weight parameters with the gradients
 - ▣ Broadcast the updated parameters to all GPUs

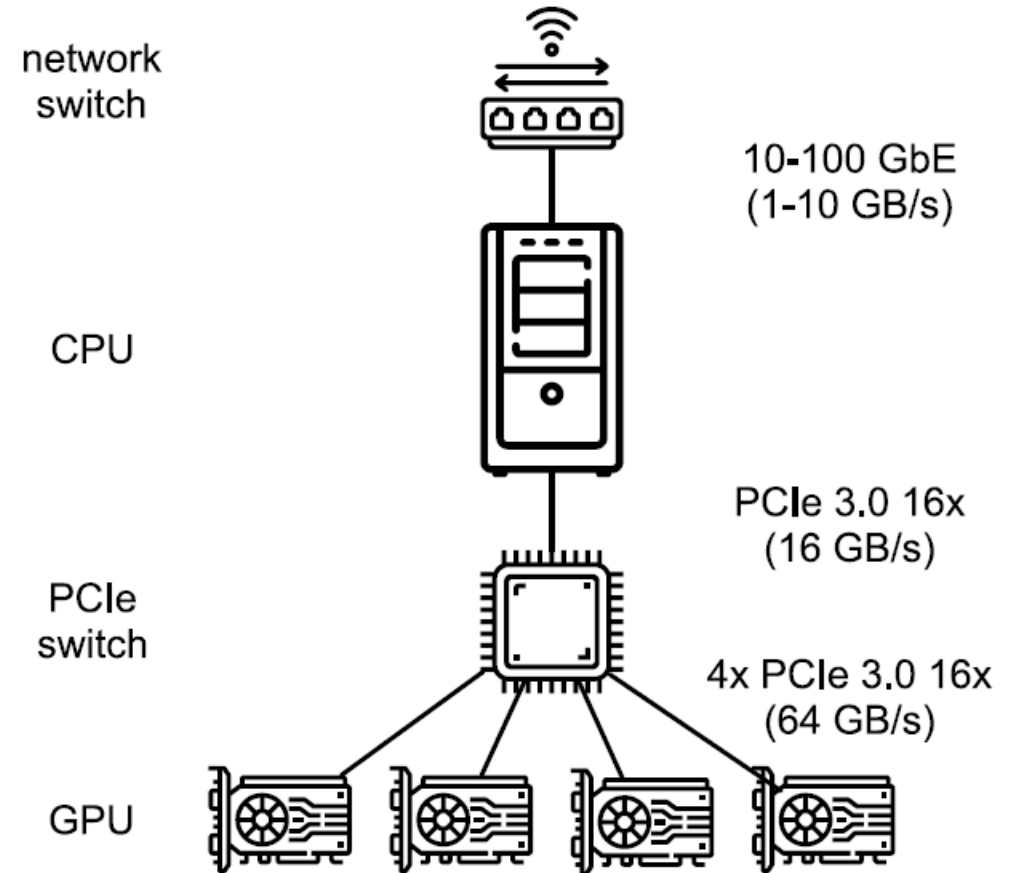


Parameter Server Running a GPU 1

Location of the Parameter Server

22

- Bandwidth is an important consideration in determining which device should host the parameter server
 - ▣ 1-10 GB/s for off-machine server connected through network switch
 - ▣ 16 GB/s for on-machine CPU server connected through PCIe **bus**
 - ▣ 64 GB/s ($4 * 16 \text{ GB/s}$) for on-machine GPU server connected through 4x PCIe **switch**



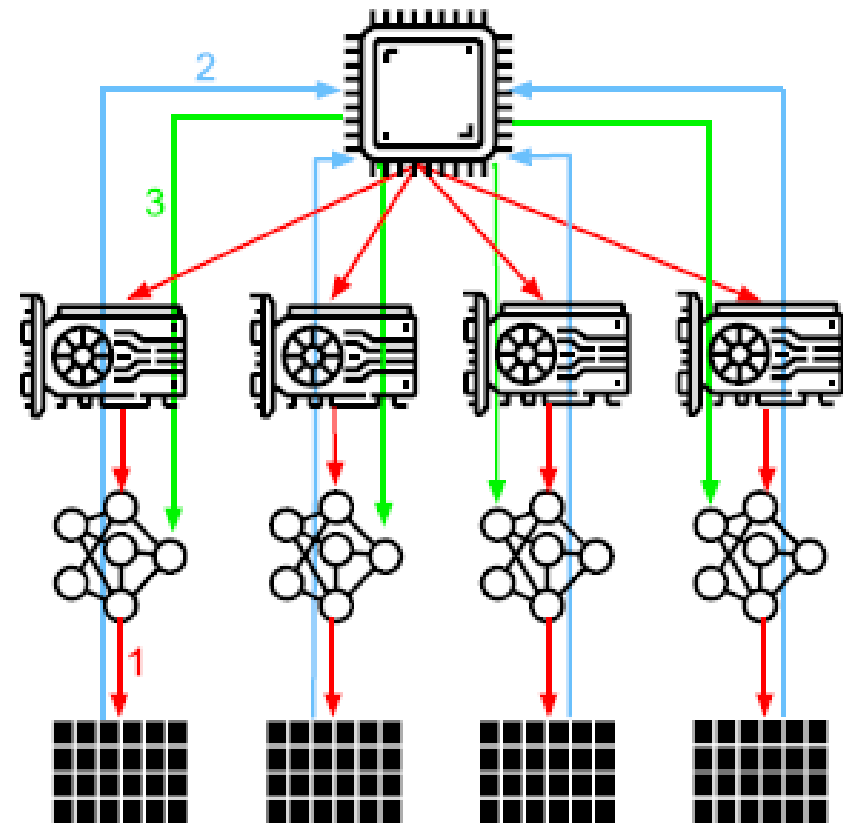
Parameter Server Example

23

Suppose parameter server has to send 160MB of gradients to GPUs

CPU Server

- 40 ms ($4 \times 160\text{MB}/16\text{GB/s}$) to send gradients to CPU Server
- 40 ms to send updates parameters to all 4 GPUs
- 80 ms total



Parameter server in CPU

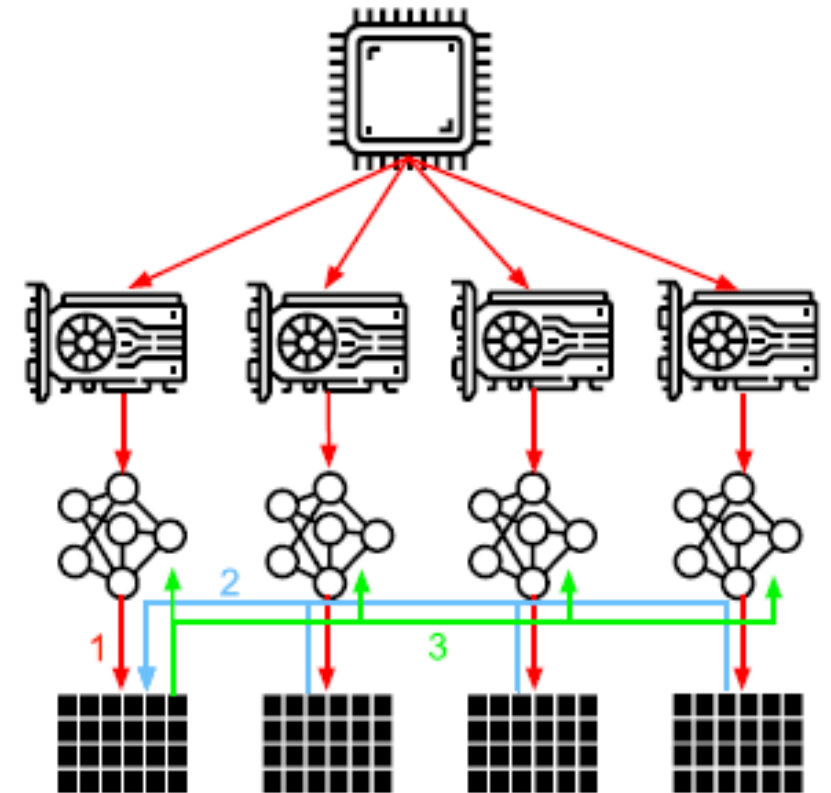
Parameter Server Example

24

Suppose parameter server has to send 160MB of gradients to GPUs

GPU Server

- ❑ Server GPU already has gradients
- ❑ 30 ms ($3 \times 160\text{MB} / 16\text{GB/s} = 3 \times 10\text{ms}$) to send gradients to GPU Server
- ❑ 30 ms to send updated parameters to other 3 GPUs
- ❑ 60 ms total

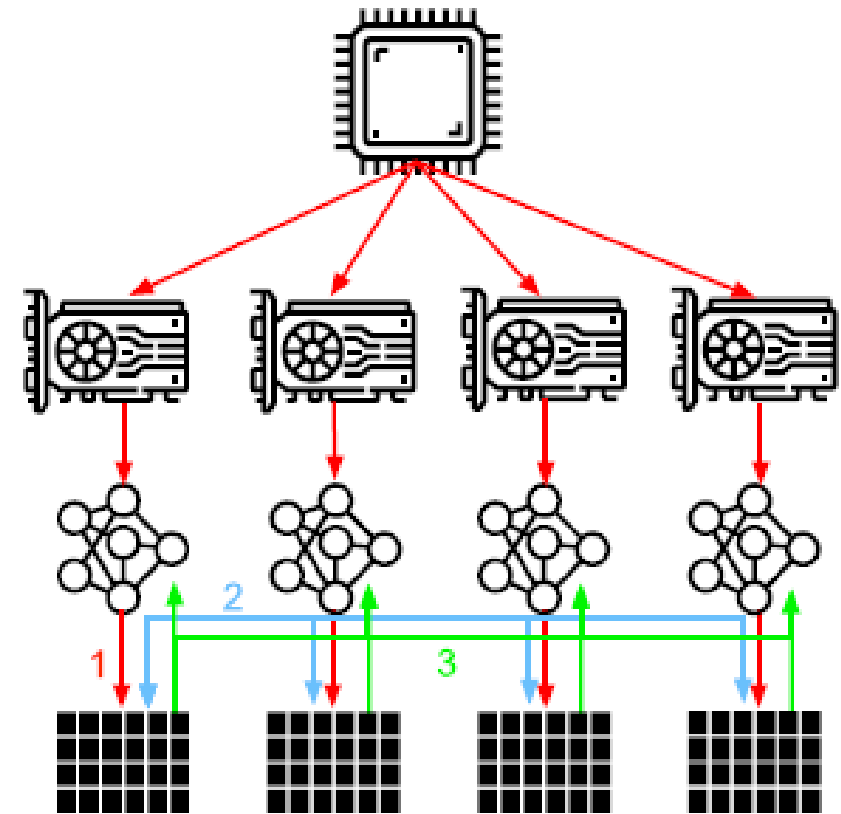


Parameter Server Example

25

Distributed GPU

- Suppose each of four GPUs holds $\frac{1}{4}$ (40MB) of the data
- 7.5 ms ($3 \times 40\text{MB} / 16 \text{ GB/s} = 3 \times 2.5\text{ms}$) send gradients
 - ▣ Each GPU sends its portion of the gradient to other GPUs
 - ▣ This can be done simultaneously, since GPUs are connected by a switch
- 7.5 ms to send updated parameters
- 15 ms total

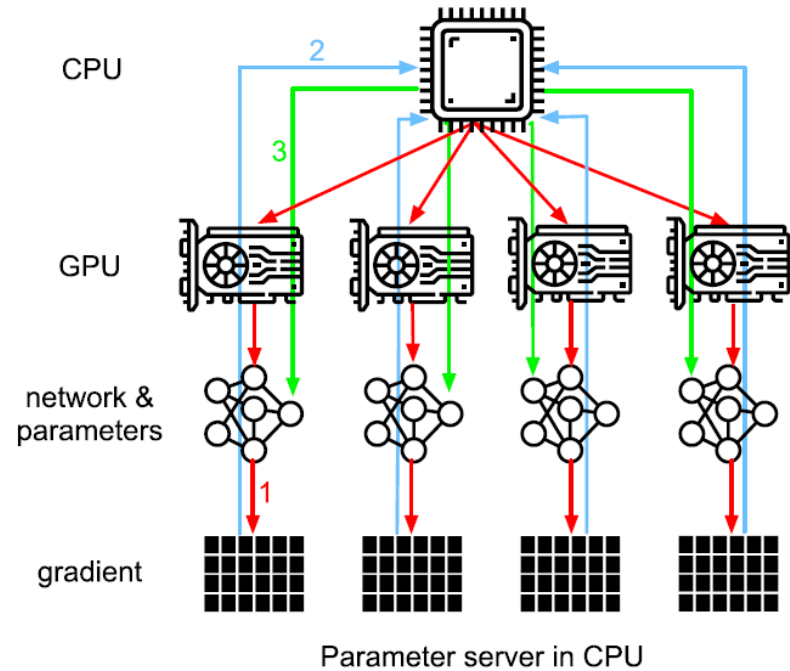


Parameter server distributed over all GPUs

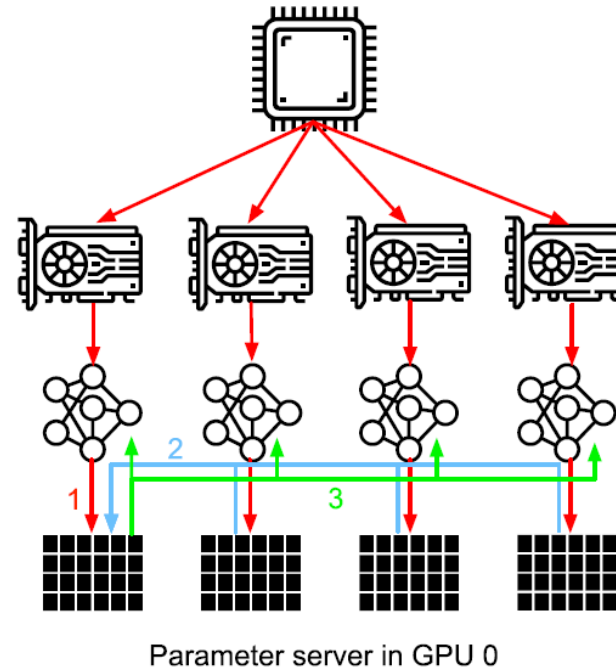
Parameter Server Example

26

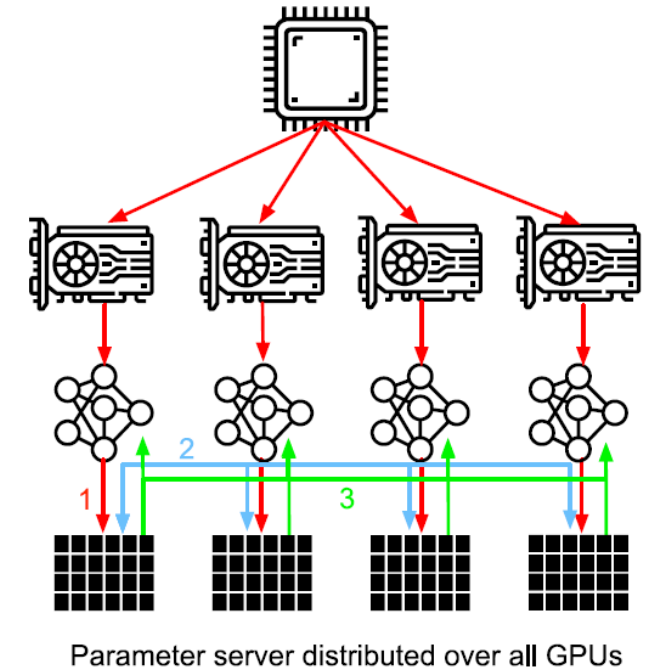
CPU: 80ms



GPU: 60ms



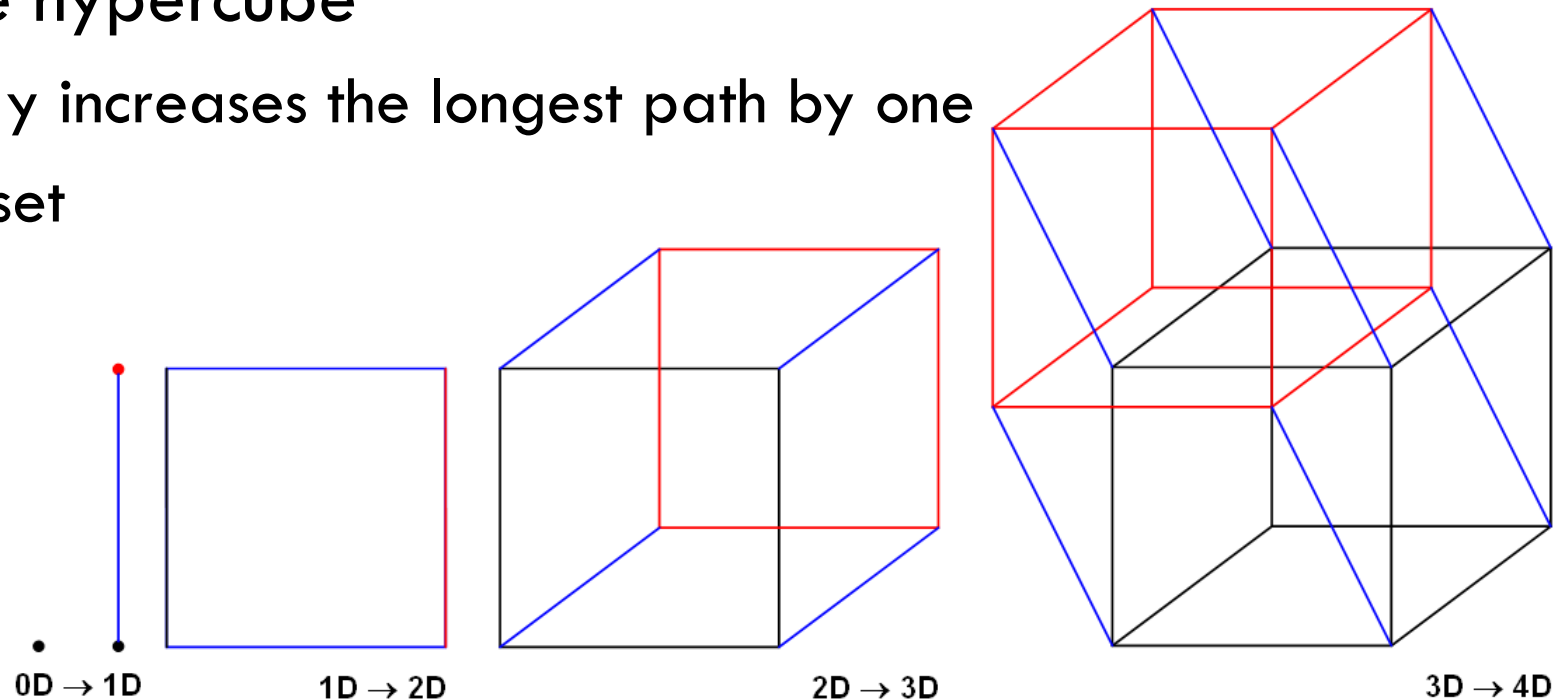
Distributed GPU: 15ms



Hypercube Topology

27

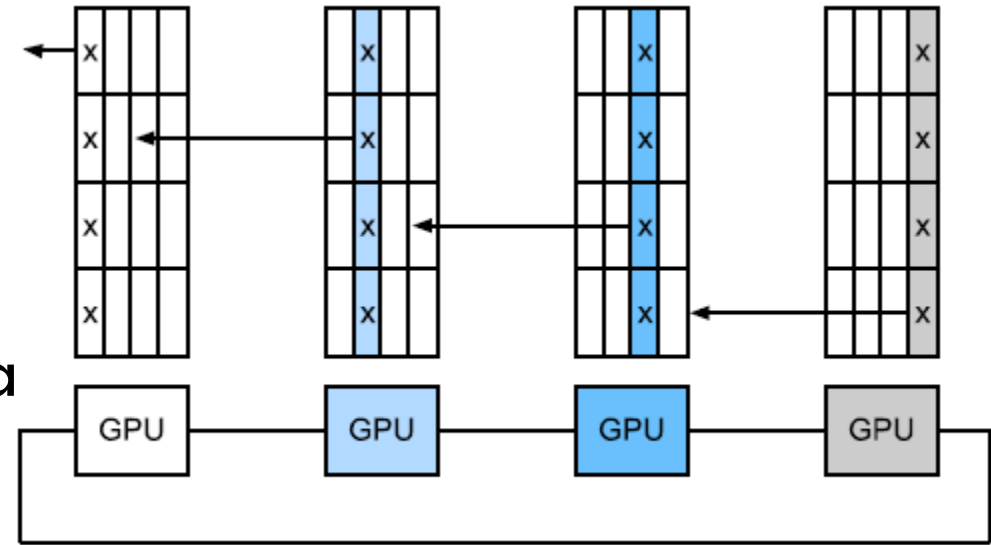
- Switches are more difficult to build
 - ▣ Any pair of nodes can connect directly
- Hypercube topology only have connections along the edges of the hypercube
 - ▣ Doubling the nodes only increases the longest path by one
 - ▣ Ring topology is a subset



Ring Synchronization

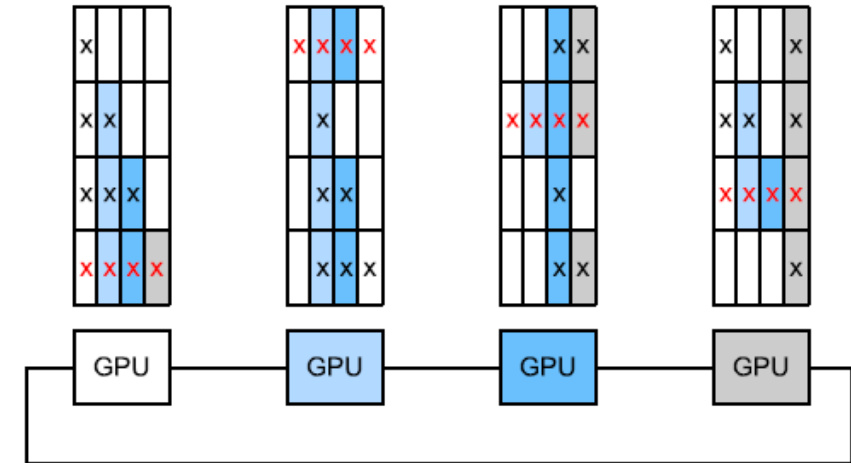
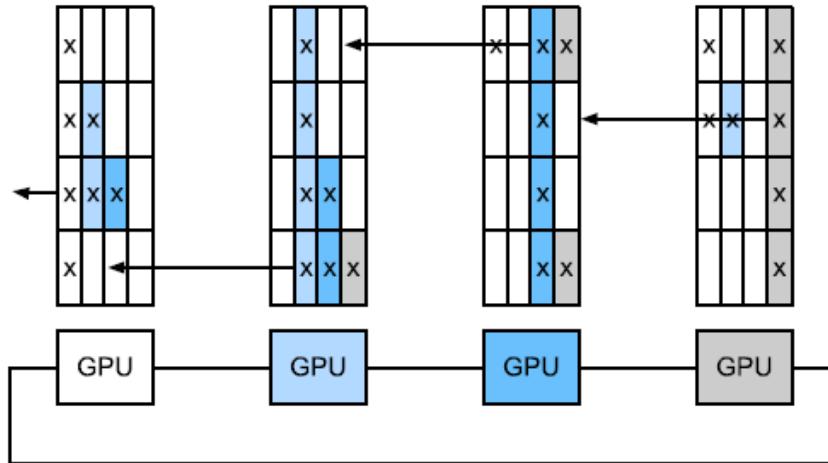
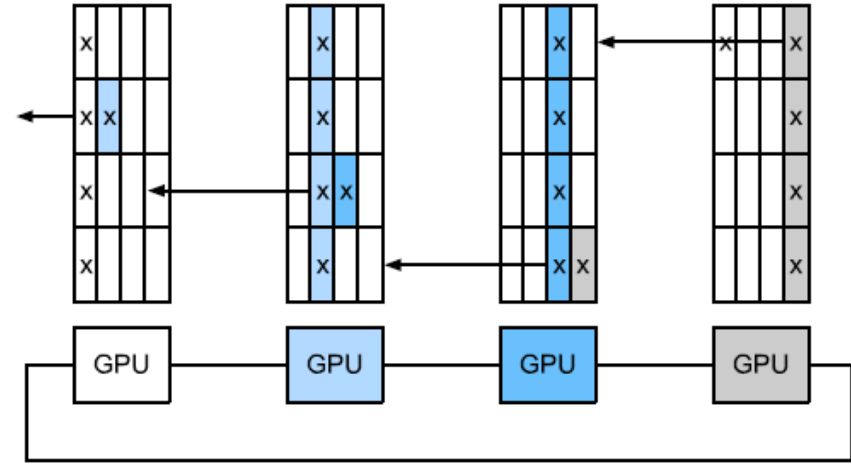
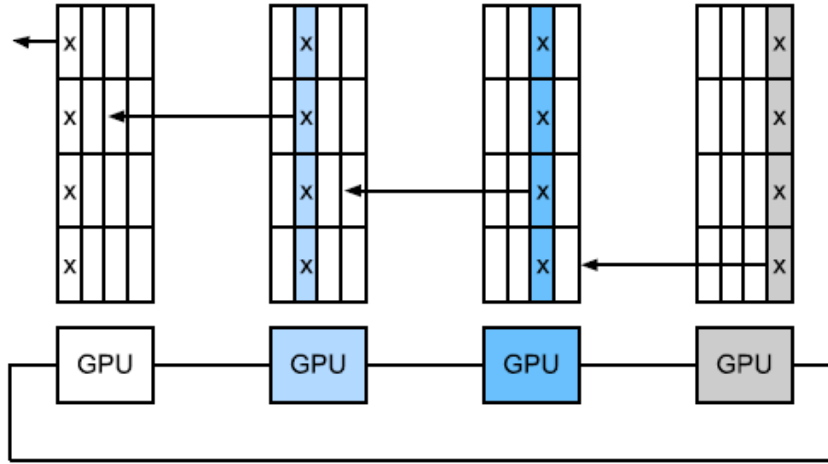
28

- Each node on a n -ring holds $\frac{1}{n}$ of the data
 - ▣ Node i holds part i of the data
- At each step $t = 0, \dots, n - 1$, each node
 - ▣ Sends part $i + t \bmod n$ data to its left neighbor
 - ▣ Receives part $i + t + 1 \bmod n$ data from its right neighbor
- After n steps, all nodes have all the data
- If T is the total time to send all the data
 - ▣ Each step takes time T/n
 - ▣ All n steps take time T



Ring Synchronization

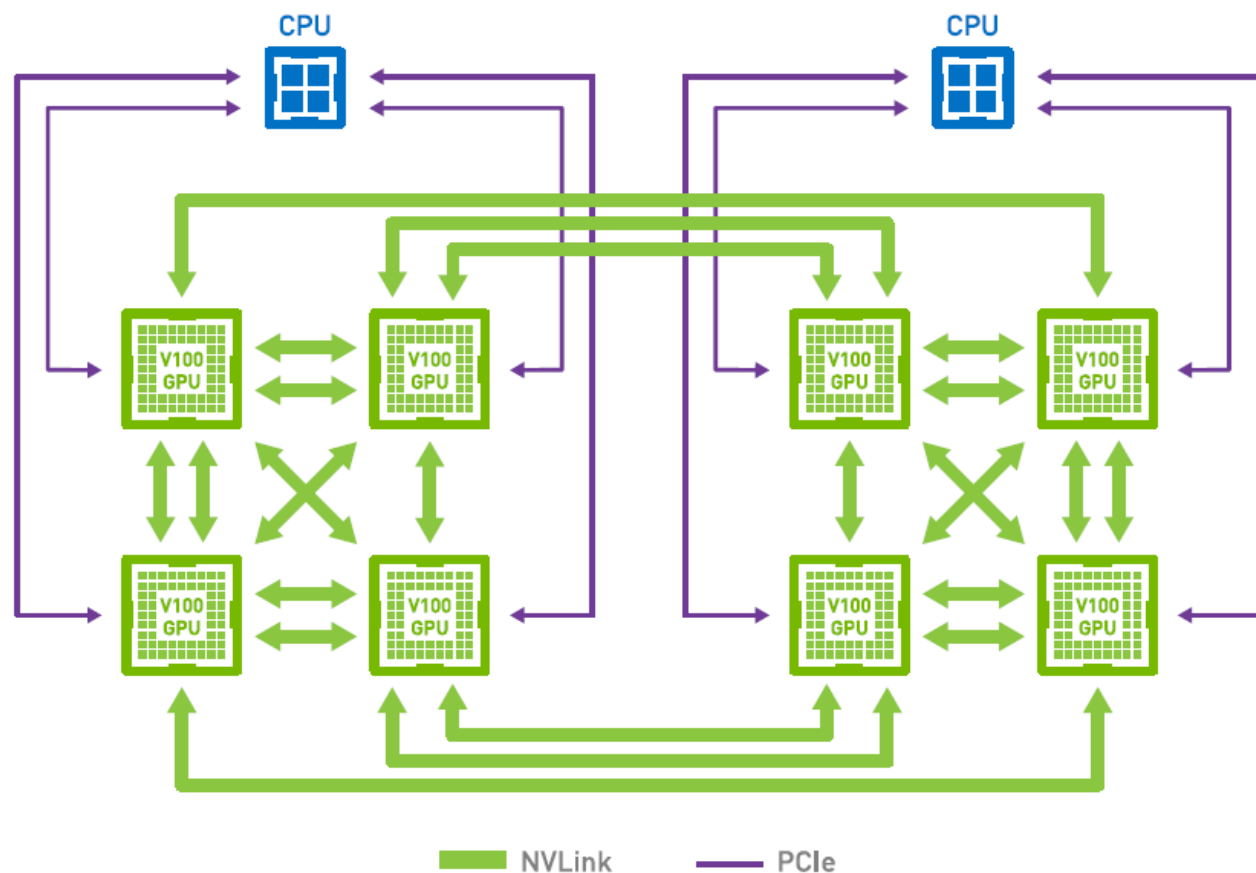
29



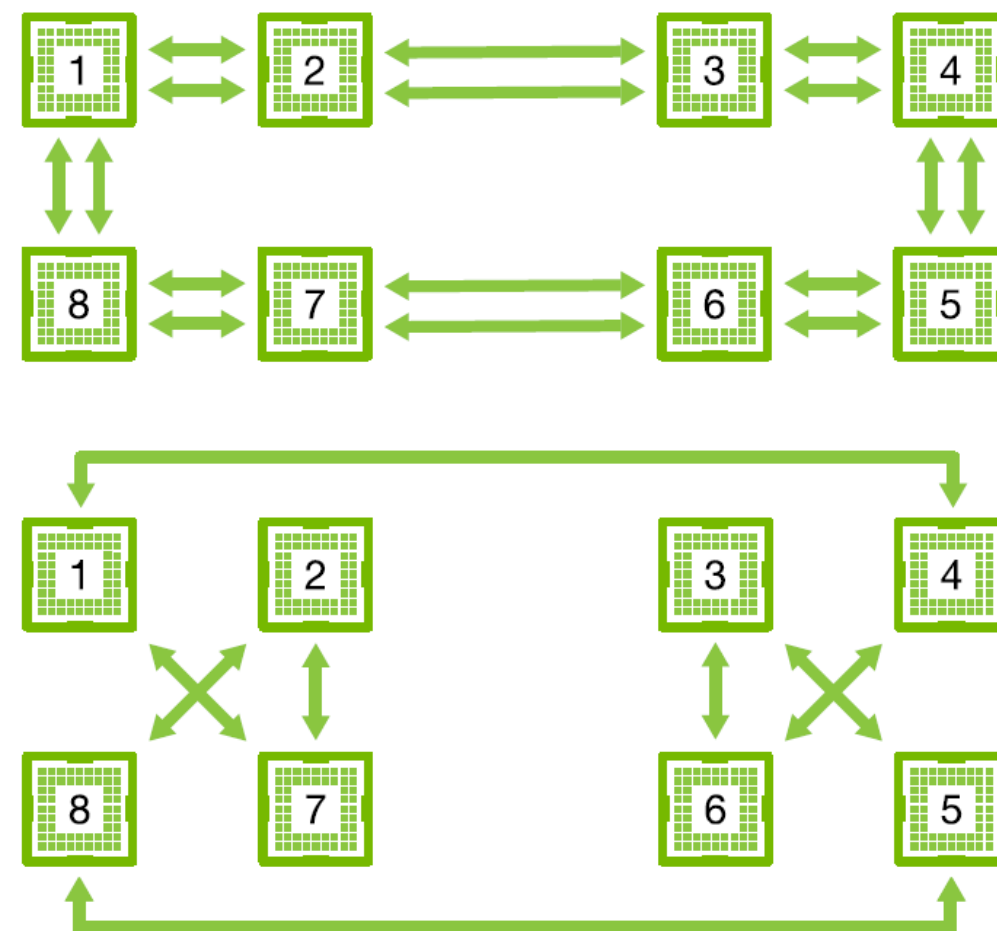
Custom Topology

30

NVLink on 8 V100 GPUs



Decomposes into 2 rings

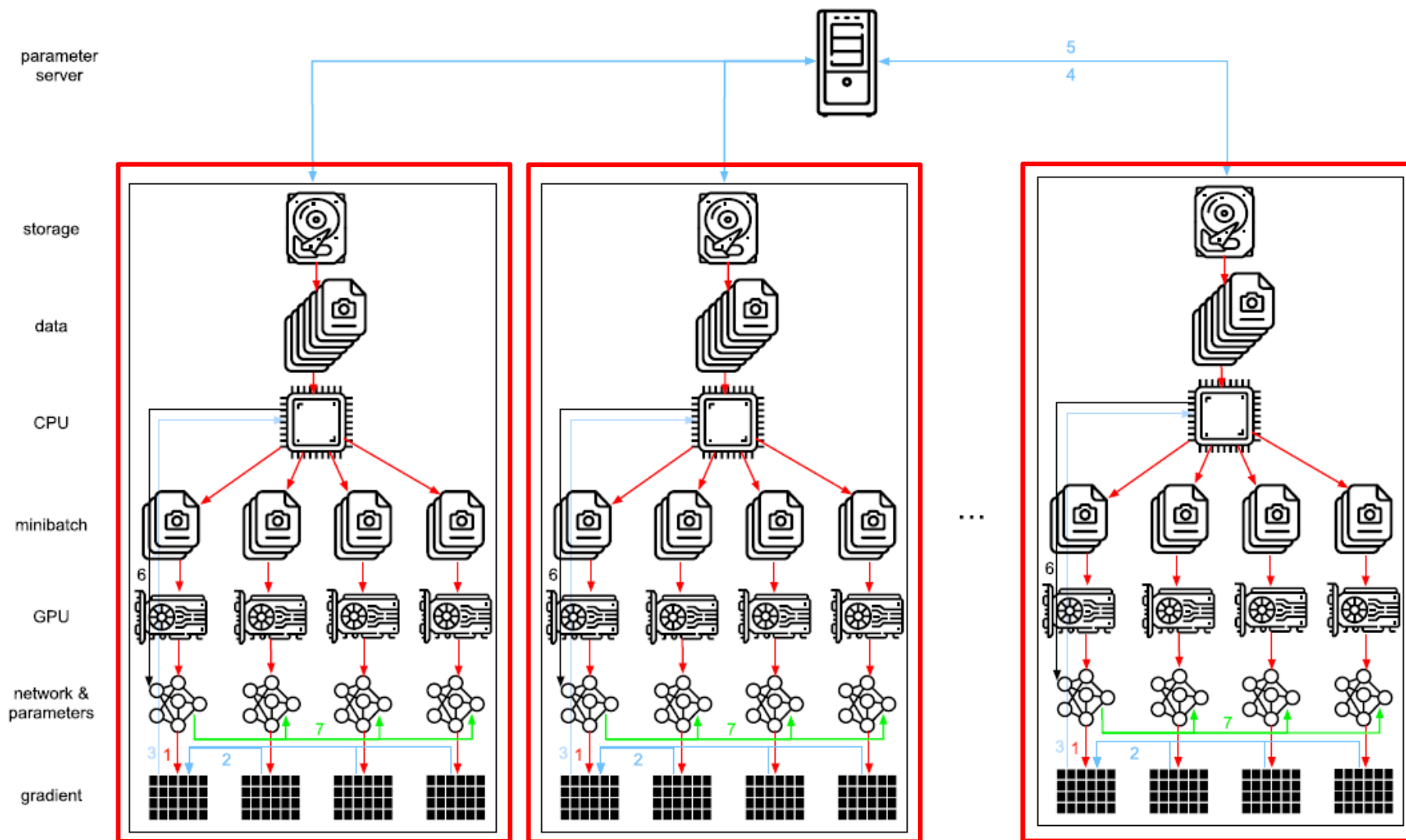


Multi-Machine Training

31

Train on more than one machine

Need to synchronize tasks, since machine runs at slightly different speeds

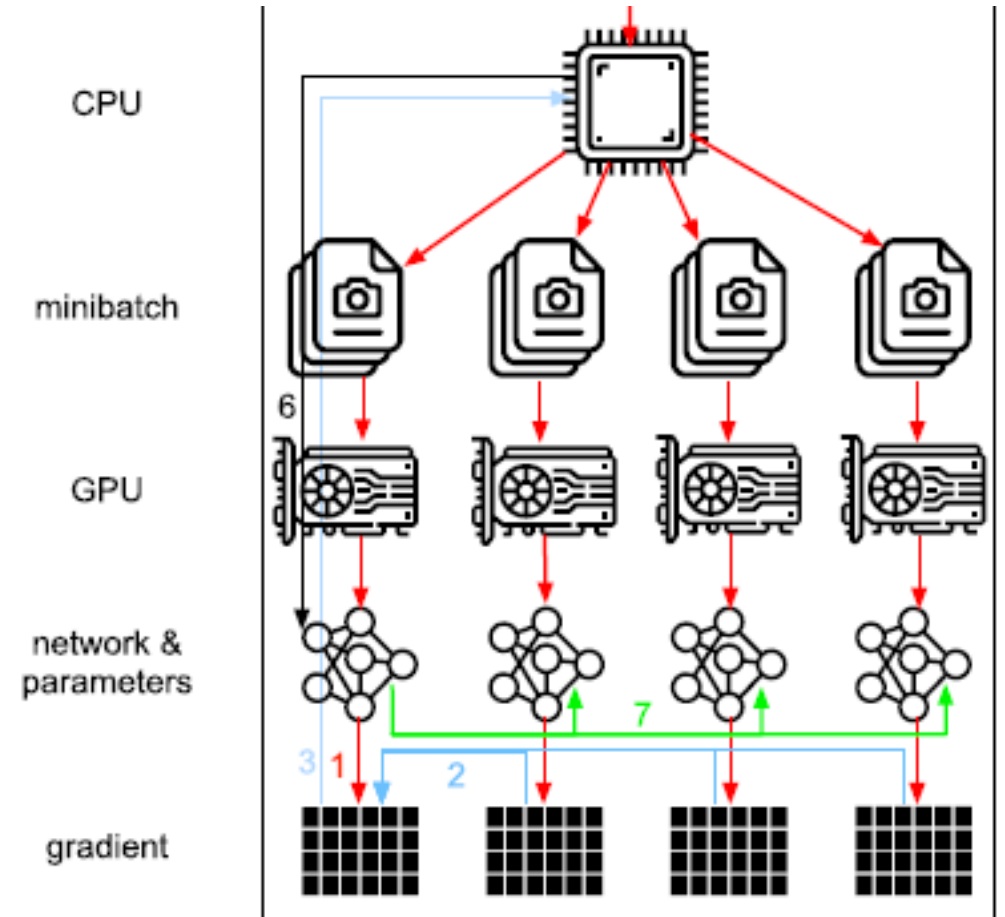


Training with Centralize Parameter Server

32

Centralize Parameter Server Example

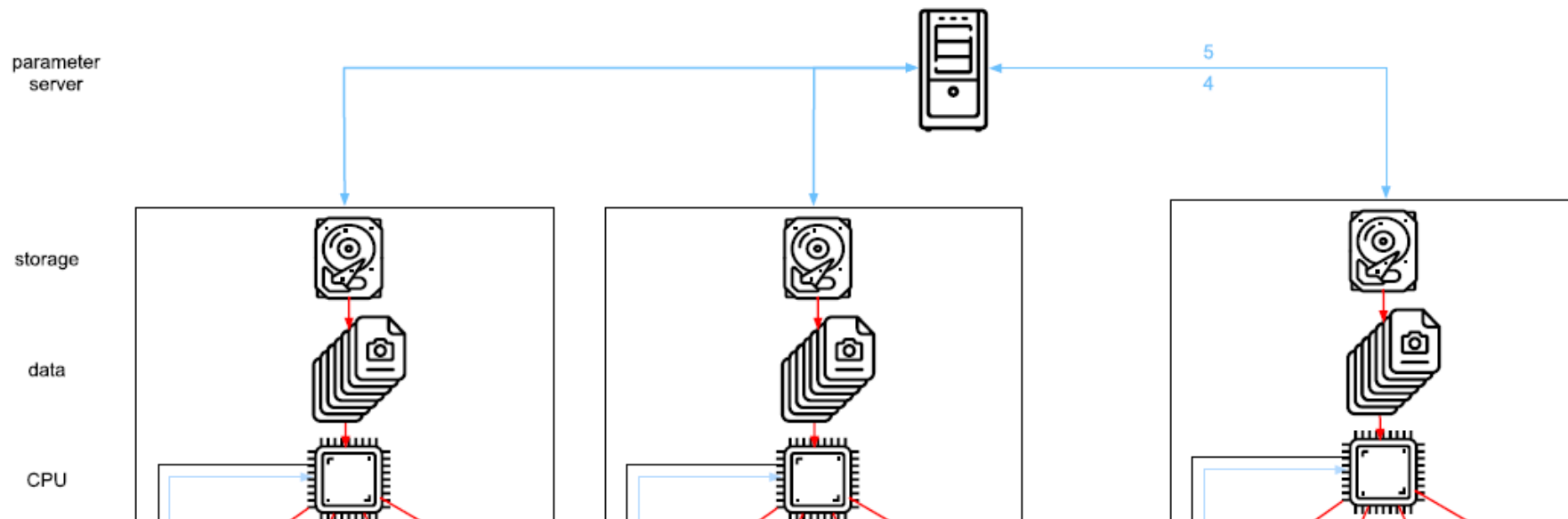
1. Data split across machines and GPUs. Each GPU computes gradient
2. Gradients from local GPUs are aggregated on one GPU
3. Send gradients to CPU



Training with Centralize Parameter Server

33

4. The CPUs send the gradients to a central parameter server
5. Update parameters with the aggregated gradients . Broadcast updated parameters to all CPUs

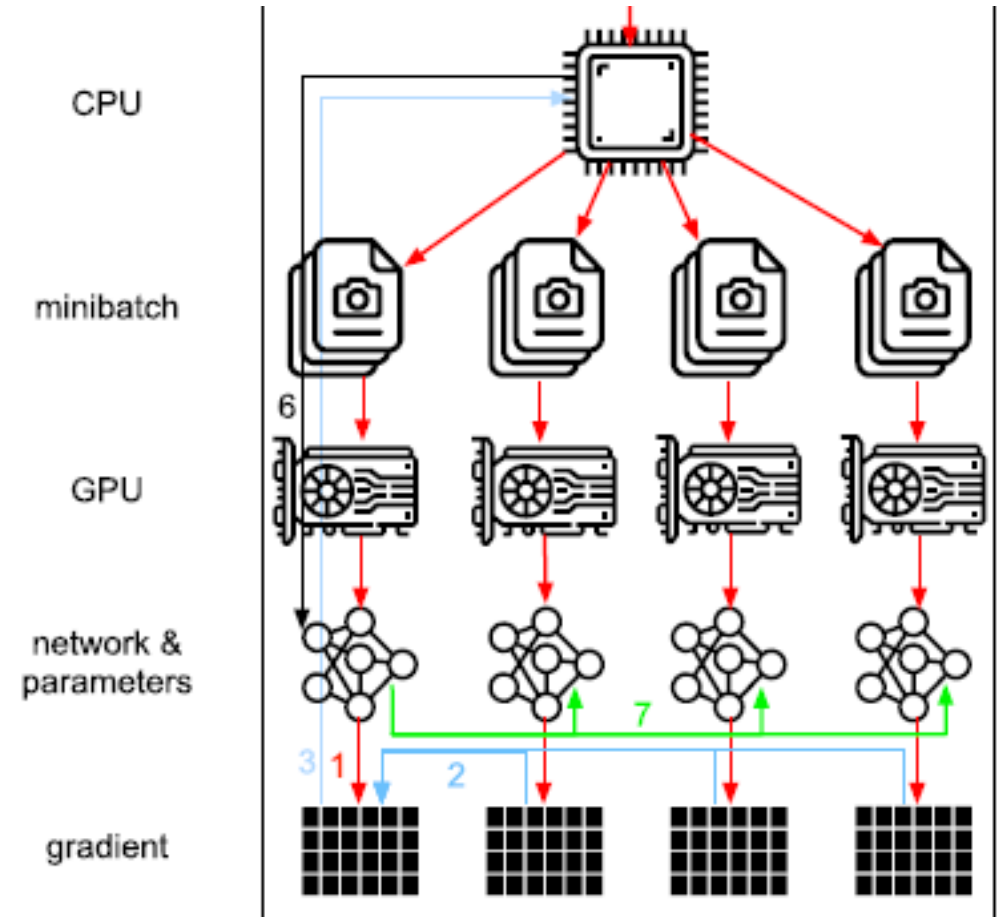


Training with Centralize Parameter Server

34

Centralize Parameter Server Example

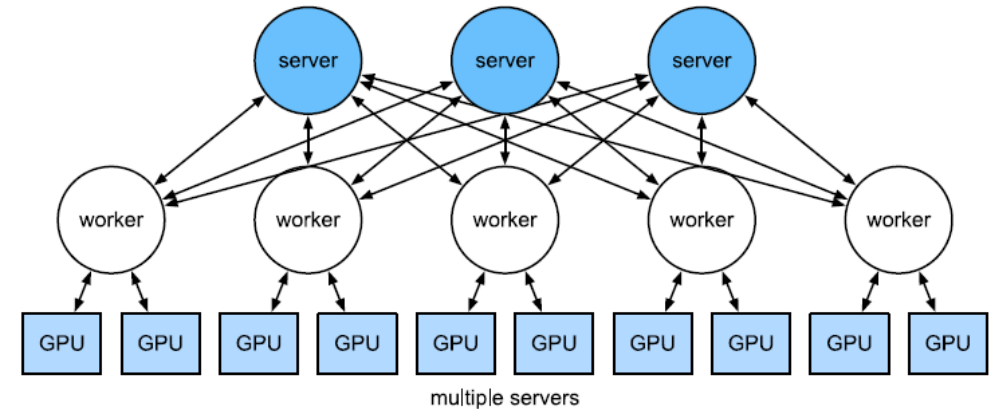
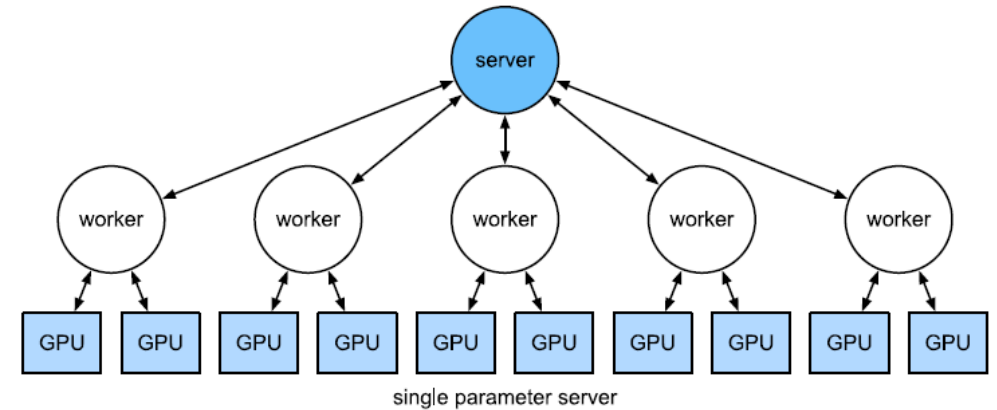
- Parameters are sent to one GPU
- Updated parameters are spread across all GPUs



Distributed Parameter Server

35

- In general, having centralized server in distributed computing is a bad idea
- With m workers it takes $\mathcal{O}(m)$ to send gradients, due to server bandwidth limitations
- With n servers it takes $\mathcal{O}\left(\frac{m}{n}\right)$



Parameter Server Operations

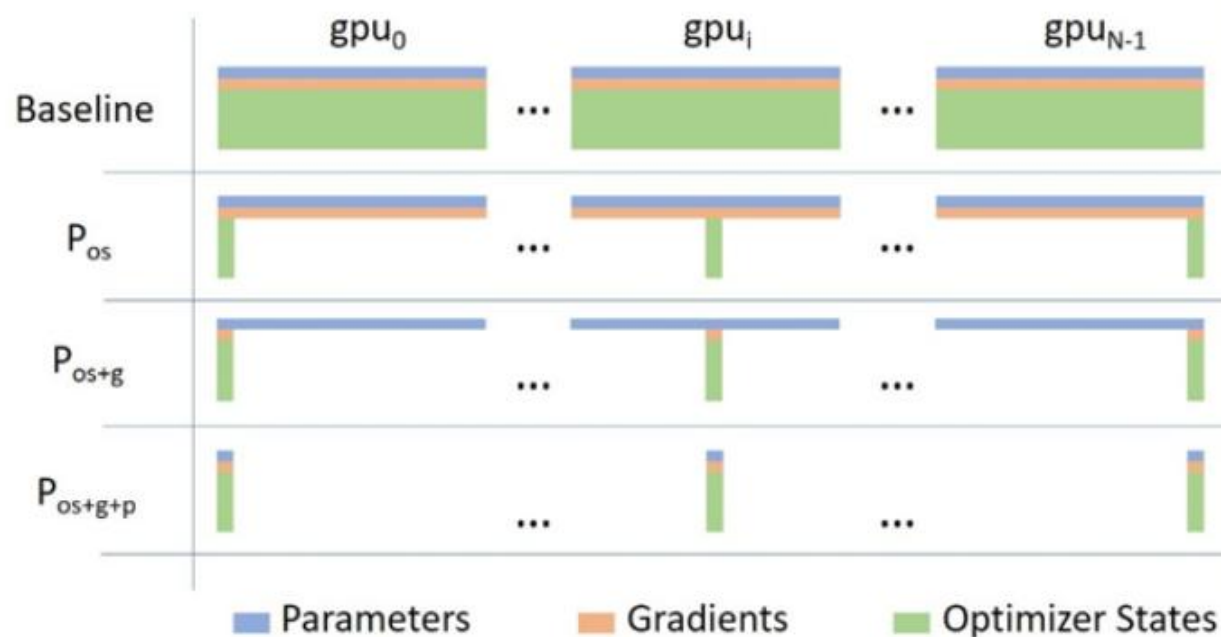
36

- Implements specialized key-value stores
 - ▣ **push(key, value)** sends a particular gradient (value) from worker to a common storage. There the value is aggregated, e.g., summing.
 - ▣ **pull(key, value)** retrieves values from common storage
- Assumes the aggregation operation is commutative (e.g., sum). The order of the push does not matter
- Synchronization operations are hidden behind the push/pull operations

Zero Redundancy Optimization (ZeRO)

37

- Uses data parallelism and pipeline parallelism
- Avoid memory overhead by broadcasting data as needed
- Can train a trillion-parameter model
- ZeRO-2 can train BERT in 44 minutes using 1024 NVIDIA V100
- See animation



<https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/>