UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CS 498: Machine Learning System Spring 2026

Minjia Zhang

The Grainger College of Engineering

Logistics

Two guest lectures from industry (remote)

- March 10, Meta, Zechun Liu (online)
- March 12, Amazon, Yida Wang (online)

Today

Training with Heterogeneous Memory

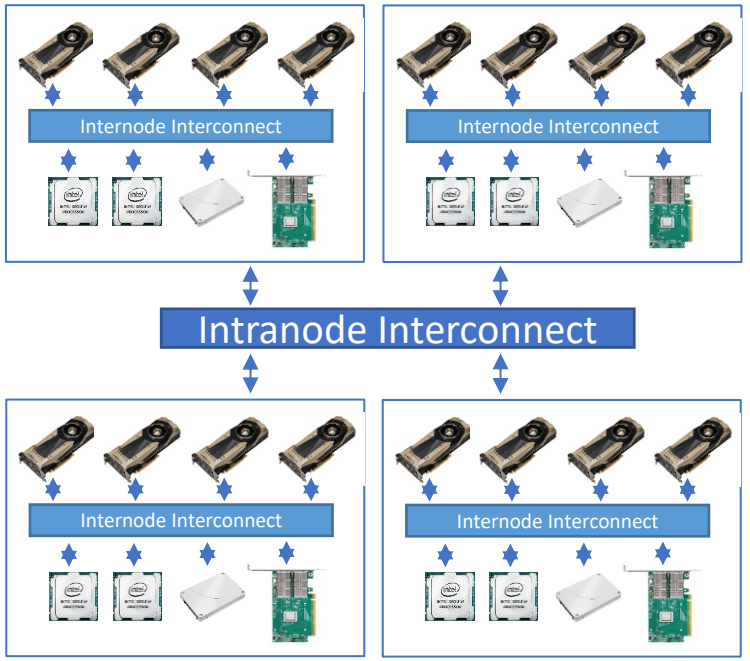
- ZeRO-Offload

Learning Objective

- Understand the how heterogeneous memory system can be leveraged to overcome GPU memory limits
- Explain the design principles behind ZeRO-Offload

Billon-Scale Model Training - Scale Out Large Model Training

- Model parallelism (Megatron-LM)
  - Partition the model states vertically across multiple GPUs.
- Pipeline parallelism (PipeDream, Megatron-LM-v2)
  - Partition the model states horizontally across layers.
- ZeRO: Zero Redundancy Optimizer (ZeRO)
  - Split the training batch across multiple GPUs without model states duplication.



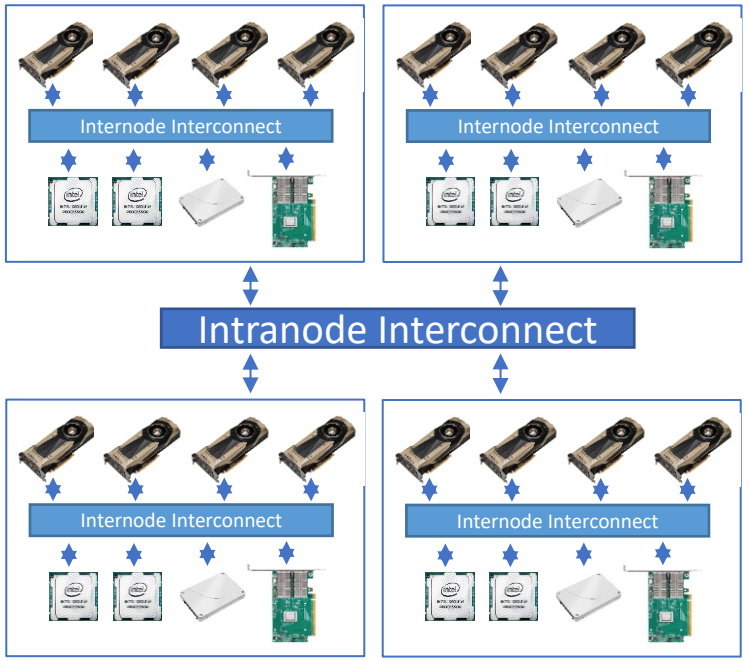
Distributed GPU Cluster

Billon-Scale Model Training - Scale Out Large Model Training

- Model parallelism (Megatron-LM)
  - Partition the model states vertically across multiple GPUs.

Require multiple GPUs resources

- ZeRO: Zero Redundancy Optimizer (ZeRO, SC'19)
  - Split the training batch across multiple GPUs without model states duplication.

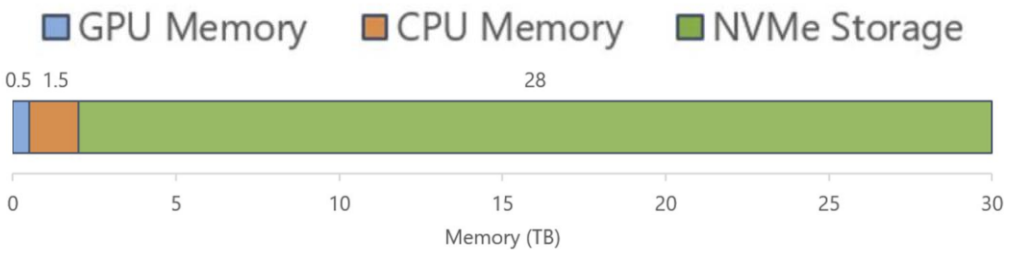


Distributed GPU Cluster

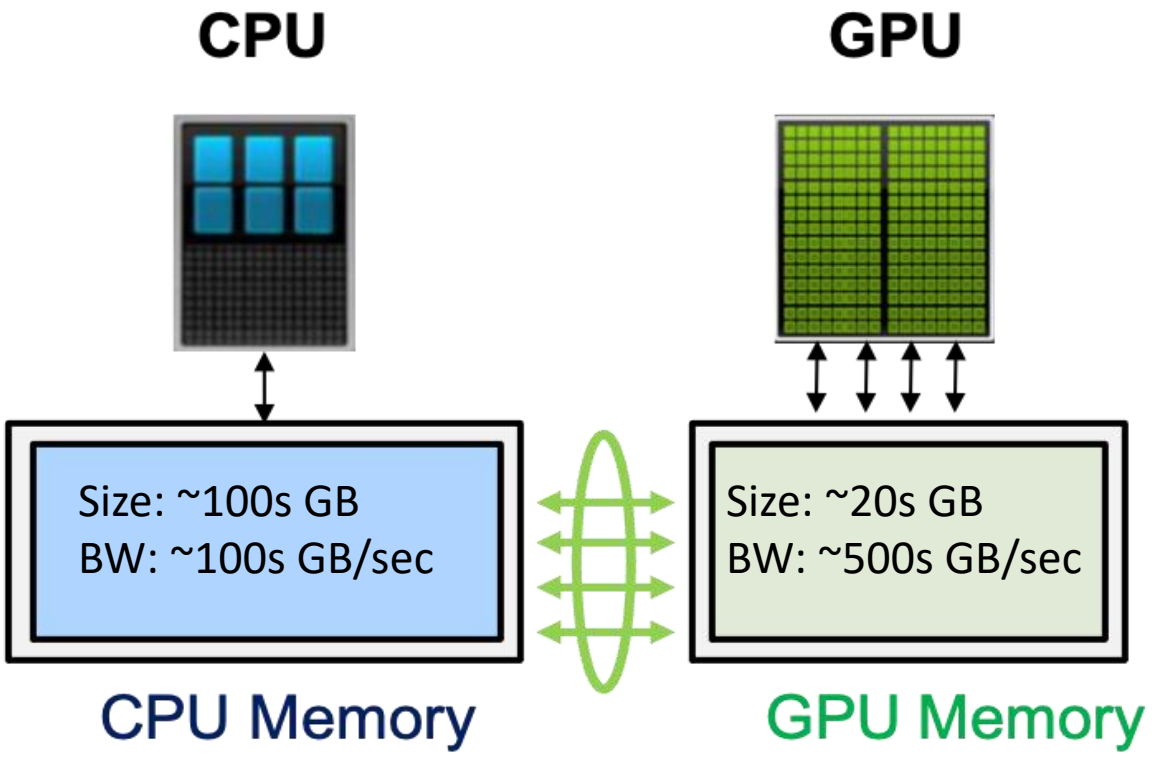
Beyond the GPU Memory

- Modern clusters have heterogeneous memory systems
- GPU memory comprises a small fraction
- Can we extend existing training technology to use CPU/NVMe memory?

Memory available on a Single DGX-2 Node

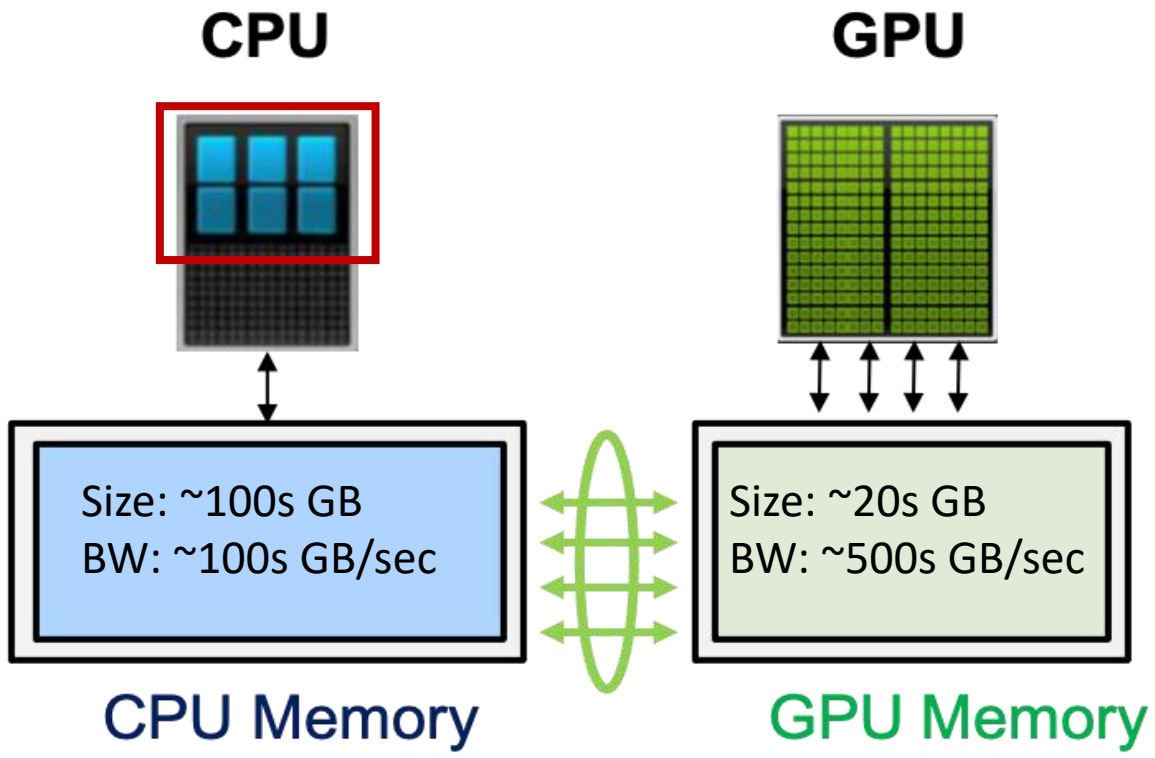


Challenges of Leveraging Heterogeneous Memory for DNN Training



Challenges of Leveraging Heterogeneous Memory for DNN Training

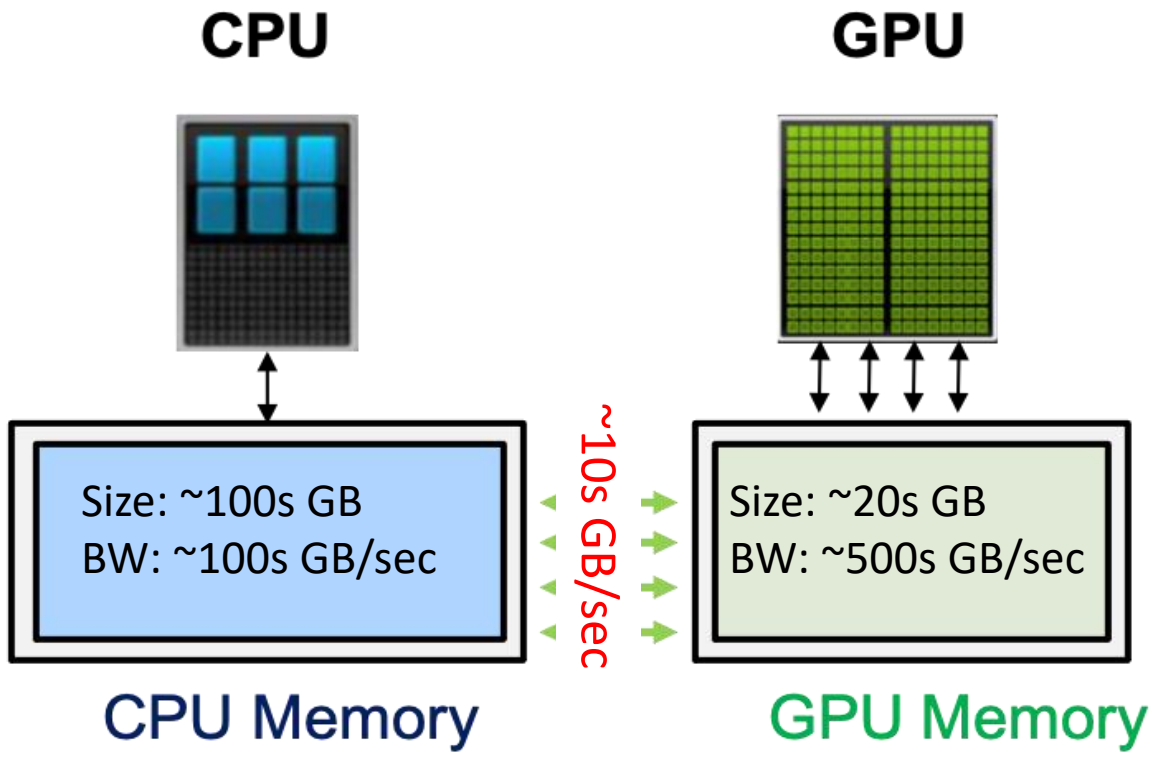
C1: Beyond CPU memory, can we also exploit CPU compute?



Challenges of Leveraging Heterogeneous Memory for DNN Training

C1: Beyond CPU memory, can we also exploit CPU compute?

C2: Is CPU←→ GPU bandwidth sufficient to achieve comparable compute efficiency to non-offload strategies?

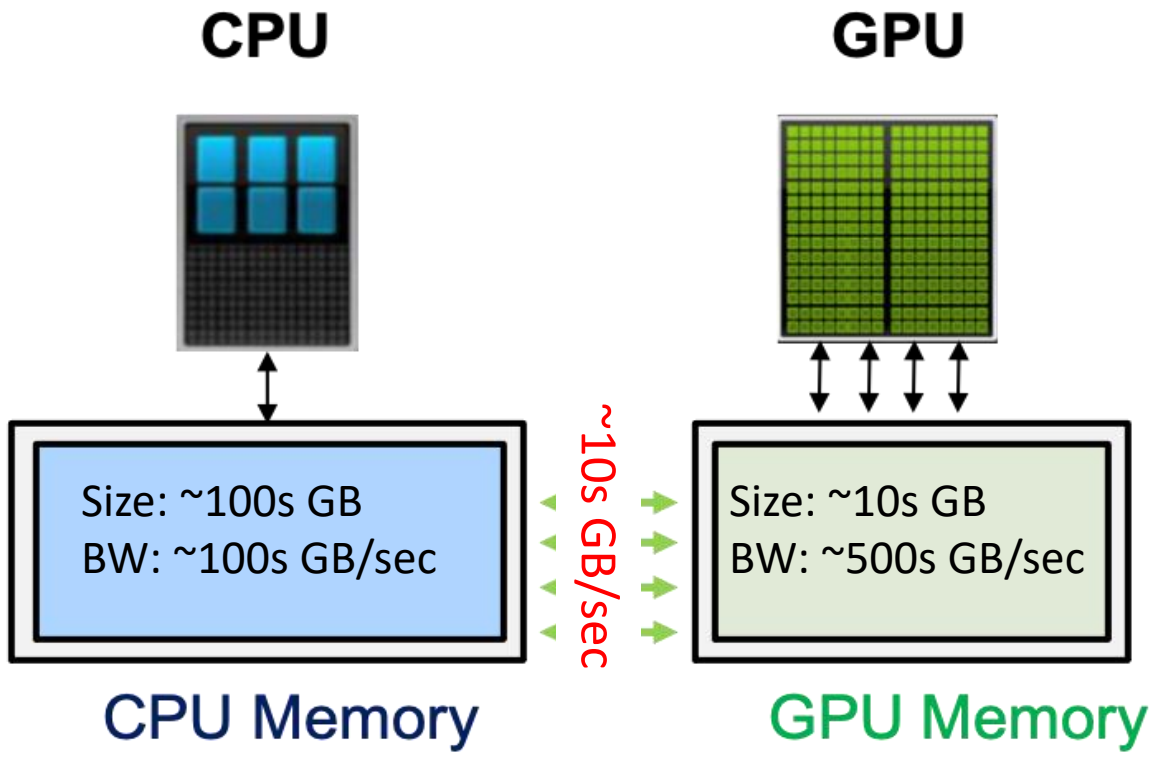


Challenges of Leveraging Heterogeneous Memory for DNN Training

C1: Beyond CPU memory, can we also exploit CPU compute?

C2: Is CPU←→ GPU bandwidth sufficient to achieve comparable compute efficiency to non-offload strategies?

C3: Scalability during multi-GPU training?



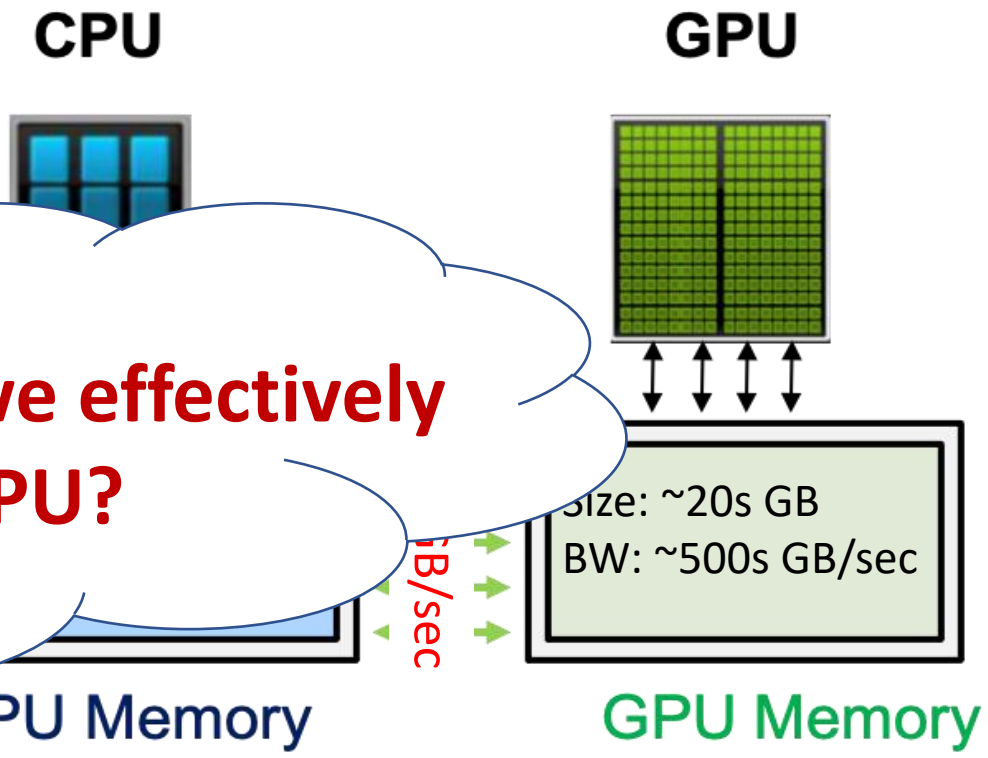
Challenges of Leveraging Heterogeneous Memory for DNN Training

C1: Beyond CPU memory, can we also exploit CPU compute?

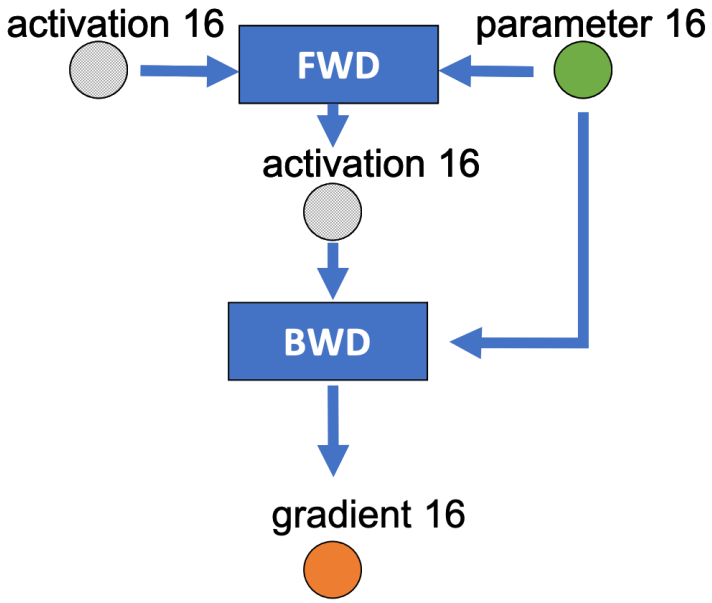
C2: Is CPU←→ to achieve co efficiency to

C3: Scalability during multi-GPU training?

Question: How do we effectively leverage CPU and GPU?

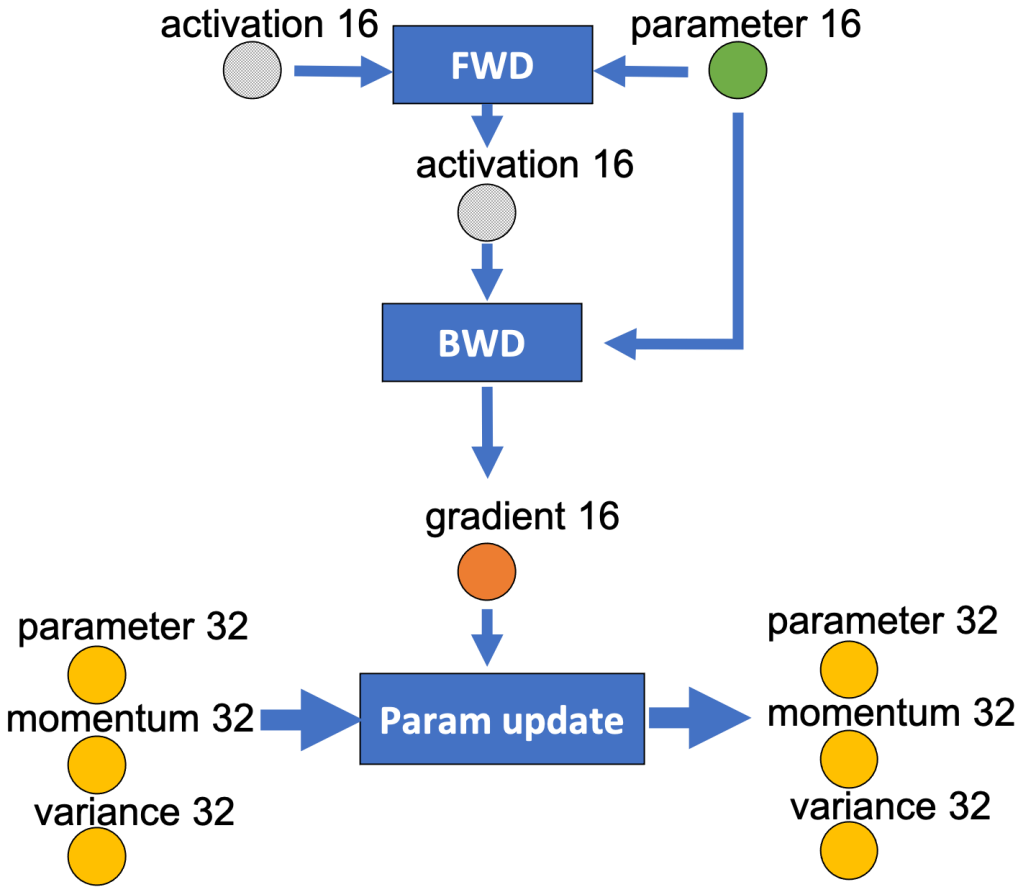


Mixed-precision Training



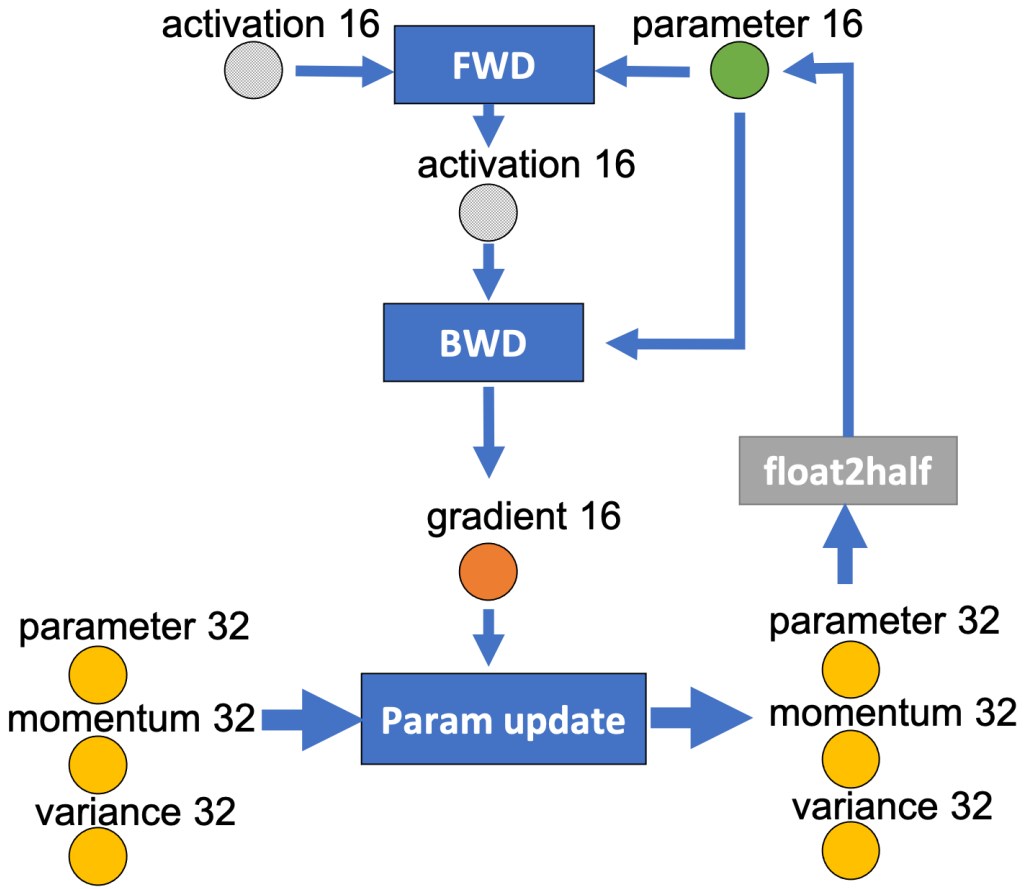
Mixed precision training iteration for a layer.

Mixed-precision Training



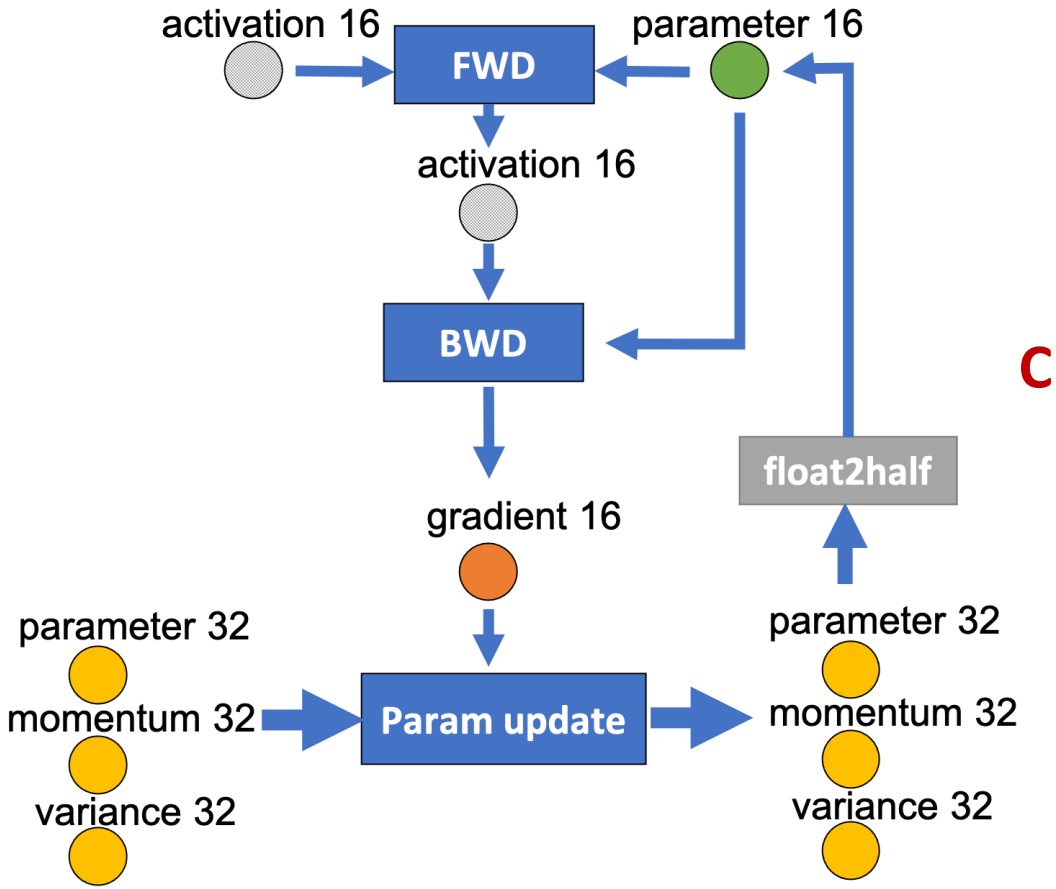
Mixed precision training iteration for a layer.

Mixed-precision Training



Mixed precision training iteration for a layer.

Formulating Heterogeneous Training as Graph Partitioning Problem

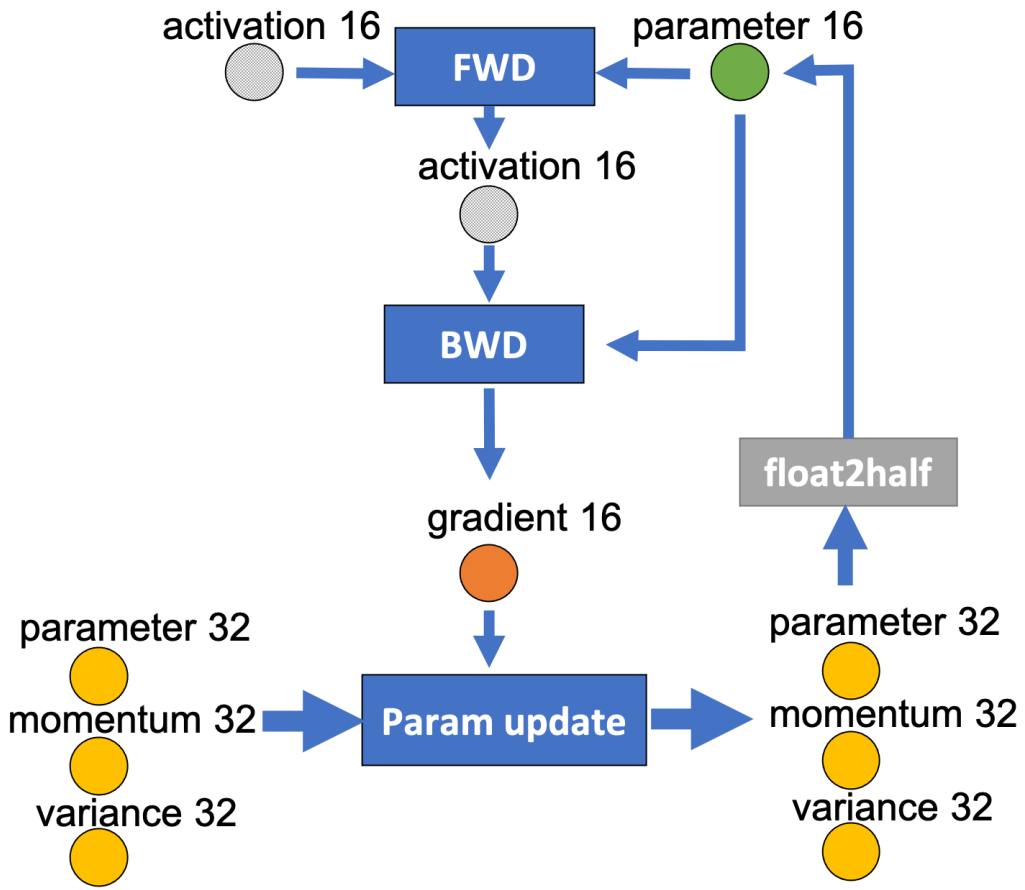


Challenge: how to partition?
NP-hard

Mixed precision training iteration for a layer.

Offload Strategy

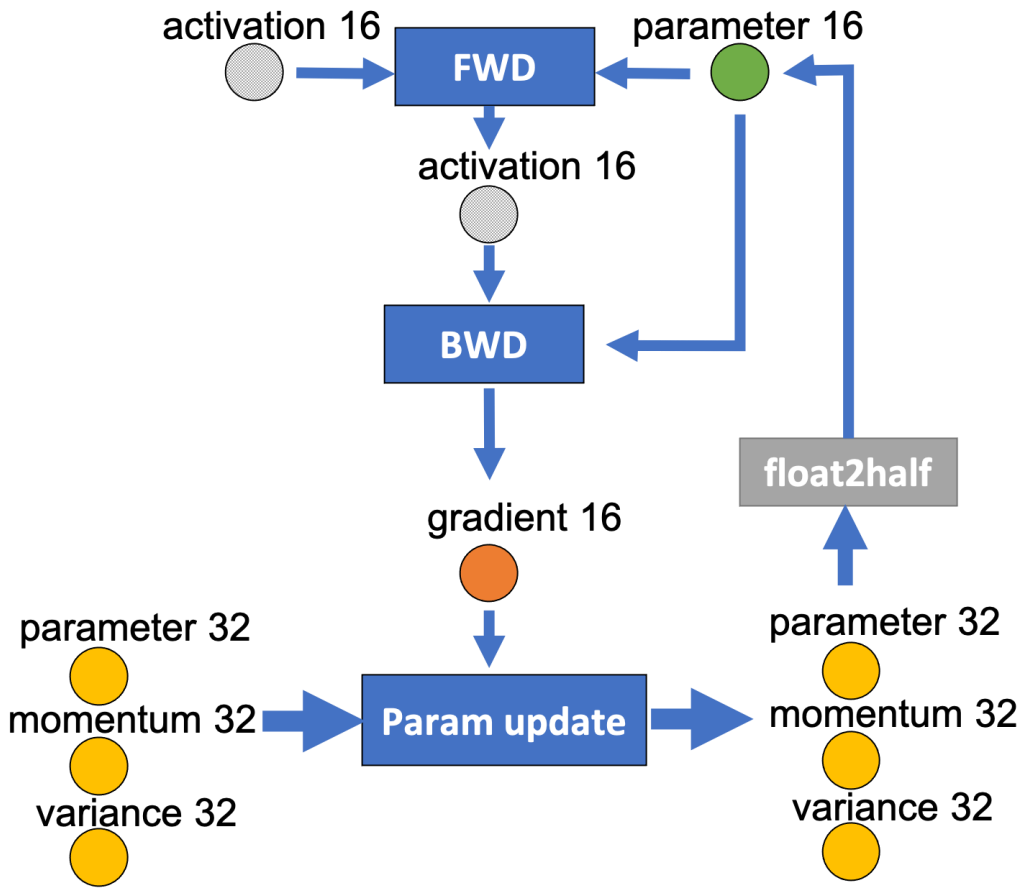
i. Few computation on CPU



Offload Strategy

i. Few computation on CPU

ii. Minimize of communication volume

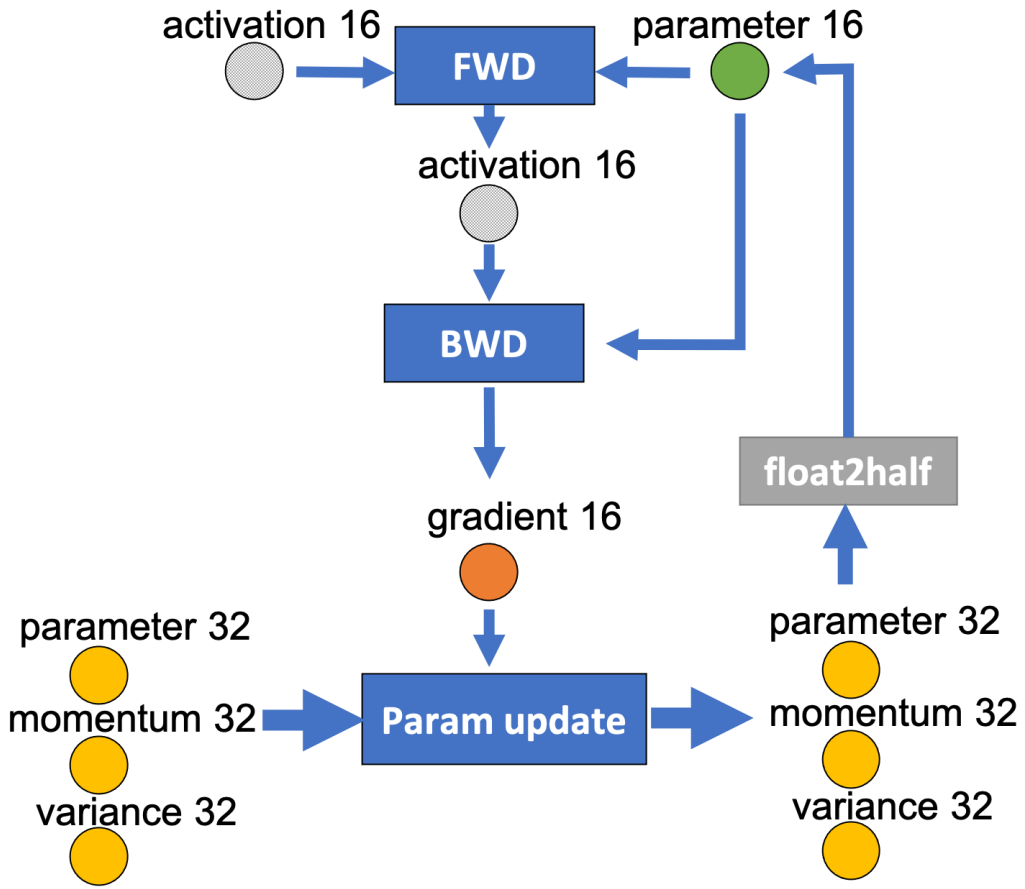


Offload Strategy

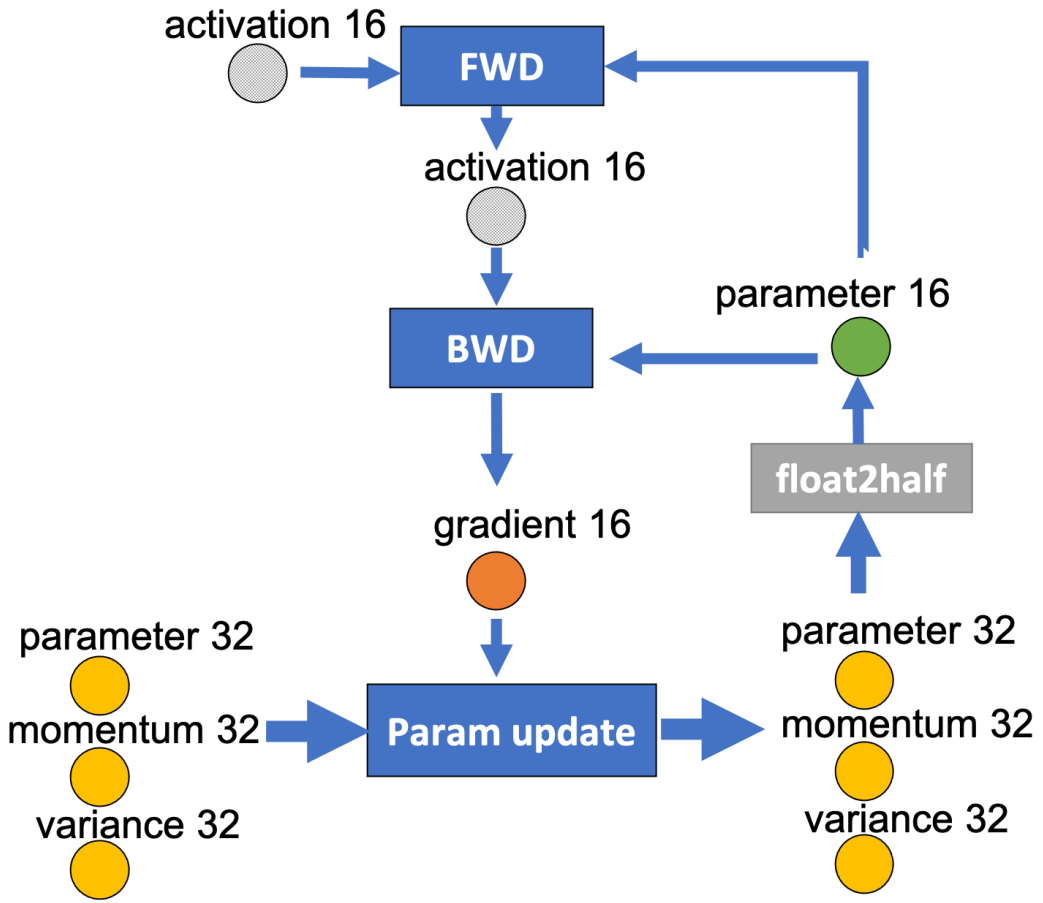
i. Few computation on CPU

ii. Minimize of communication volume

iii. Maximize memory saving while achieving minimum communication volume



Offload Strategy

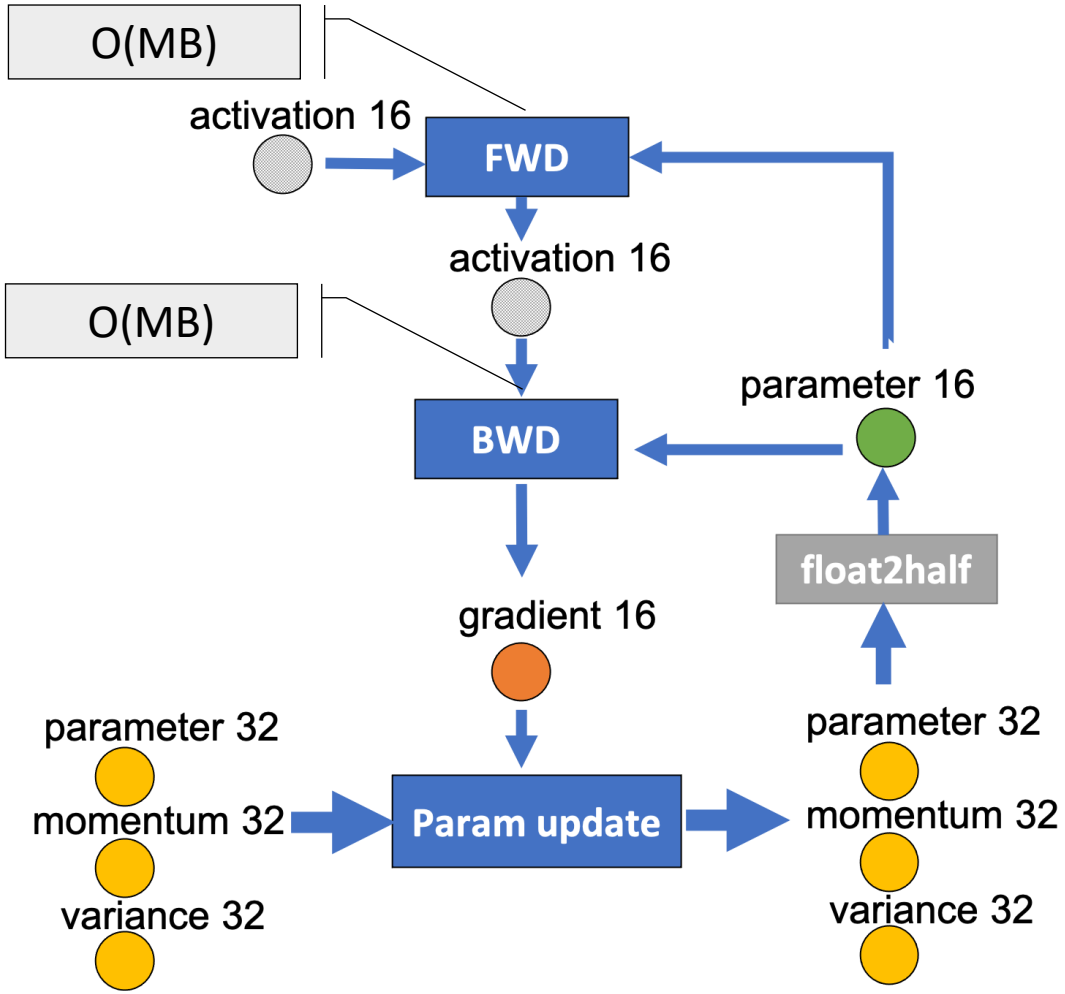


M: model param count
B: batch size

Computational Complexity
O (MB)

The dataflow of fully connected neural networks with M parameters.

Offload Strategy

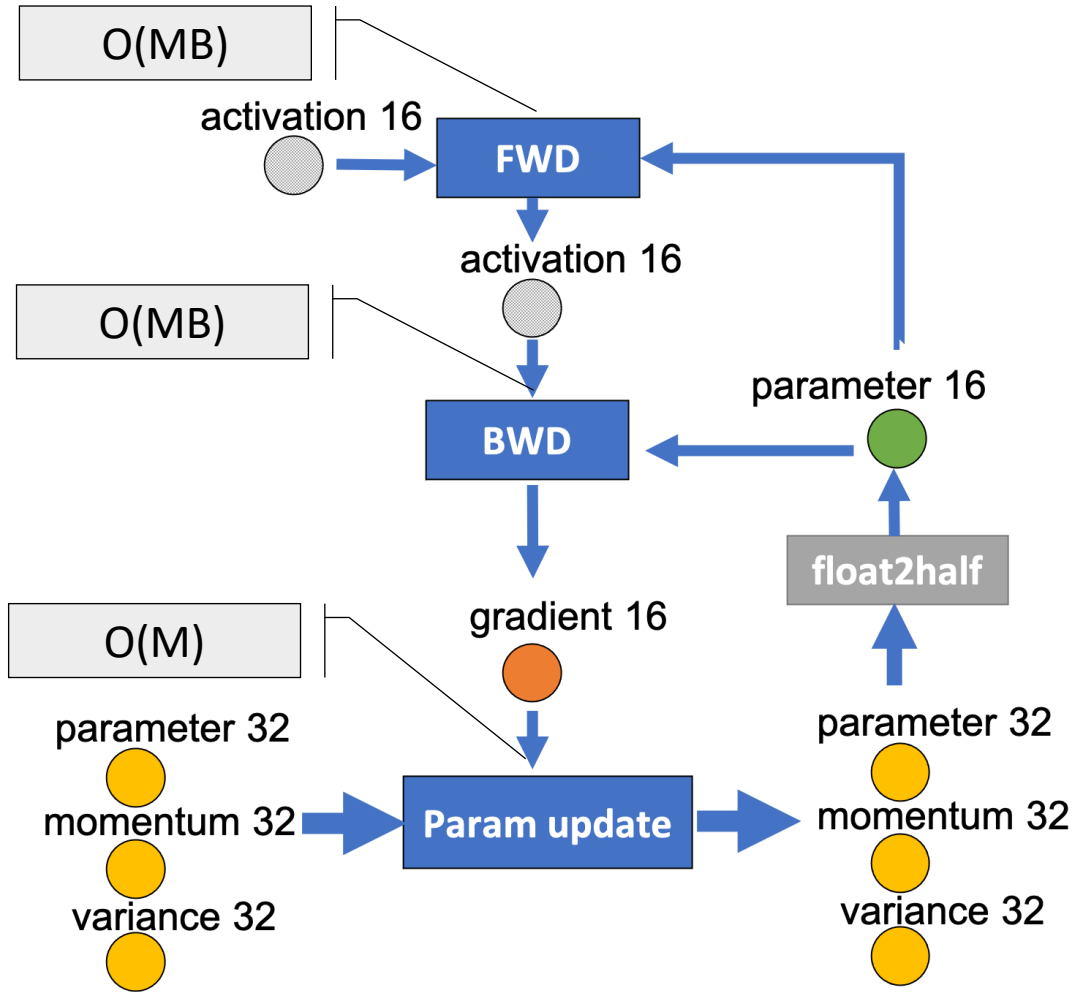


M: model param count
B: batch size

Computational Complexity
O (MB)

The dataflow of fully connected neural networks with M parameters.

Offload Strategy

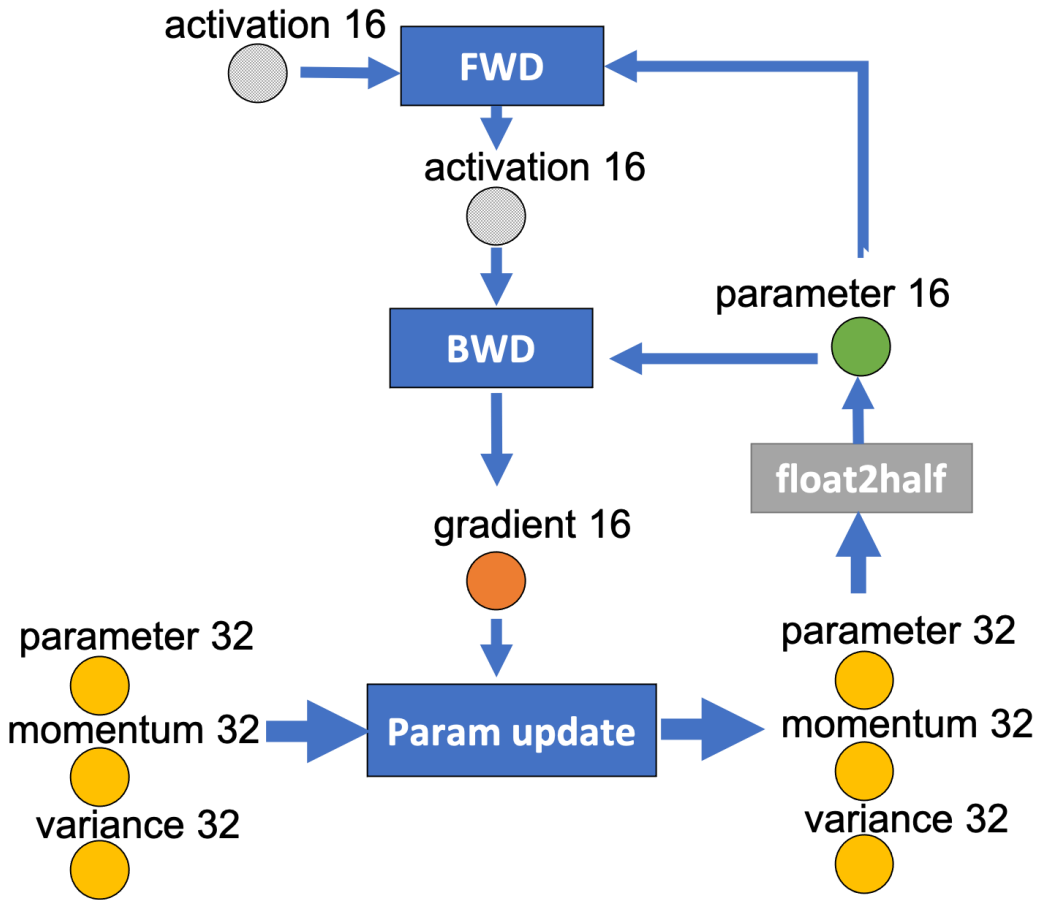


M: model param count
B: batch size

Computational Complexity
O (MB)

The dataflow of fully connected neural networks with M parameters.

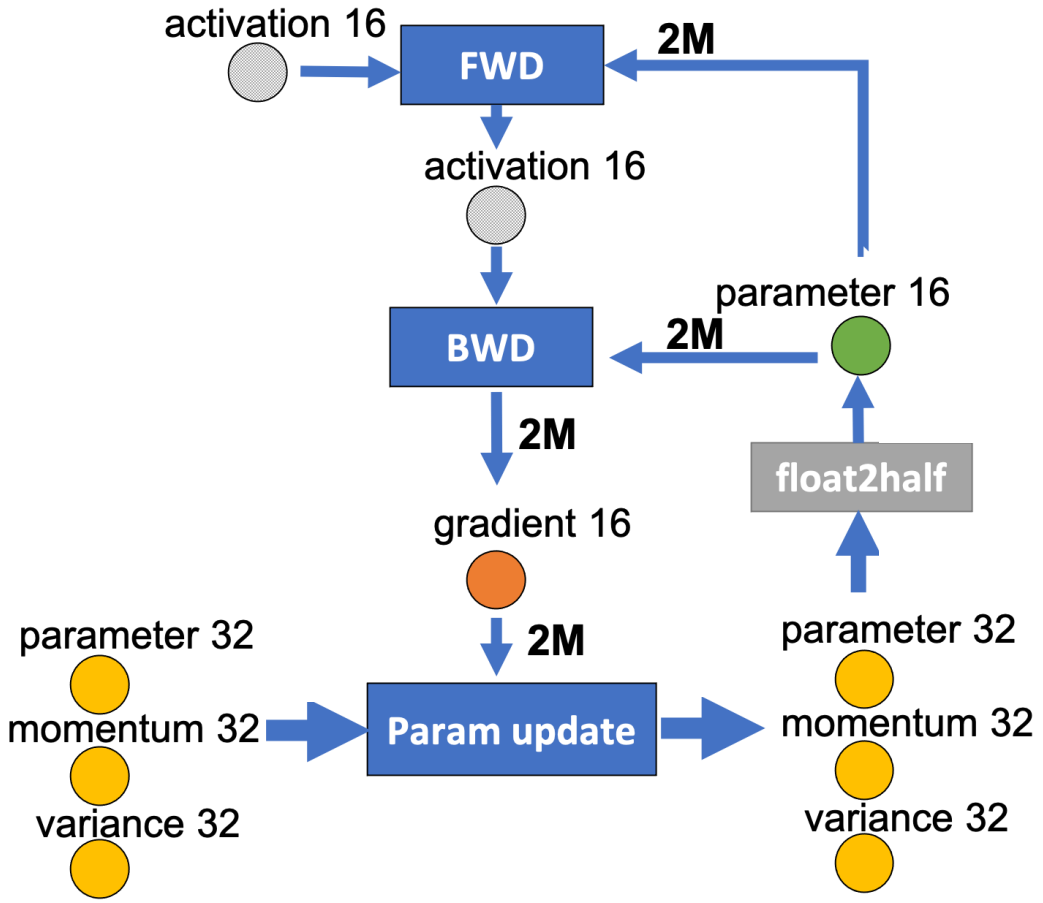
Offload Strategy



Data transfer volume?

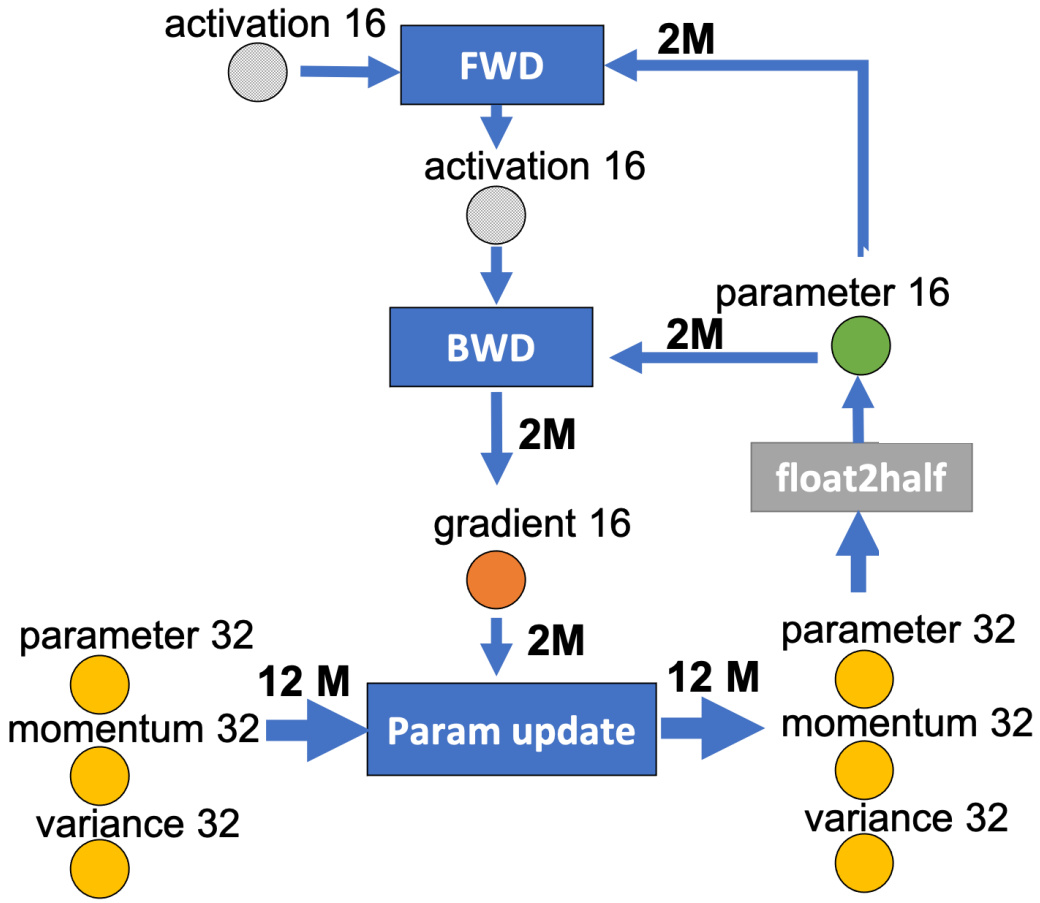
The dataflow of fully connected neural networks with M parameters.

Offload Strategy



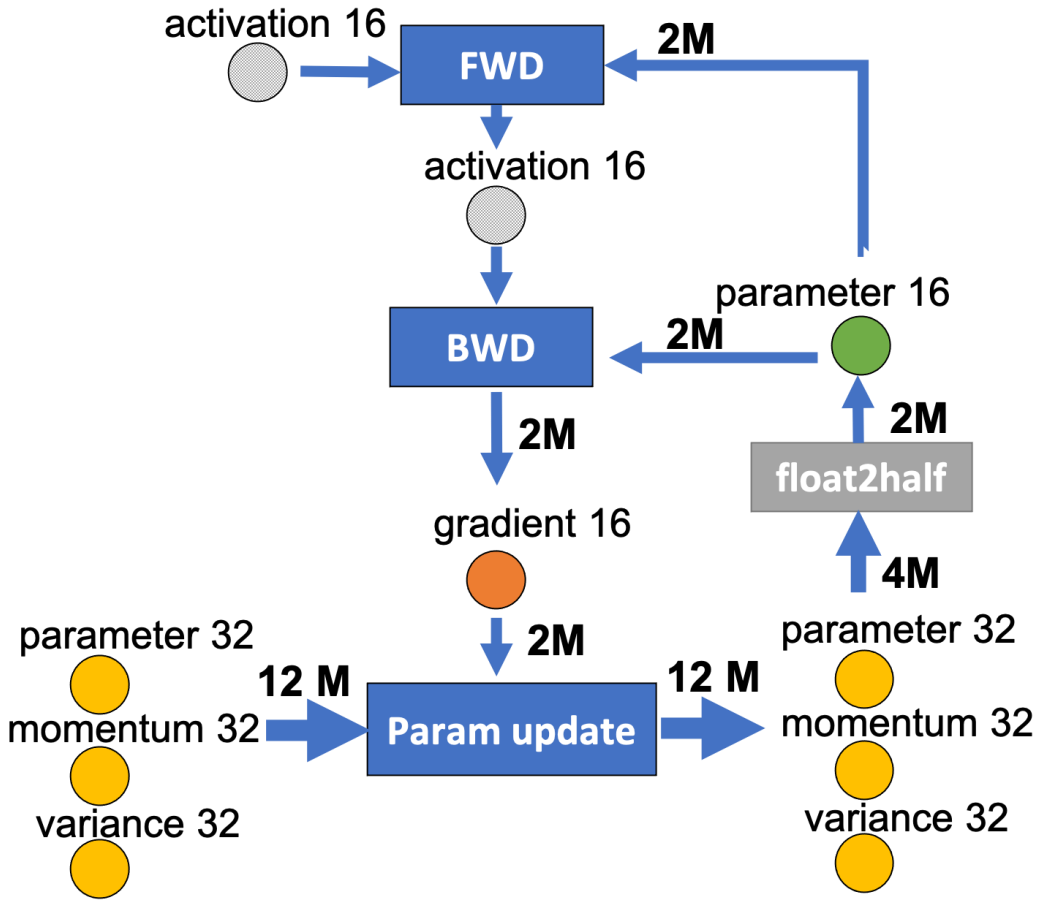
The dataflow of fully connected neural networks with M parameters.

Offload Strategy



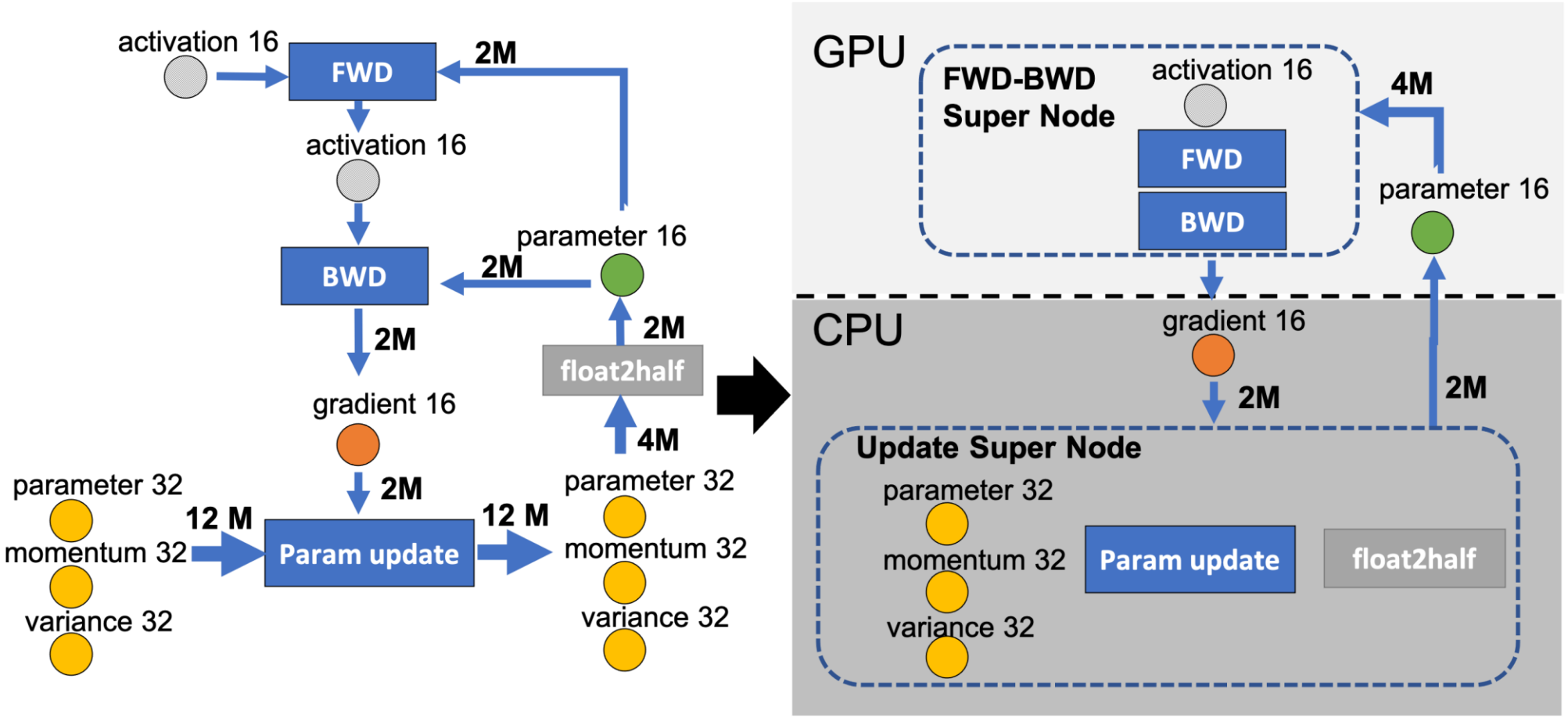
The dataflow of fully connected neural networks with M parameters.

Offload Strategy



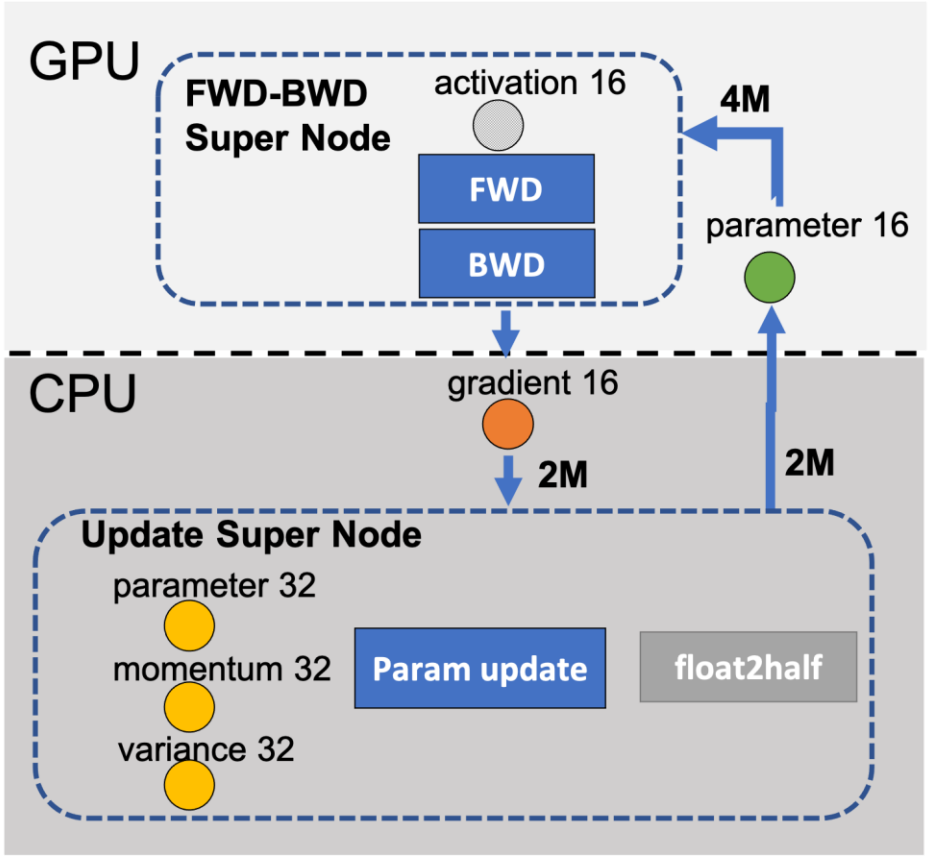
The dataflow of fully connected neural networks with M parameters.

Enable large model training by offloading data and compute to CPU



Offloading fp16 gradients and Update Super Node to CPU

Memory Savings



FWD-BWD	param16	gradient16	Update	Memory	Reduction
GPU	GPU	GPU	GPU	16M	1x(baseline)
GPU	GPU	CPU	GPU	14M	1.14x
GPU	GPU	GPU	CPU	4M	4x
GPU	GPU	CPU	CPU	2M	8x

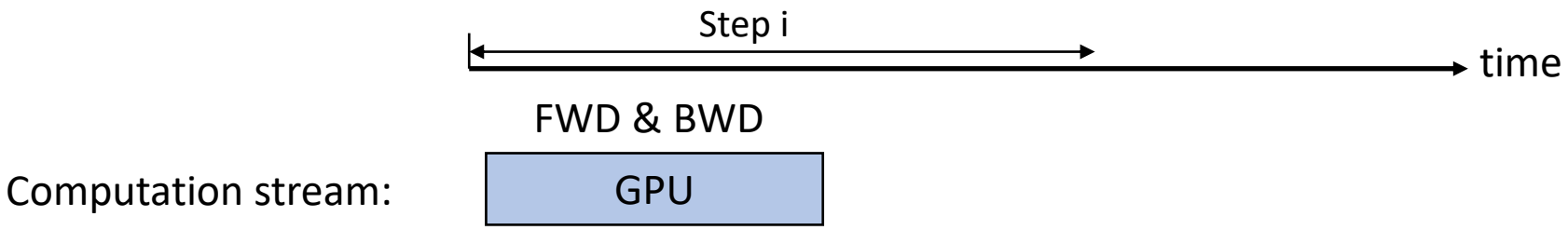
Memory savings for offload strategies that minimize communication volume compared to the baseline.

ZeRO-Offload Single GPU Schedule

Challenge: how and when to offload?

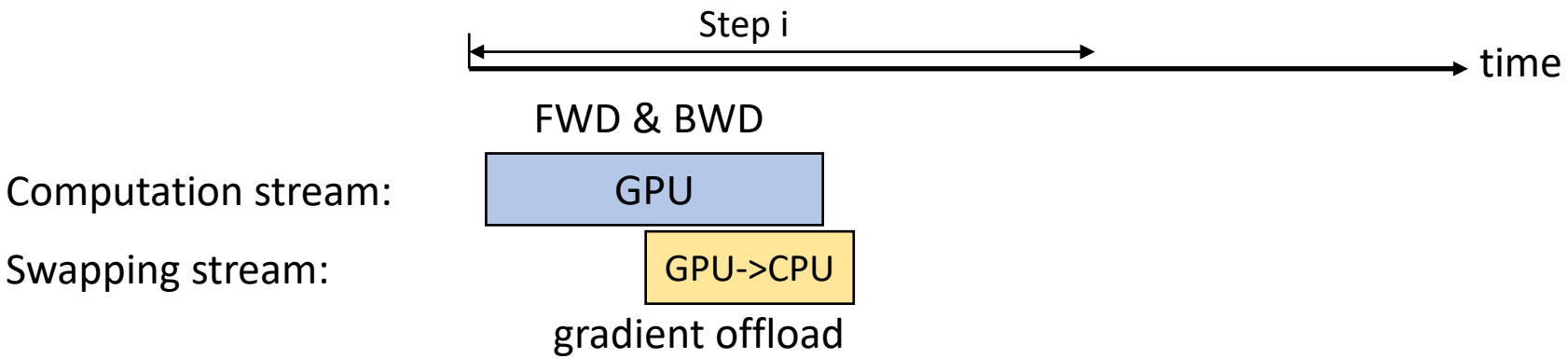
- Would offloading significantly increase total iteration time?
- Would CPU execution become a bottleneck?

ZeRO-Offload Single GPU Schedule



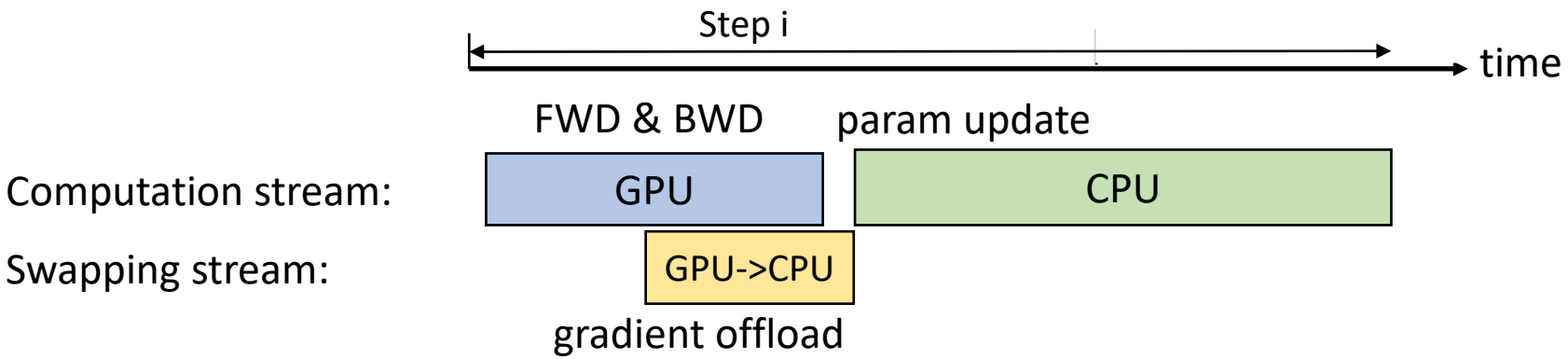
ZeRO-Offload training process on a single GPU.

ZeRO-Offload Single GPU Schedule

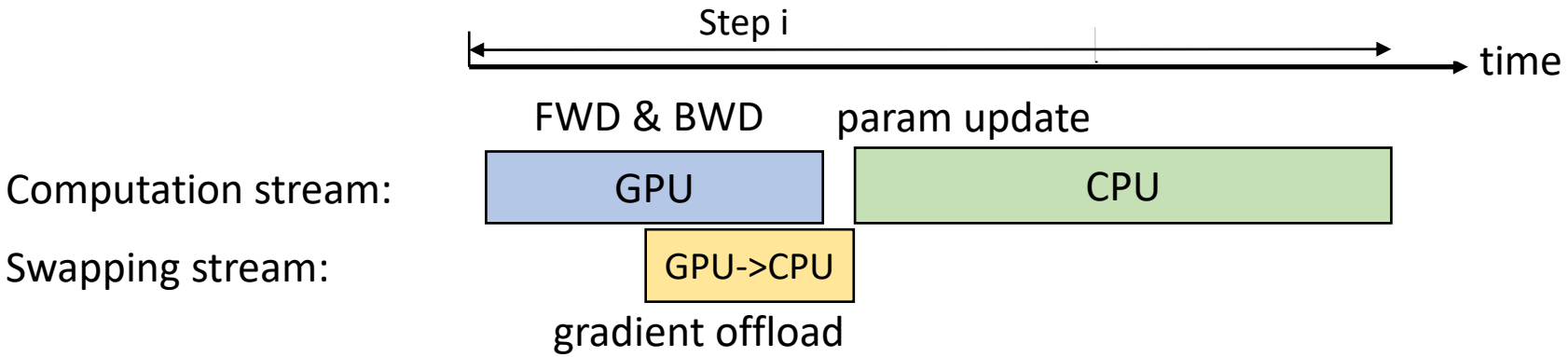


ZeRO-Offload training process on a single GPU.

ZeRO-Offload Single GPU Schedule



ZeRO-Offload Single GPU Schedule



Question: What can we do if the CPU execution time is slow?

Optimized CPU Execution

- Highly parallelized CPU optimizer implementation

    1) SIMD vector instruction for fully exploiting the hardware parallelism supported on CPU architectures.

Optimized CPU Execution

- Highly parallelized CPU optimizer implementation

    1) SIMD vector instruction for fully exploiting the hardware parallelism supported on CPU architectures.

    2) Loop unrolling to increase instruction level parallelism.

Optimized CPU Execution

- Highly parallelized CPU optimizer implementation

    1) SIMD vector instruction for fully exploiting the hardware parallelism supported on CPU architectures.

    2) Loop unrolling to increase instruction level parallelism.

    3) OMP multithreading for effective utilization of multiple cores and threads on the CPU in parallel.

Optimized CPU Execution

- Highly parallelized CPU optimizer implementation

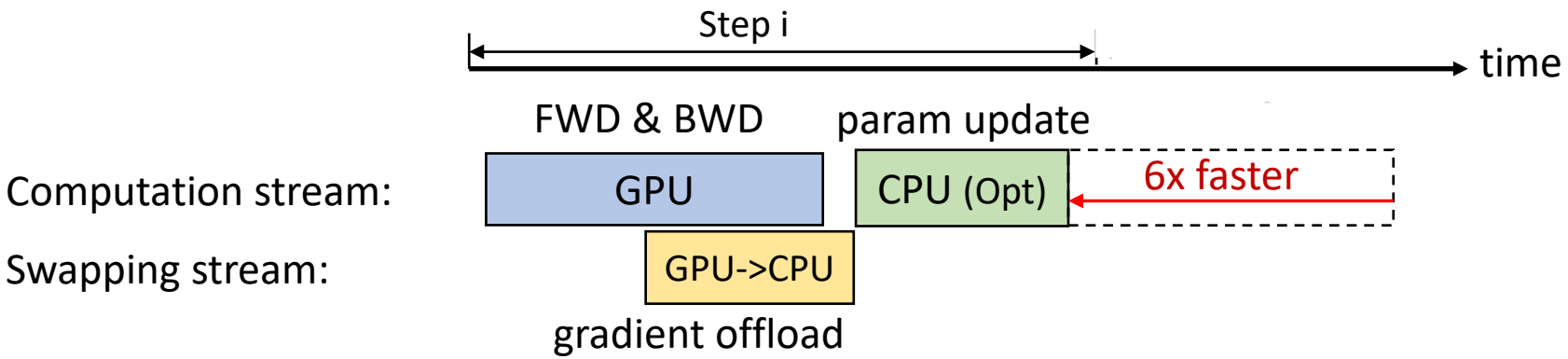
    1) SIMD vector instruction for fully exploiting the hardware parallelism supported on CPU architectures.

    2) Loop unrolling to increase instruction level parallelism.

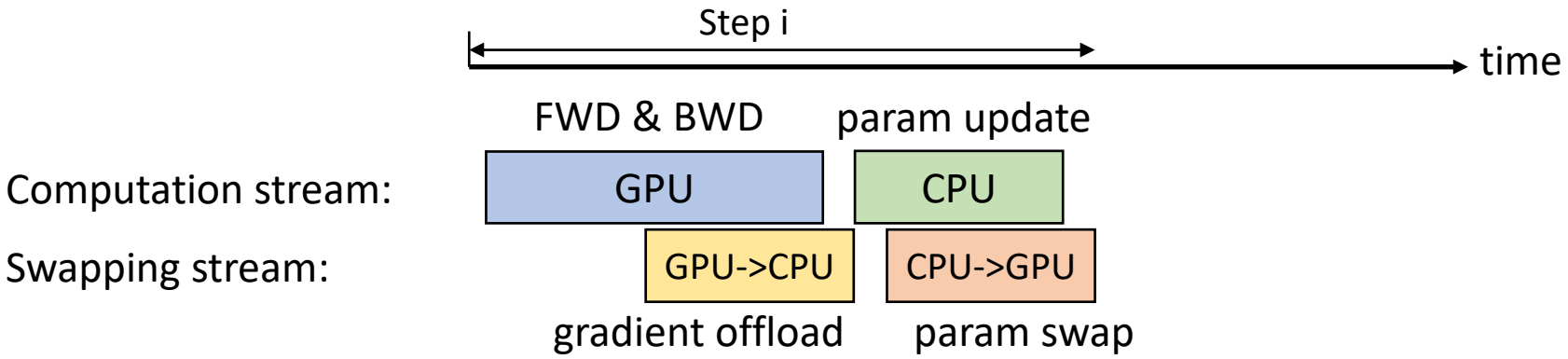
    3) OMP multithreading for effective utilization of multiple cores and threads on the CPU in parallel.

	Adam Optimizer: Parameter Update Latency (seconds)		
Model Size (B)	PyTorch-CPU	Optimized-CPUAdam (now in PyTorch)	PyTorch-GPU
1	1.39	0.22	0.10
2	2.75	0.51	0.26
4	5.71	1.03	0.64
8	11.93	2.41	0.87
10	14.00	2.57	1.00

ZeRO-Offload Single GPU Schedule

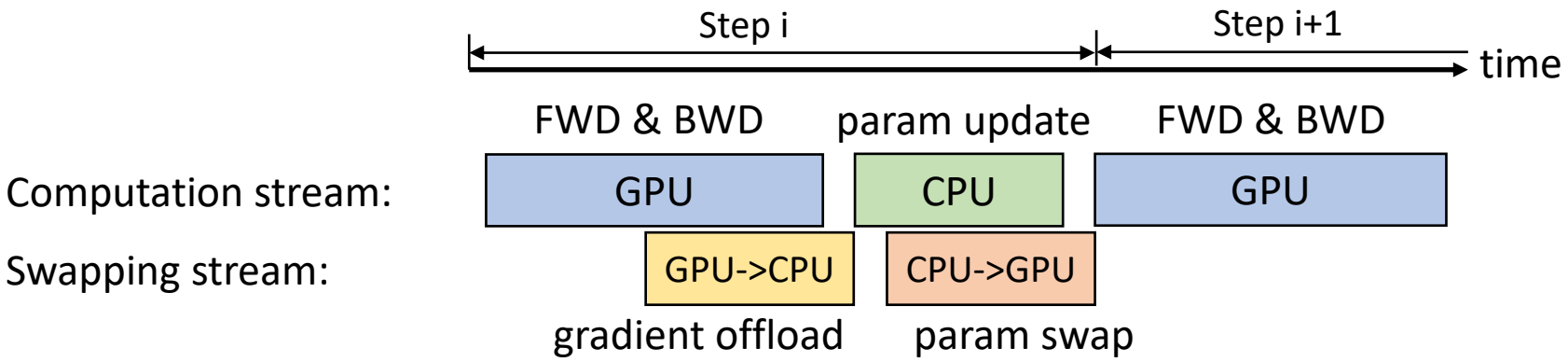


ZeRO-Offload Single GPU Schedule



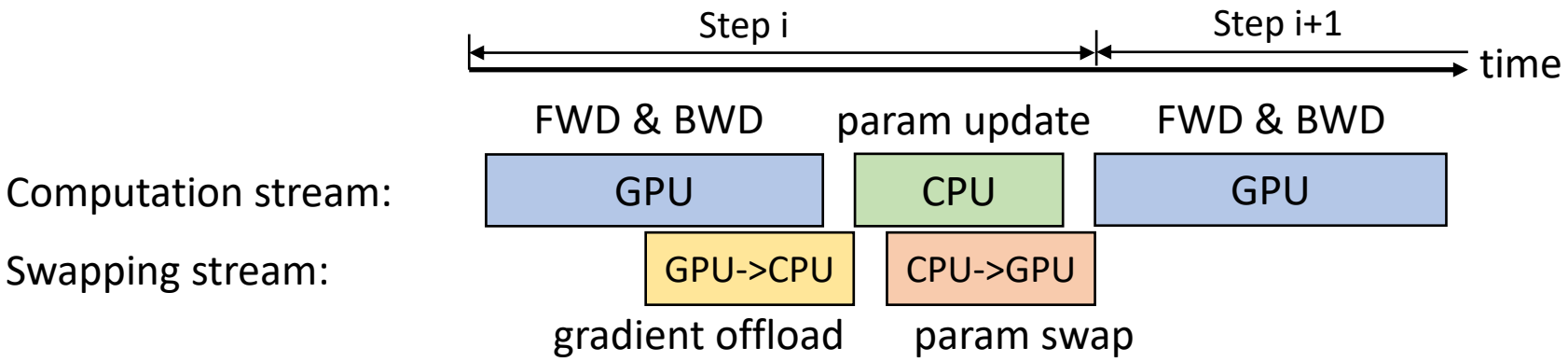
ZeRO-Offload training process on a single GPU.

ZeRO-Offload Single GPU Schedule



ZeRO-Offload training process on a single GPU.

ZeRO-Offload Single GPU Schedule



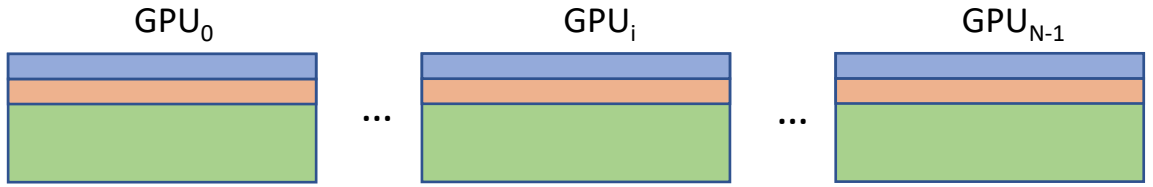
ZeRO-Offload training process on a single GPU.

Minimal impact to critical path length by hiding offloading latency and with highly parallelized CPU optimizer

How to Scale to Multi-GPUs?

- Redundancy across data parallel process

Baseline



Memory consumption		Comm Volume
Formulation	Specific Example K=12 Φ=10B N_d=4	
(2+2+K) * Φ	200GB	1x

OOM

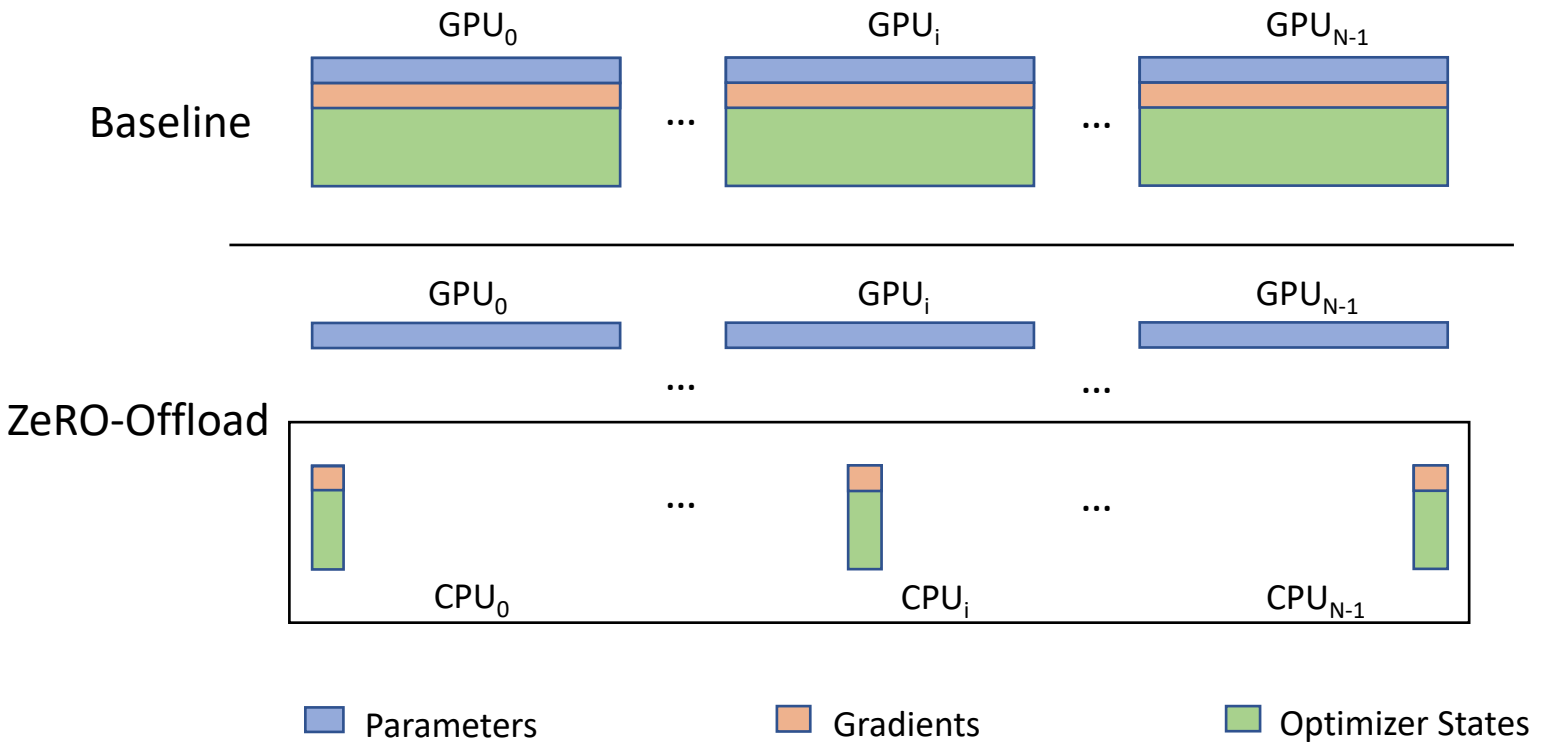
C1: Would offloading model states or computation to CPU in combination with data parallelism result in replication of CPU memory usage and compute?

C2: Would offloading increase communication and limit throughput scalability?

Parameters Gradients Optimizer States

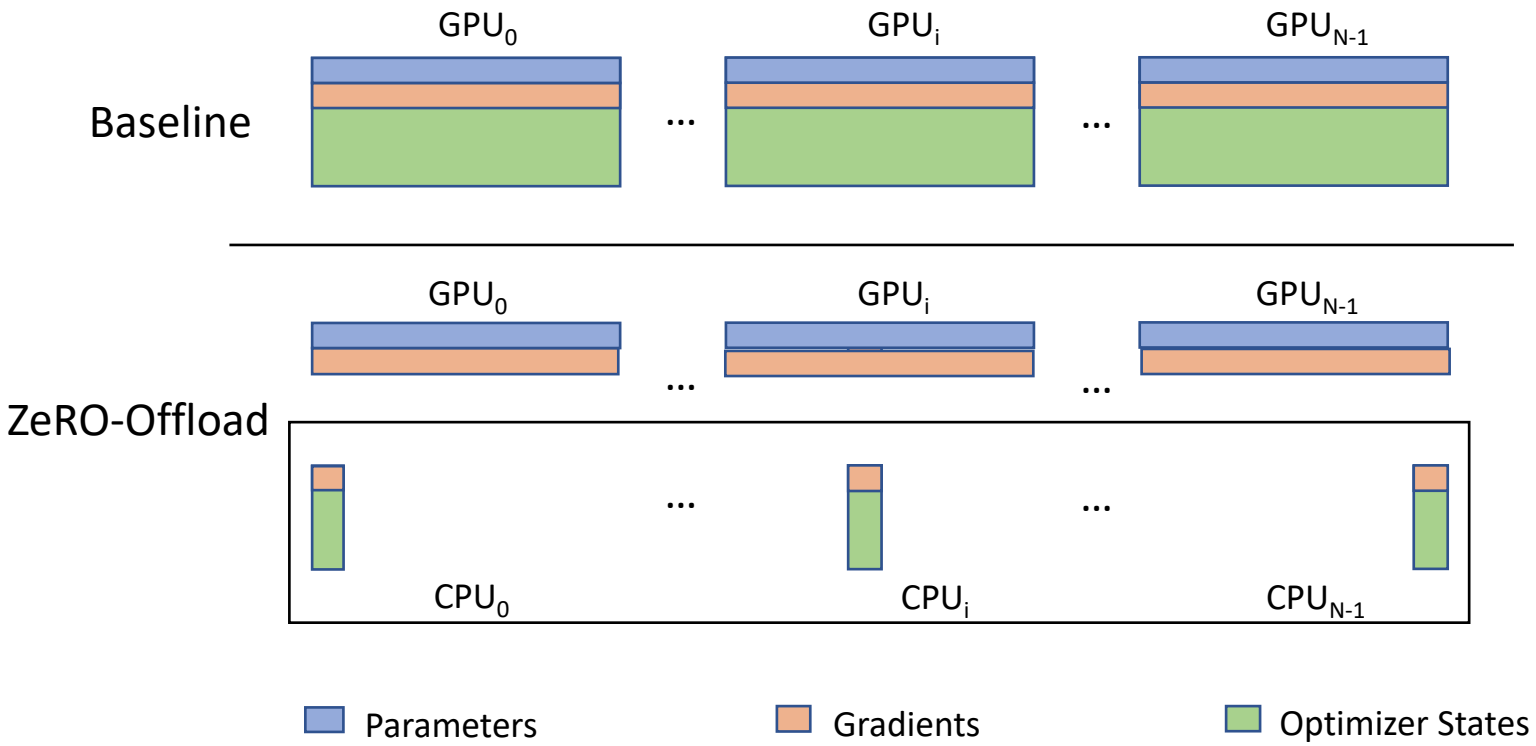
ZeRO-Offload Scaling to Multi-GPUs

- Partitioning **optimizer states and gradients**



ZeRO-Offload Scaling to Multi-GPUs

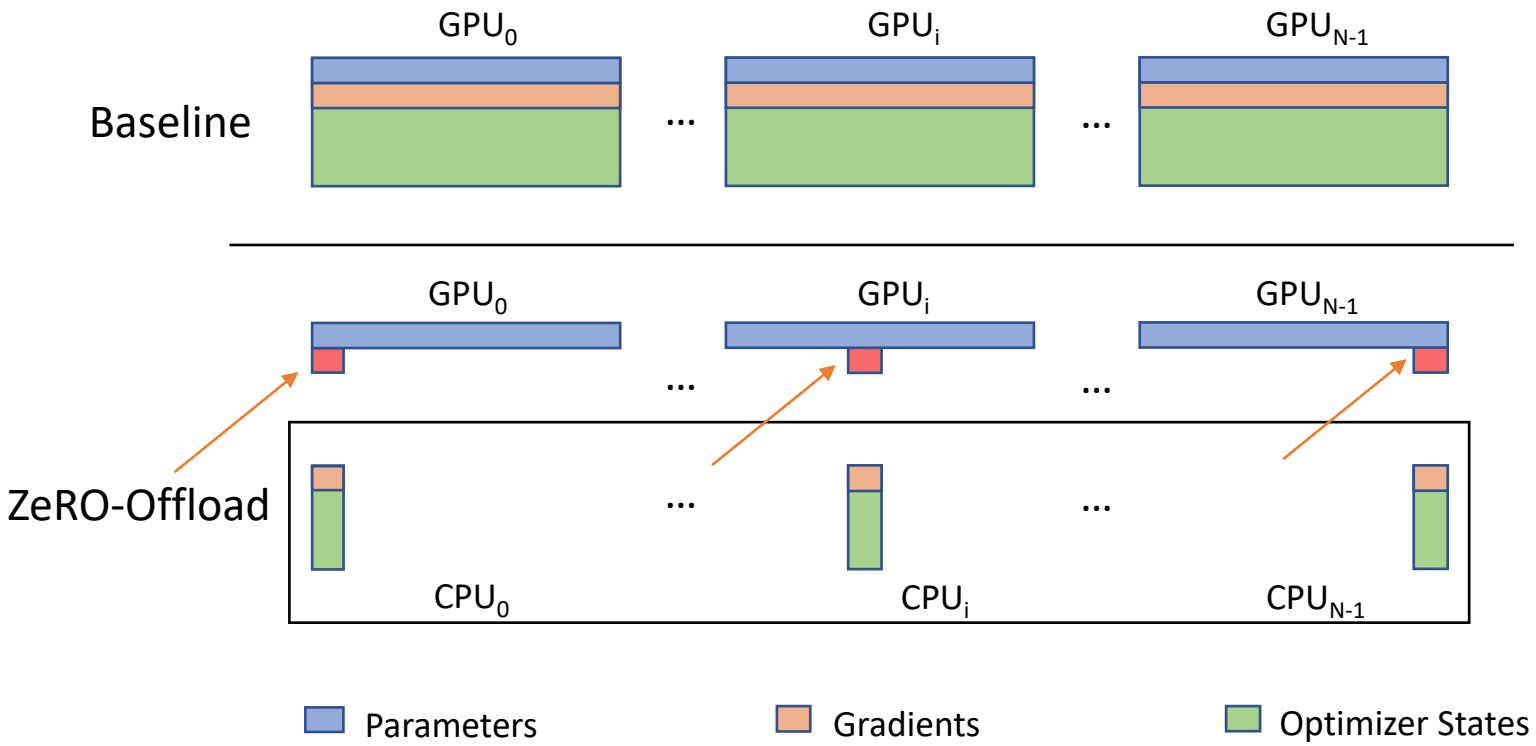
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data

ZeRO-Offload Scaling to Multi-GPUs

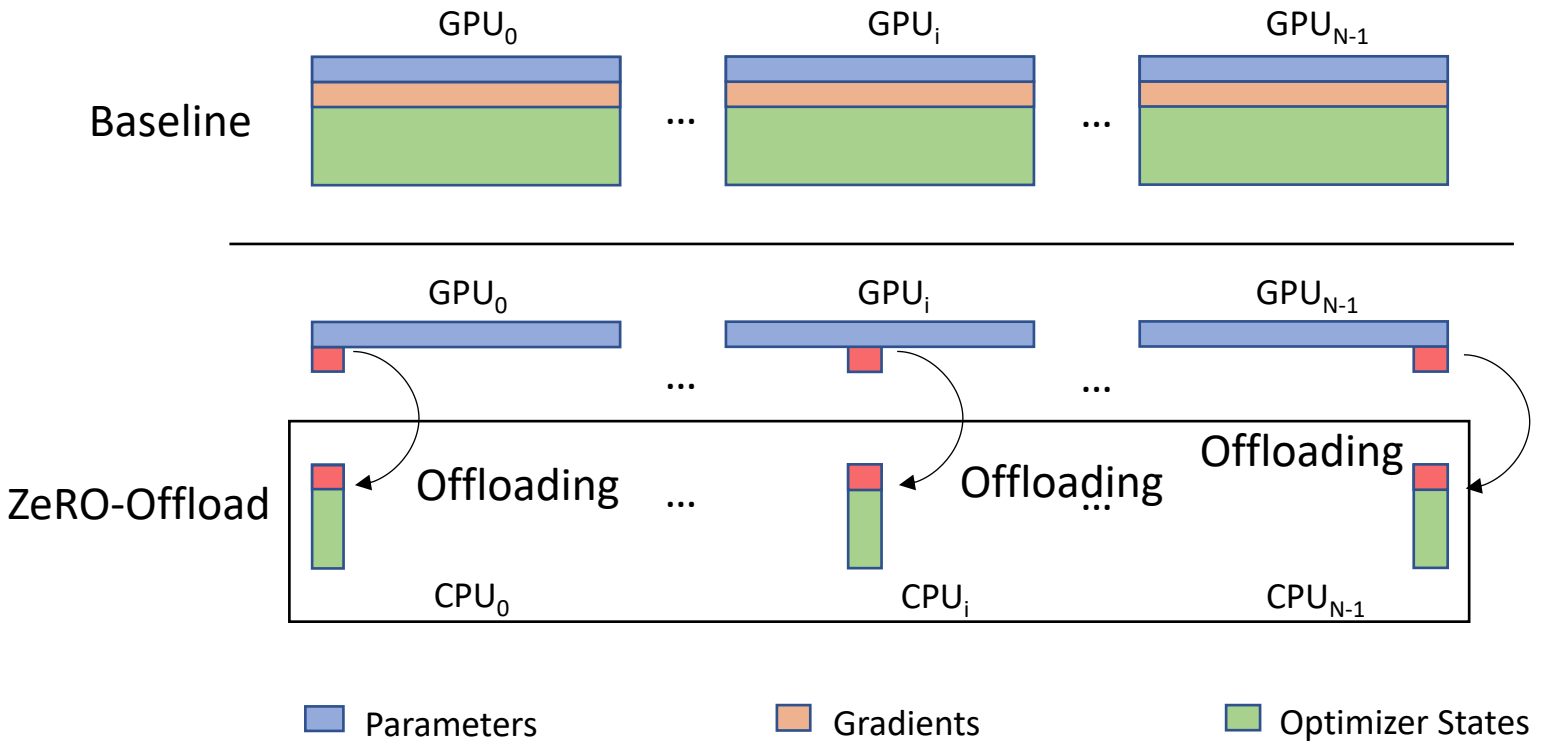
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data
2. Reduce-scatter gradients

ZeRO-Offload Scaling to Multi-GPUs

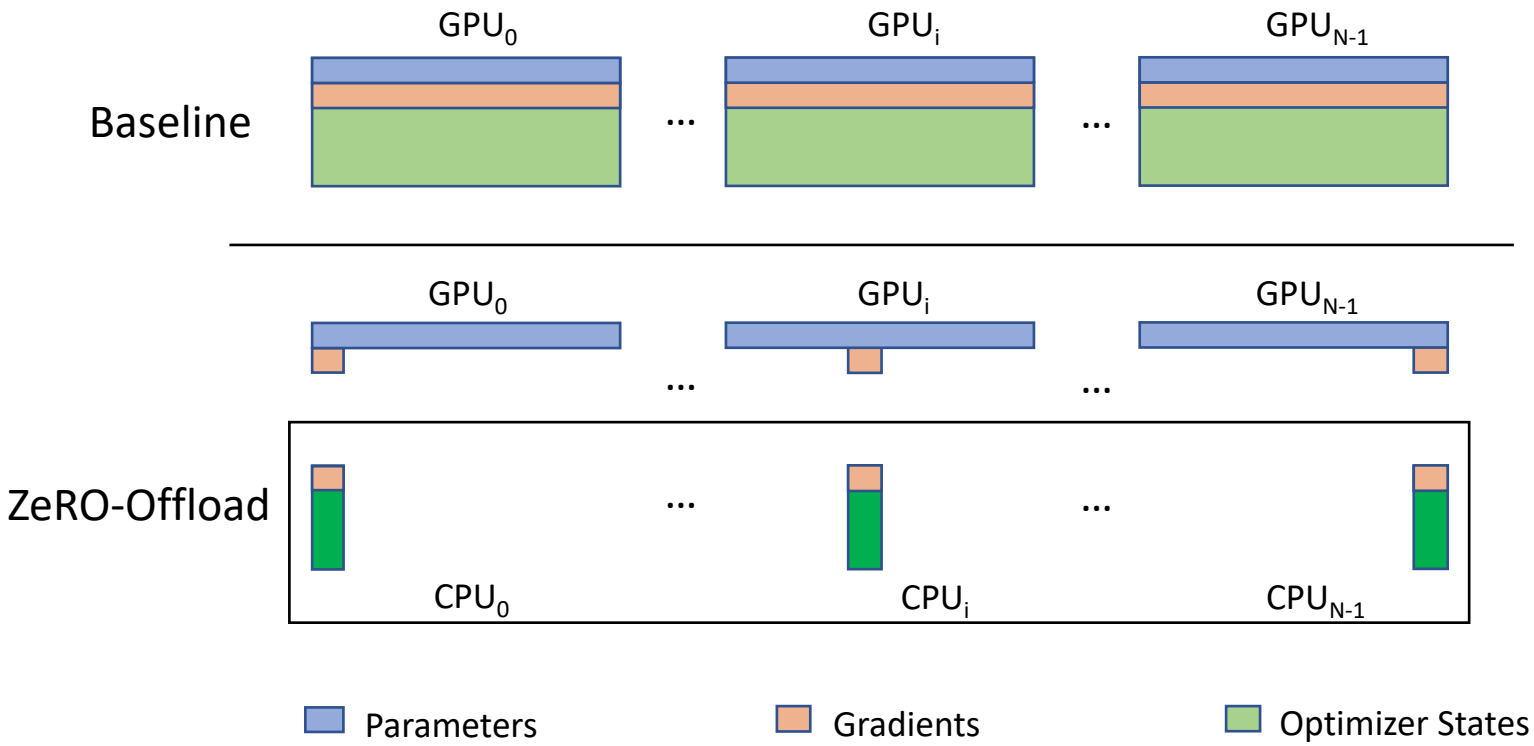
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data
2. Reduce-scatter gradients
3. Offload gradients

ZeRO-Offload Scaling to Multi-GPUs

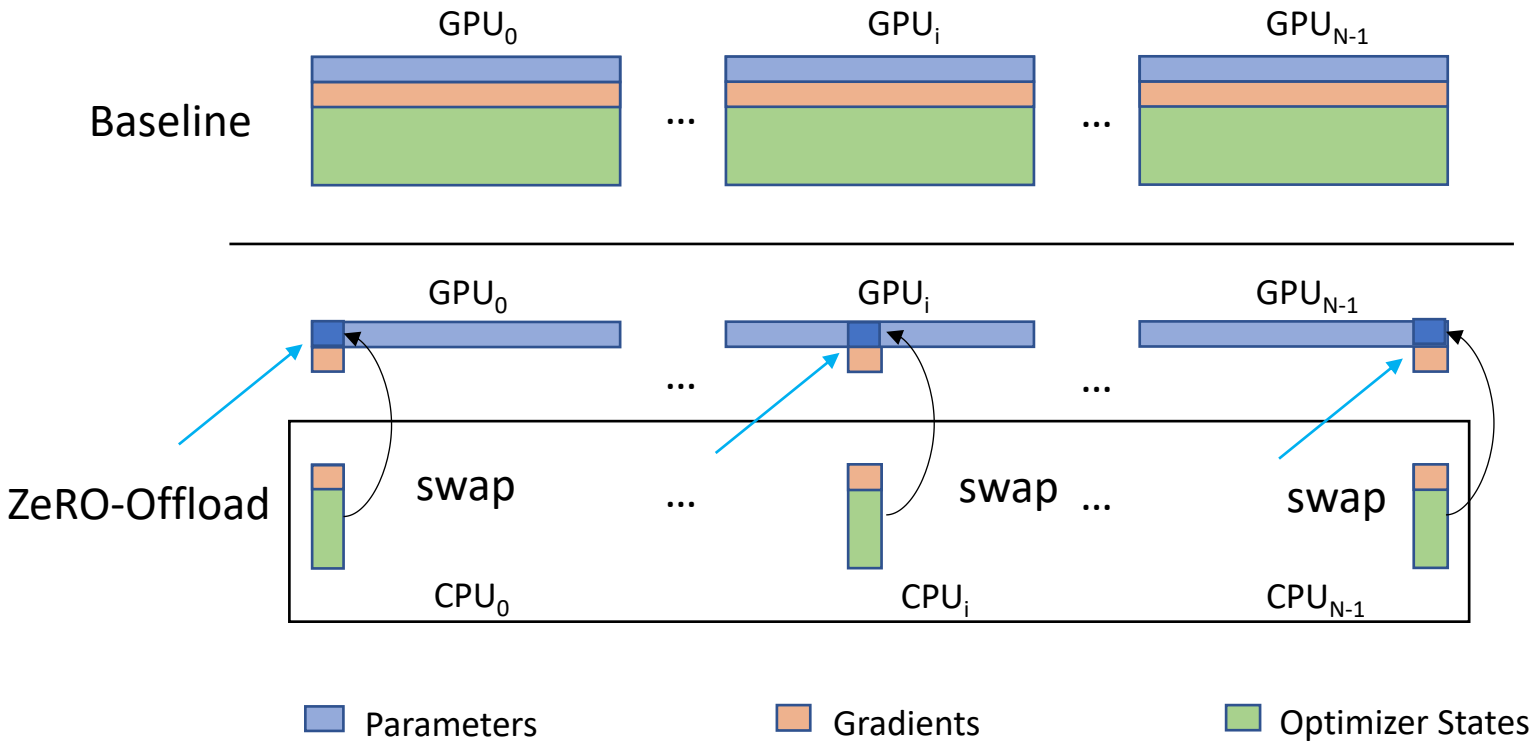
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data
2. Reduce-scatter gradients
3. Offload gradients
4. CPU parameter update

ZeRO-Offload Scaling to Multi-GPUs

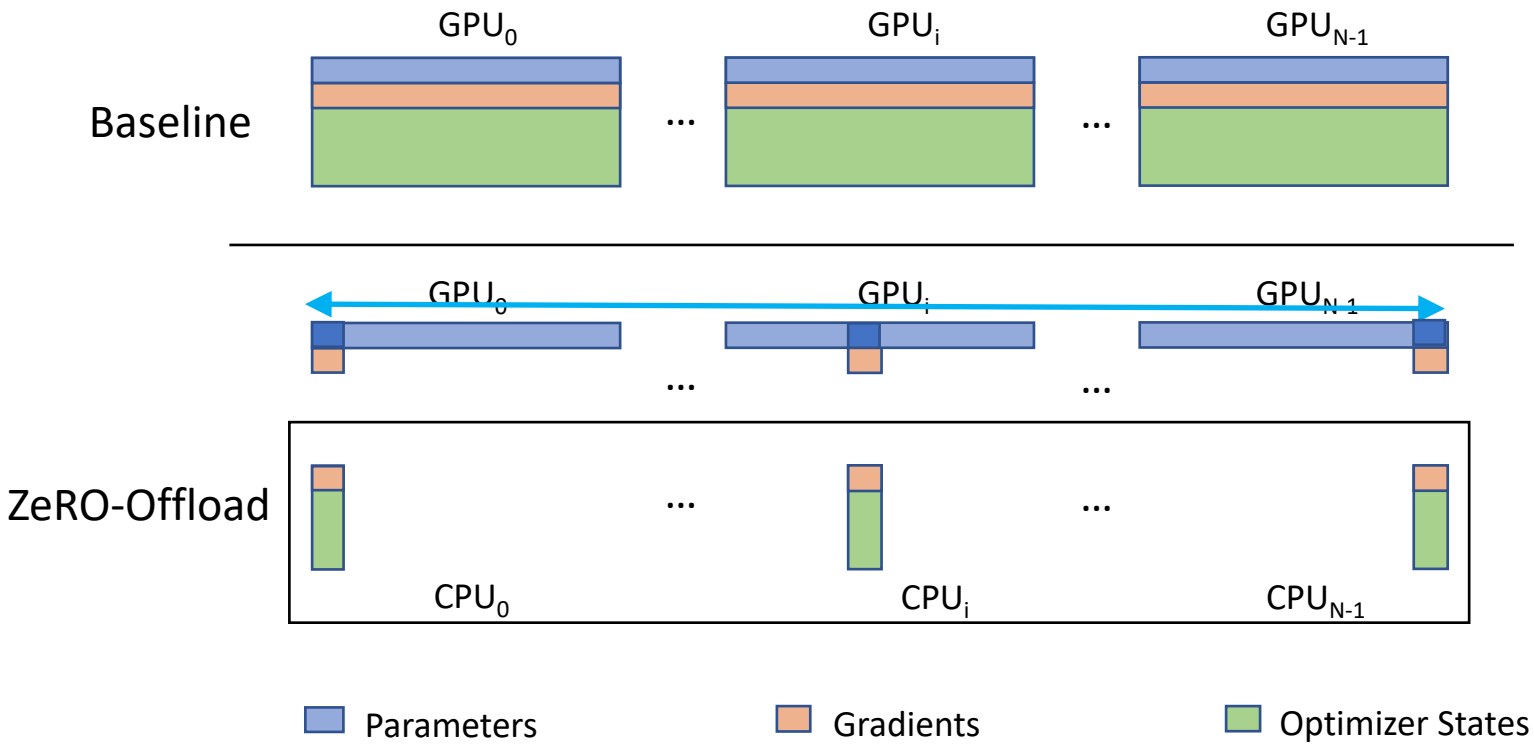
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data
2. Reduce-scatter gradients
3. Offload gradients
4. CPU parameter update
5. Each DP process swaps only the parameter it owns

ZeRO-Offload Scaling to Multi-GPUs

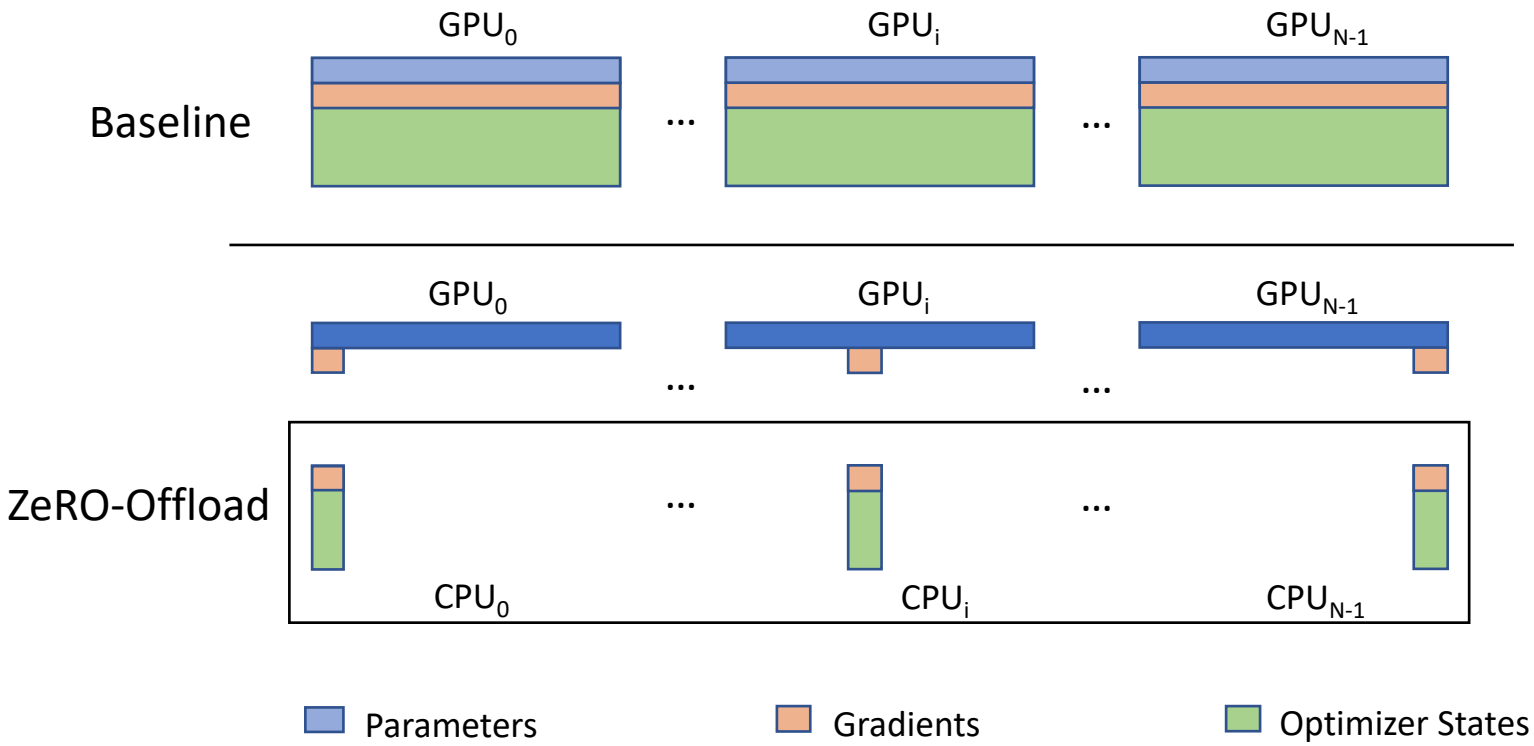
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data
2. Reduce-scatter gradients
3. Offload gradients
4. CPU parameter update
5. Each DP process swaps only the parameter it owns
6. All-gather the parameters

ZeRO-Offload Scaling to Multi-GPUs

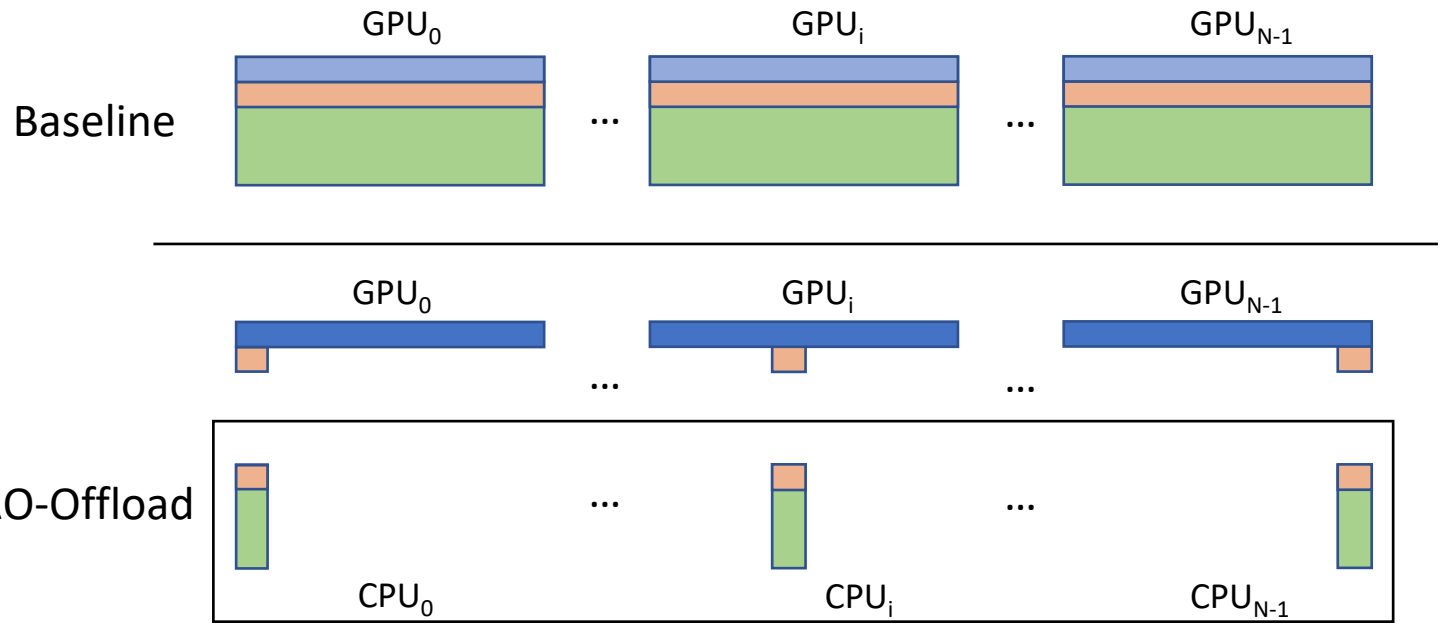
- Partitioning **optimizer states and gradients**



1. Compute gradients on different data
2. Reduce-scatter gradients
3. Offload gradients
4. CPU parameter update
5. Each DP process swaps only the parameter it owns
6. All-gather the parameters

ZeRO-Offload Scaling to Multi-GPUs

- Benefits:
  - Single copy of model states in CPU regardless of DP degree
  - No additional communication volume in between CPU and GPU



Memory consumption		Comm Volume
Formulation	Specific Example K=12 Φ=10B N_d=4	
(2+2+K) * Φ	**200GB (GPU)**	1x
$2\Phi + \frac{2 * \Phi}{N_d}$	**25GB (GPU)**	1x
(2 + K) * Φ	180GB (CPU)	N/A

OOM

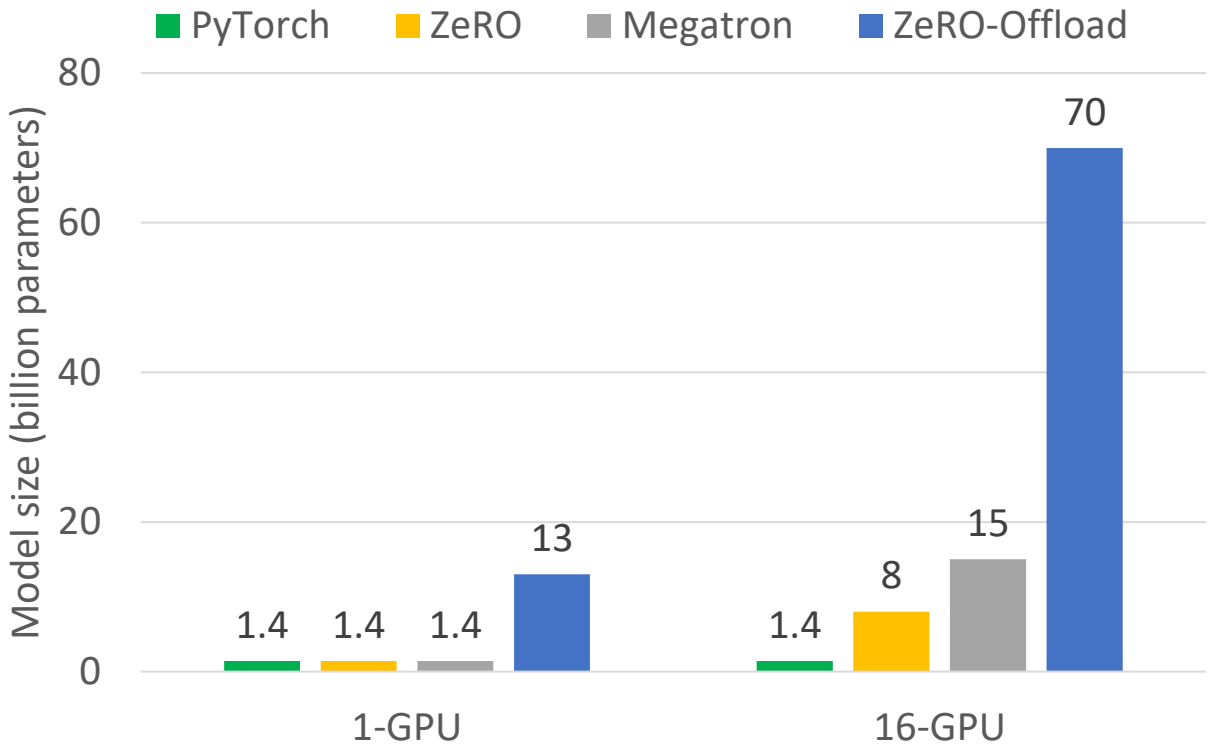
Evaluation

- Testbed

	DGX-2 node
GPU	16 NVIDIA Tesla V100 Tensor Core GPUs
GPU Memory	32GB HBM2 on each GPU
CPU	2 Intel Xeon Platinum 8168 Processors
CPU Memory	1.5TB 2666MHz DDR4
CPU cache	L1, L2, and L3 are 32K, 1M, and 33M, respectively
PCIe	bidirectional 32 GBps PCIe

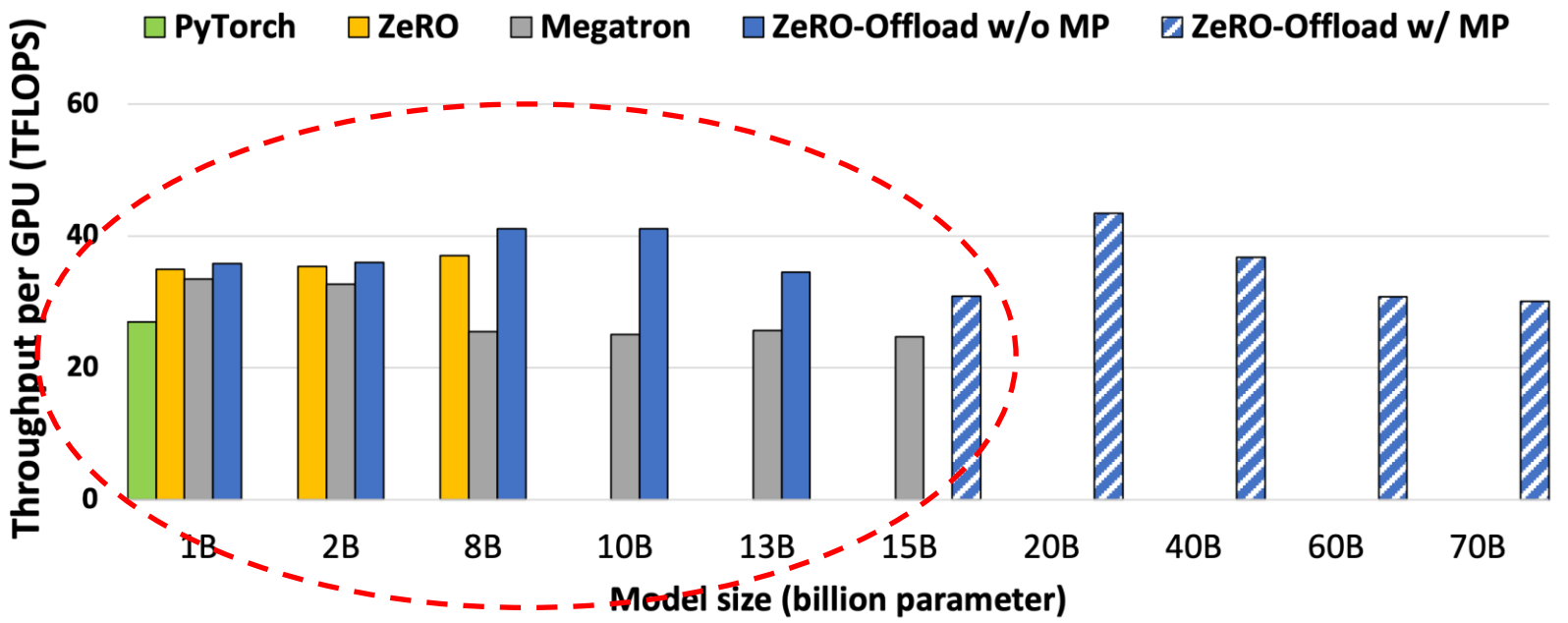
- Baselines
  1) Pytorch DDP: distributed data parallelism
  2) Megatron: model parallelism
  3) ZeRO: extended data parallelism by eliminating memory redundancies across multiple GPUs

Model Scale



ZeRO-Offload enables 13B model training on a single GPU, and easily enables training of up to 70B parameter with 16 GPUs.

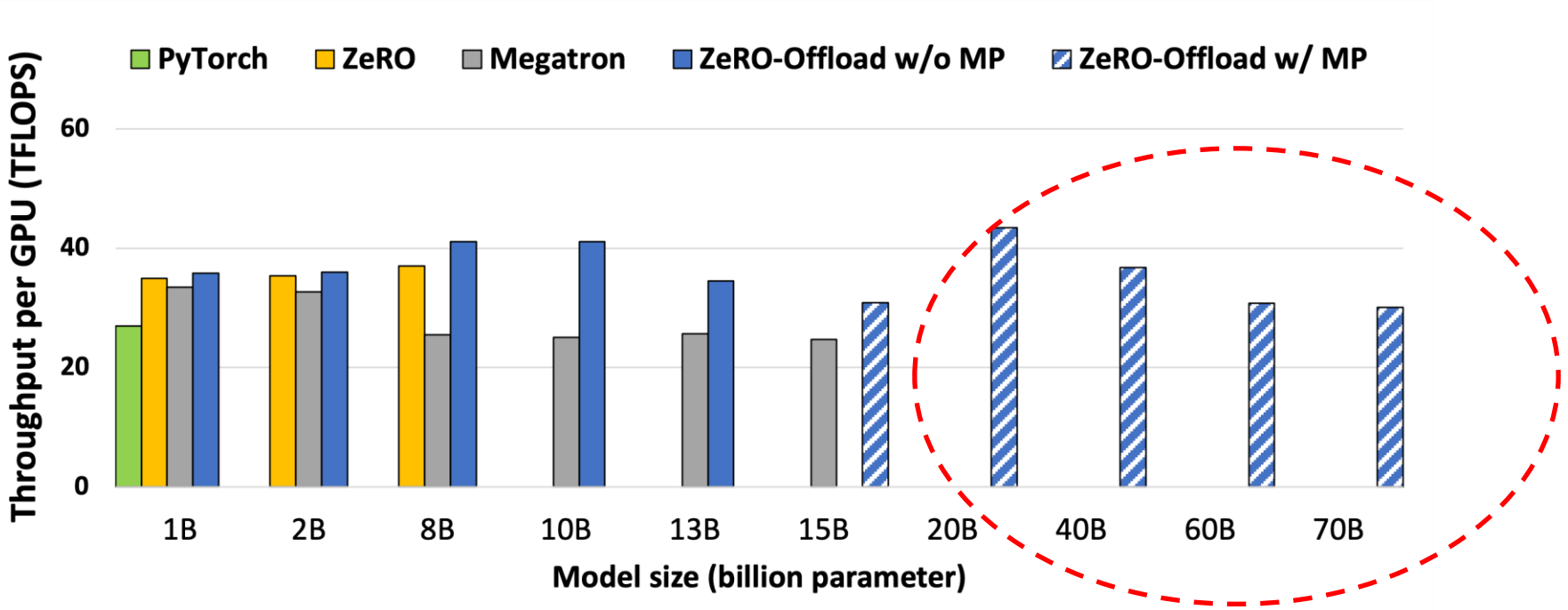
Training Throughput – Multiple GPUs



For 1B to 15B models, ZeRO-Offload achieves the highest throughput compared with PyTorch, ZeRO, and Megatron.

Single DGX-2 node (x16 V100-32GB)

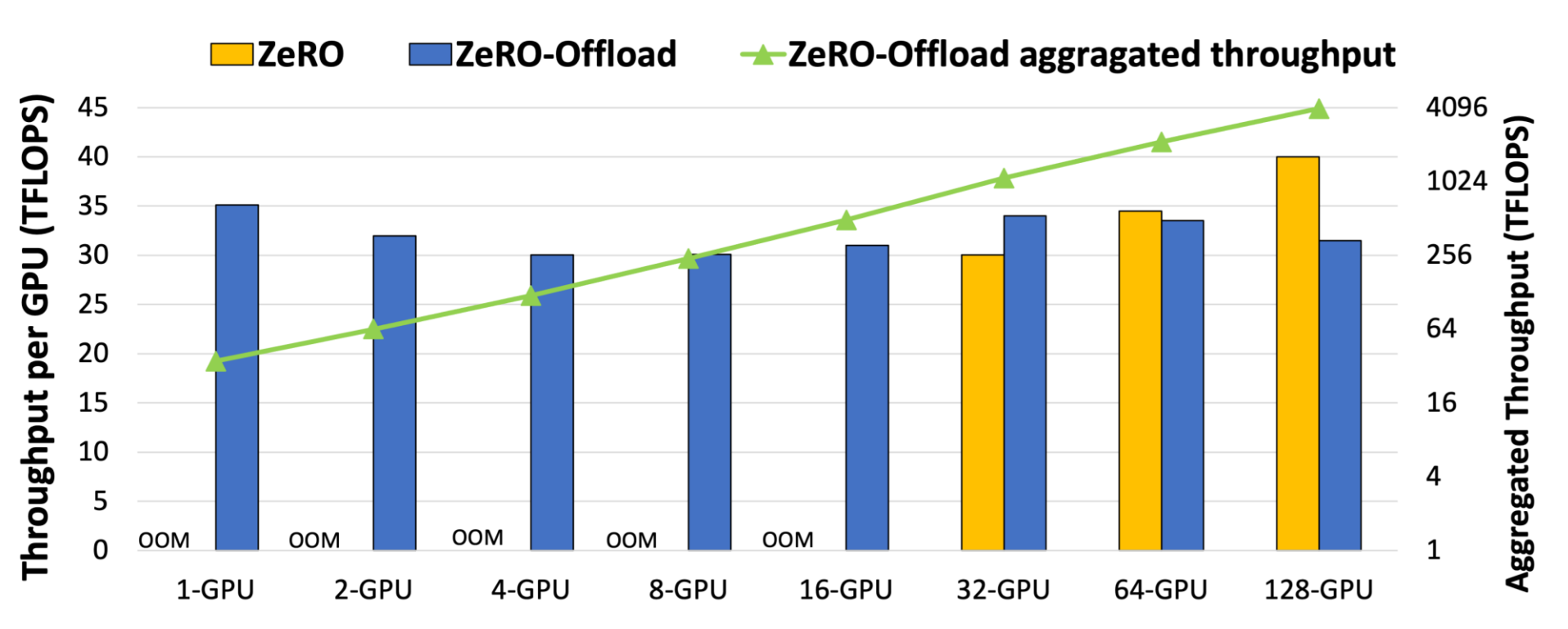
Training Throughput – Multiple GPUs



Combined with model parallelism, ZeRO-Offload enables training up to 70B parameter models with more than 30 TFLOPS throughput per GPU.

Single DGX-2 node (x16 V100-32GB)

Throughput Scalability



ZeRO-Offload achieves near perfect linear speedup in terms of aggregated throughput running at over 30 TFlops per GPU.

Questions?