ZeRO-Offload: Democratizing Billion-Scale Model Training

Jie Ren[*] Samyam Rajbhandari[†] Reza Yazdani Aminabadi[†] Olatunji Ruwase[†]

Shuangyan Yang[*] Minjia Zhang[†] Dong Li[*] Yuxiong He[†]

[*] University of California, Merced [†] Microsoft

Large DNN Workloads Are Hungry for Memory

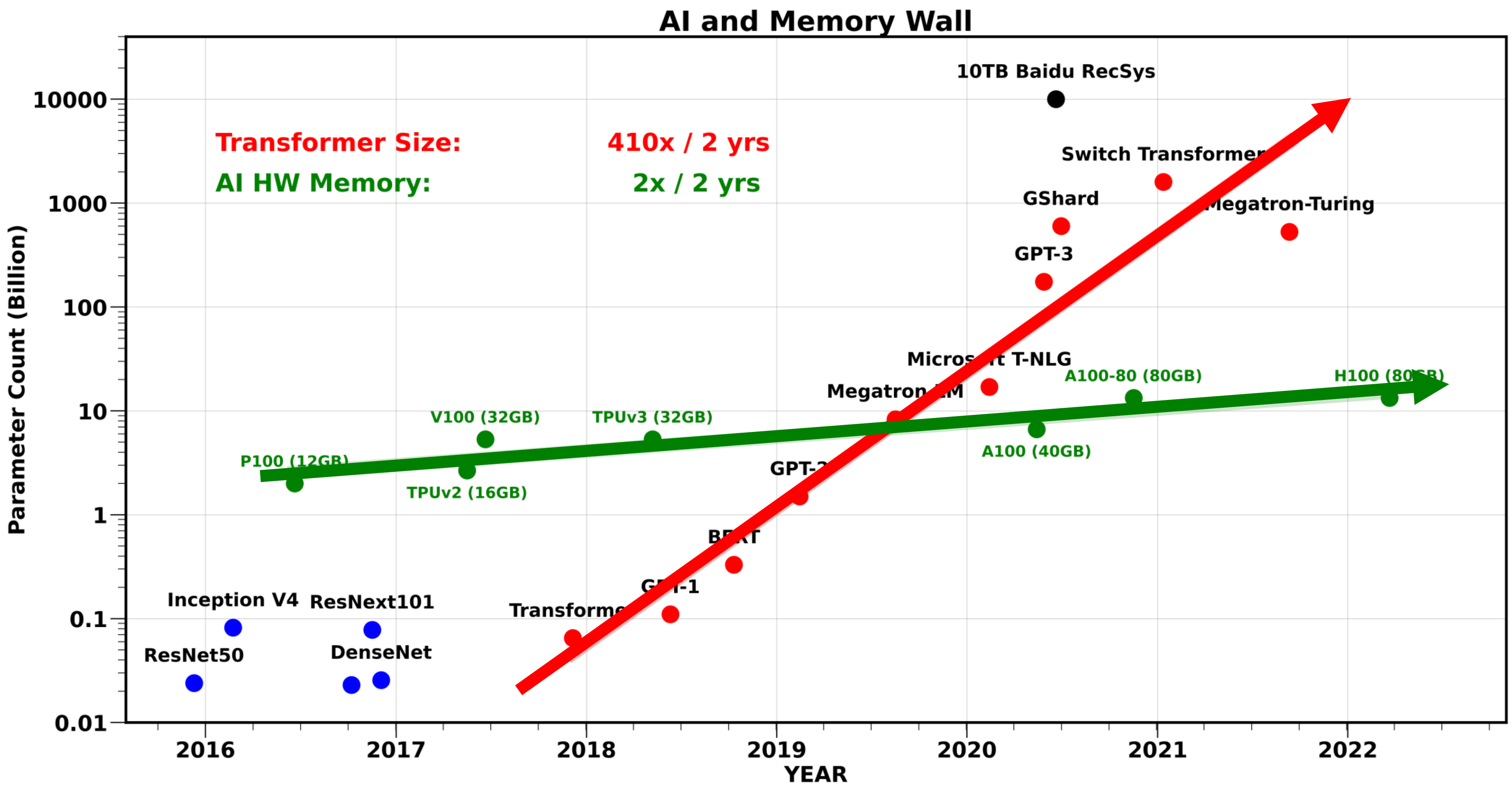
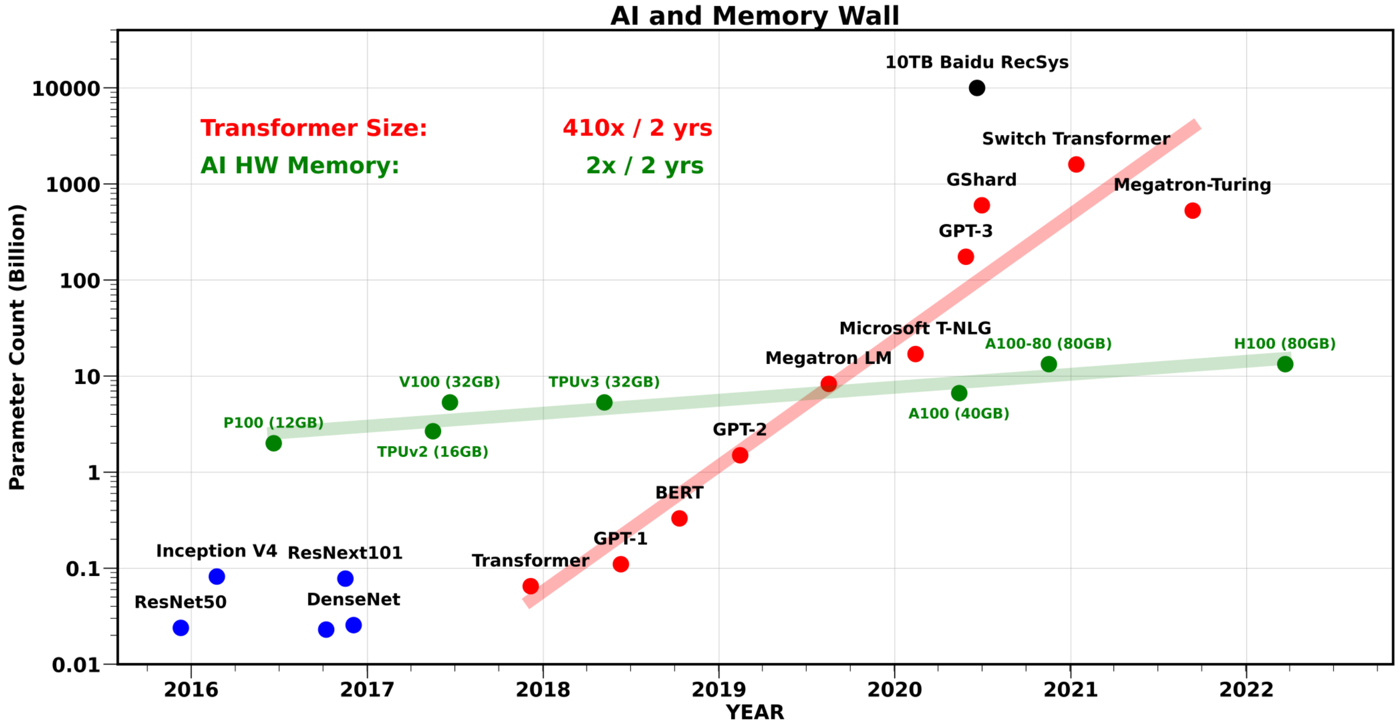
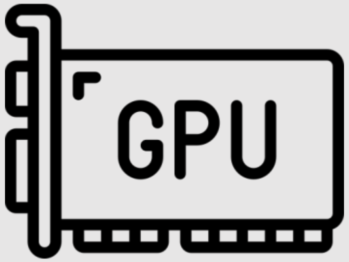


Photo credit: https://github.com/amirgholami/ai_and_memory_wall

Large DNN Workloads Are Hungry for Memory

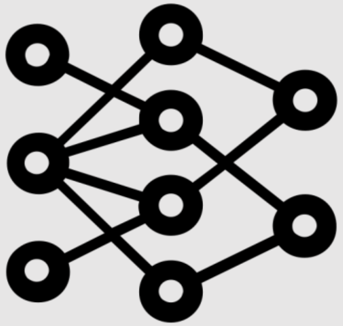




Tens of GBs

On-board Memory

vs.



TBs

DNN Model Size

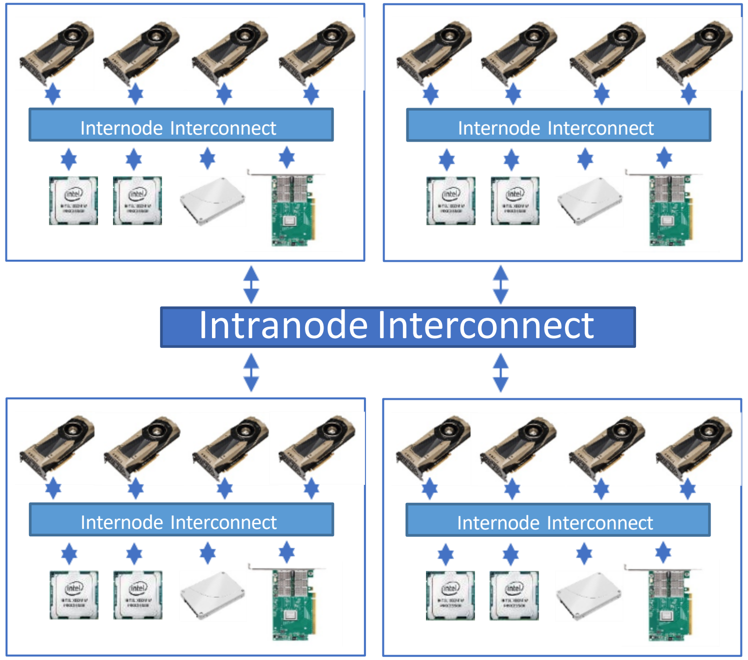
Photo credit: https://github.com/amirgholami/ai_and_memory_wall

Billon-Scale Model Training - Scale Out Large Model Training

- Model parallelism (Megatron-LM)
  - Partition the model states vertically across multiple GPUs.
- Pipeline parallelism (PipeDream, SOSP'19)

Require having enough GPU devices!

- ZeRO: Zero Redundancy Optimizer (ZeRO, SC'20)
  - Split the training batch across multiple GPUs without model states duplication.

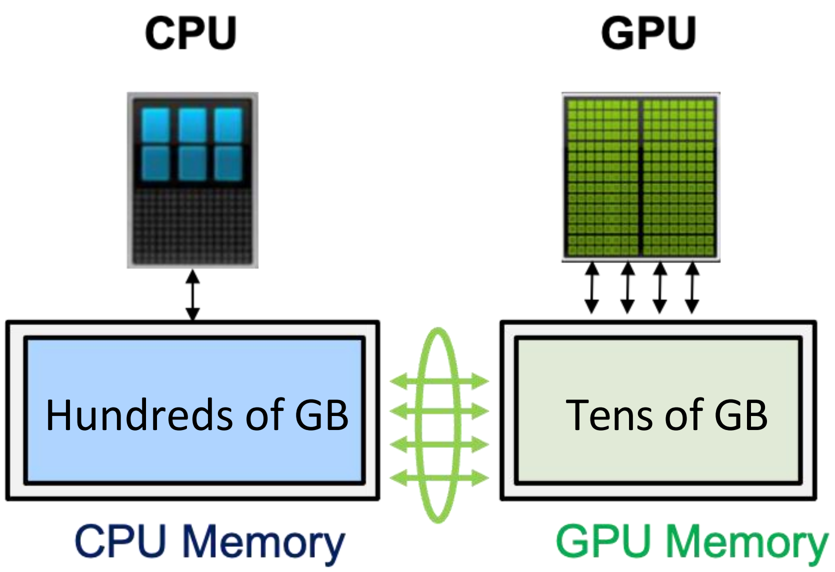


Distributed GPU Cluster

Billon-Scale Model Training - Scale Up Large Model Training

- Heterogeneous DL training (SwapAdvisor, ASPLOS'20; Sentinel, HPCA'21; L2L)
  - Offload tensors from GPU memory to CPU memory when tensors are not used in computation.
  - Prefetch tensors from CPU memory to GPU memory before computation happens.

Only use CPU memory but not CPU computation;
Designed for a single GPU



ZeRO-Offload: Democratizing Billion-Scale Model Training

Efficiency

- Enable 13B-parameter model training on a single NVIDIA V100 GPU at 40 TFLOPS.

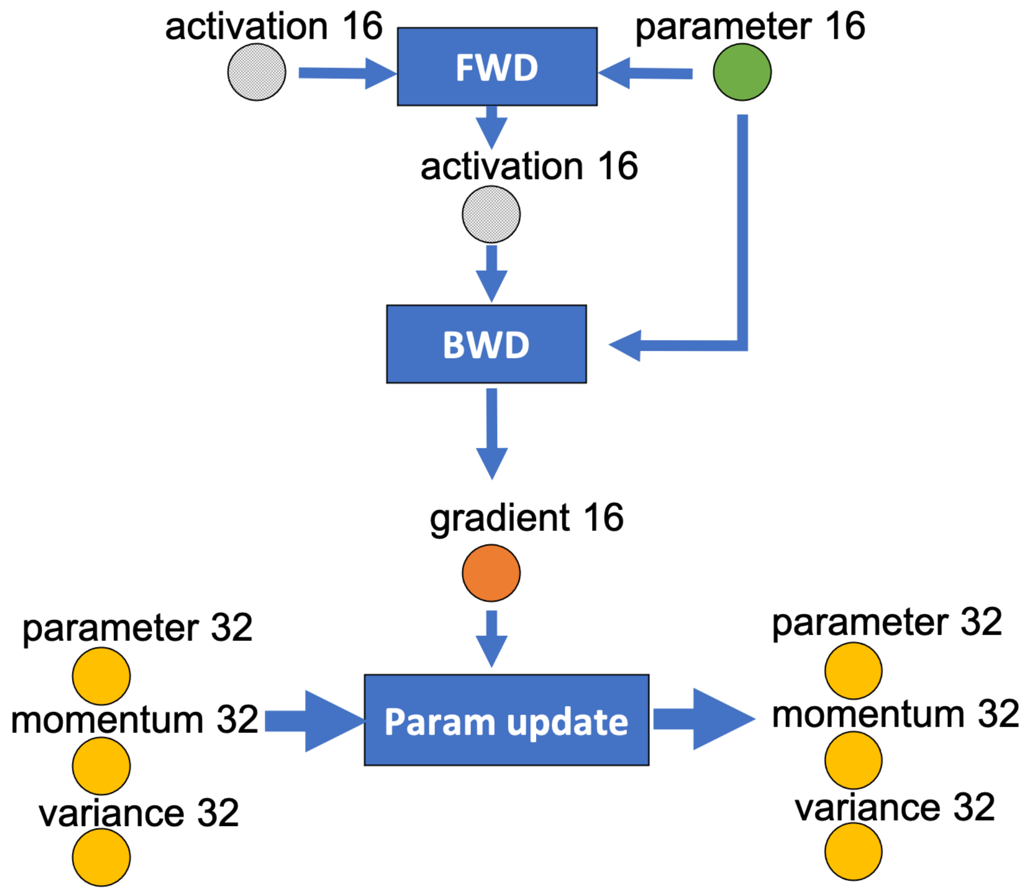
Scalability

- Achieve near perfect linear speedup with multiple GPUs.

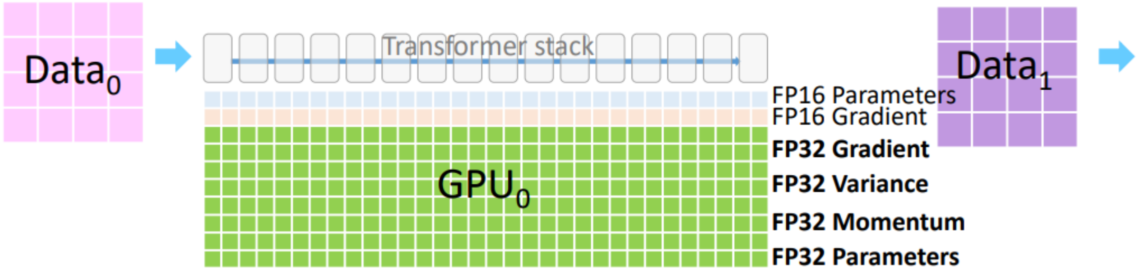
Usability

- Require no model refactoring.

Mixed Precision Training

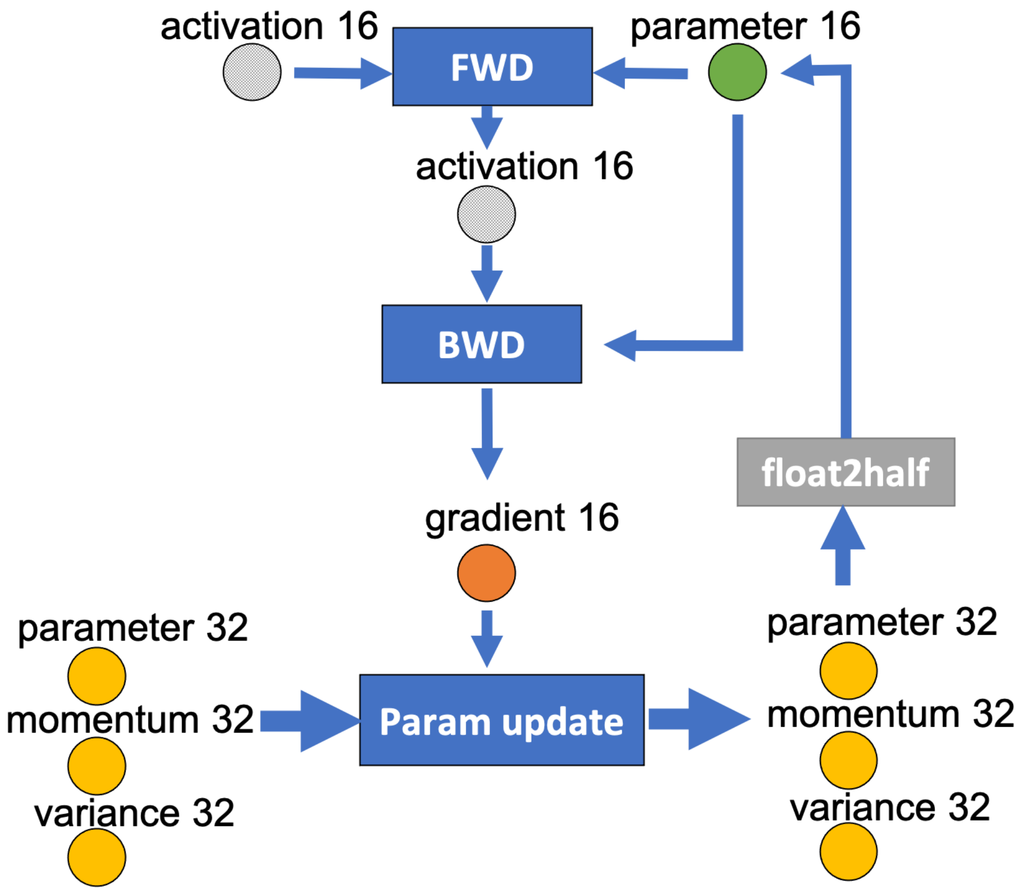


Mixed precision training iteration for a layer.

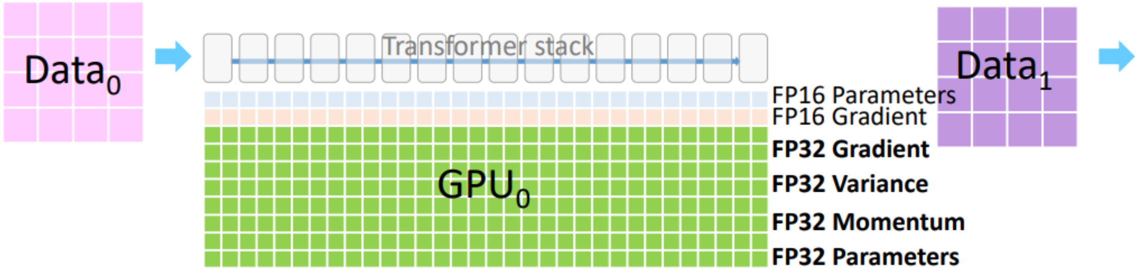


- FP16 parameter
- FP16 Gradients
- FP32 Optimizer States
    - Gradients, Variance, Momentum, Parameters

Mixed Precision Training



Mixed precision training iteration for a layer.



- FP16 parameter
- FP16 Gradients
- FP32 Optimizer States
  - Gradients, Variance, Momentum, Parameters

Offload Strategy

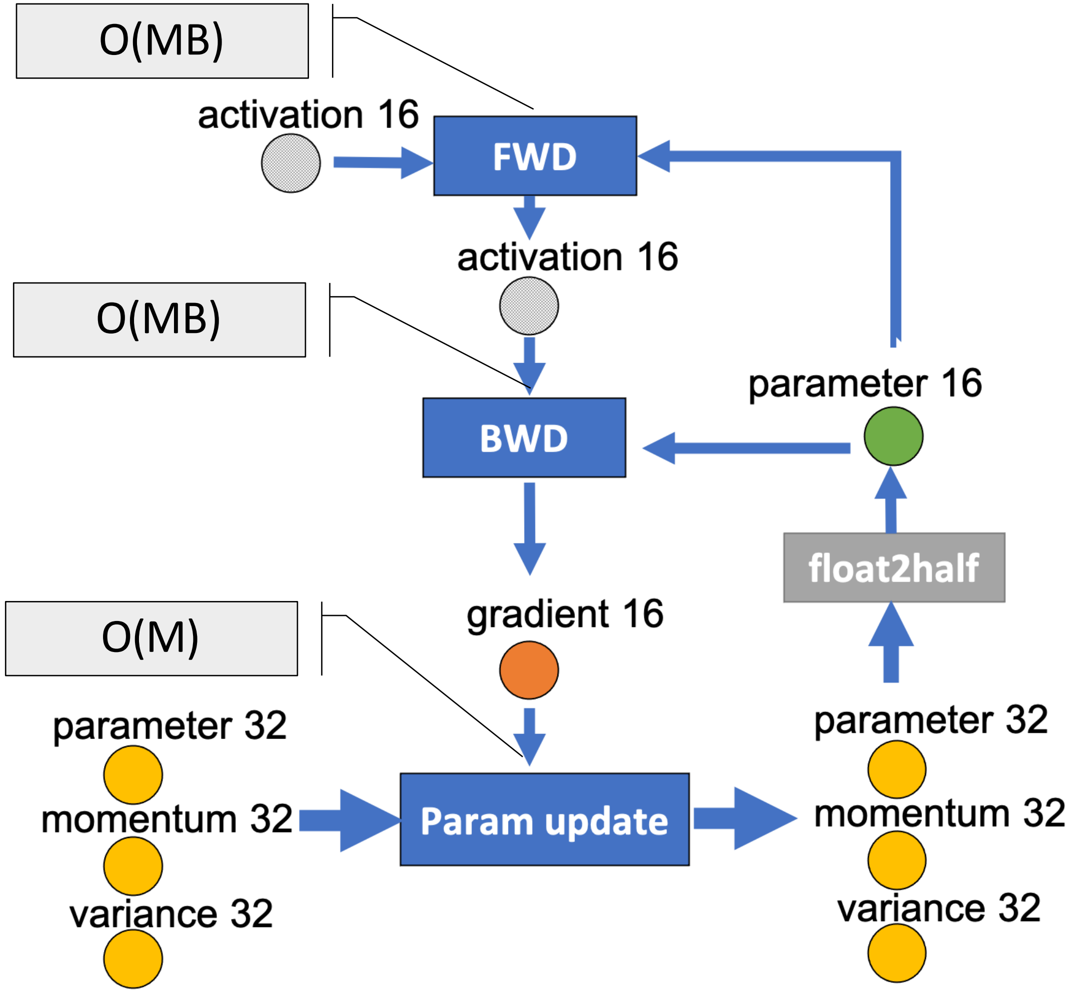
- ZeRO-Offload partitions the dataflow graph with:

  i. Few computation on CPU

  ii. Minimization of communication volume

  iii. Maximization of memory saving while achieving minimum communication volume

Limiting CPU Computation



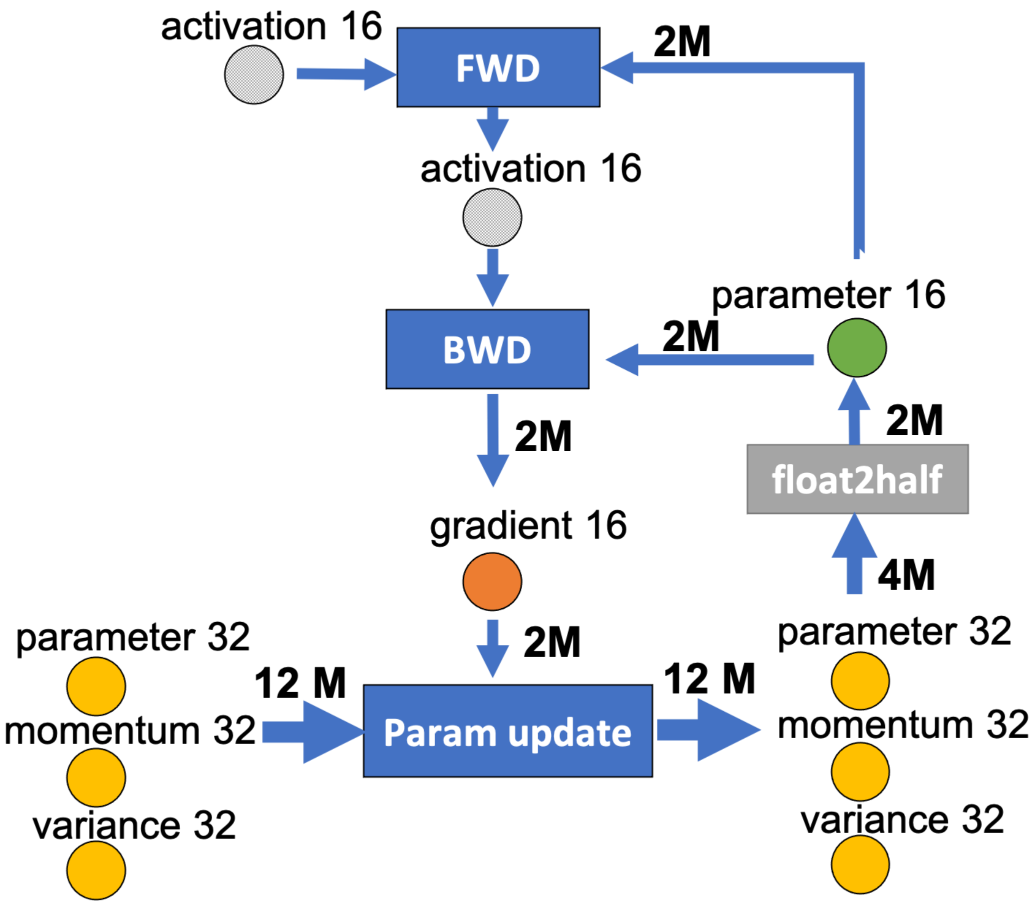
Computational Complexity

M: #Parameters

B: Batch size

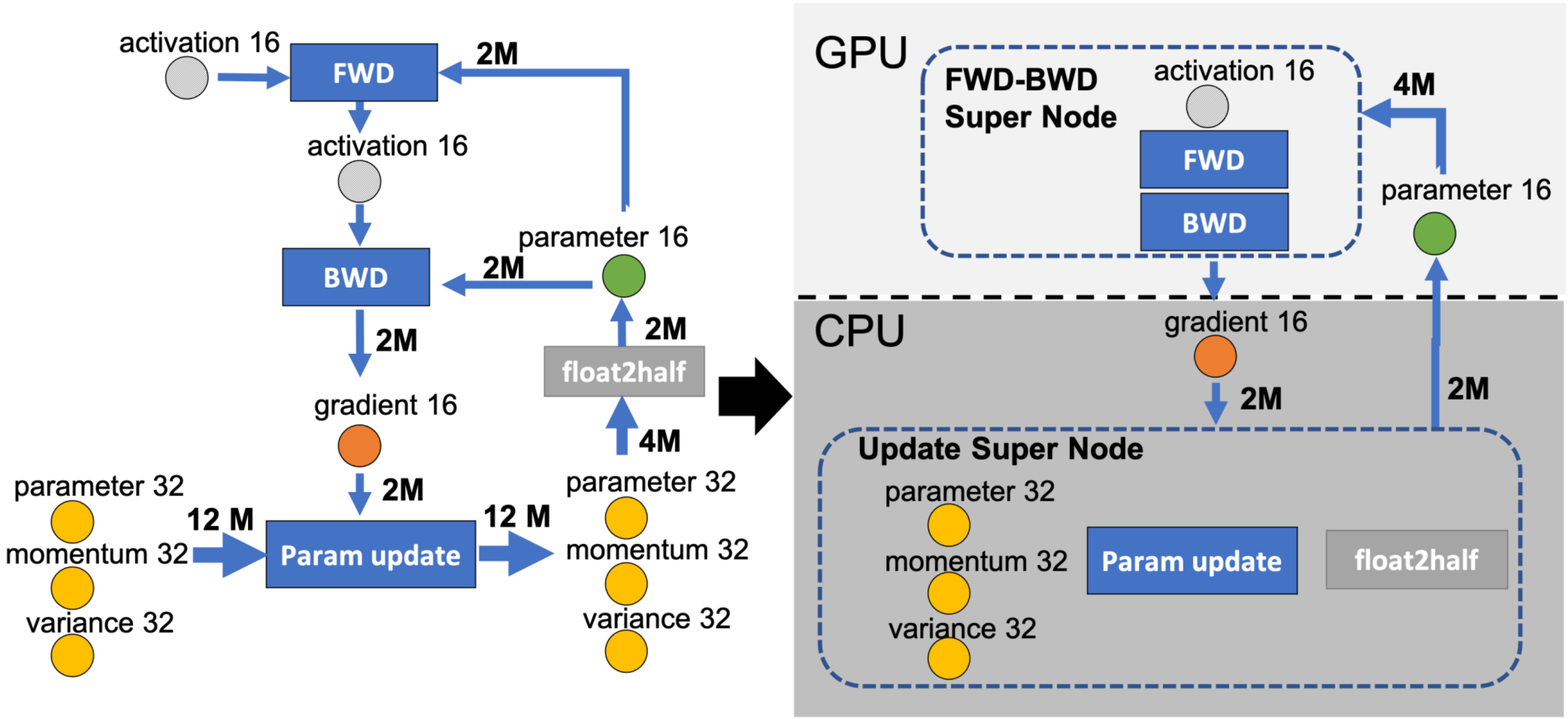
The dataflow of fully connected neural networks with M parameters.

Minimizing Communication Volume



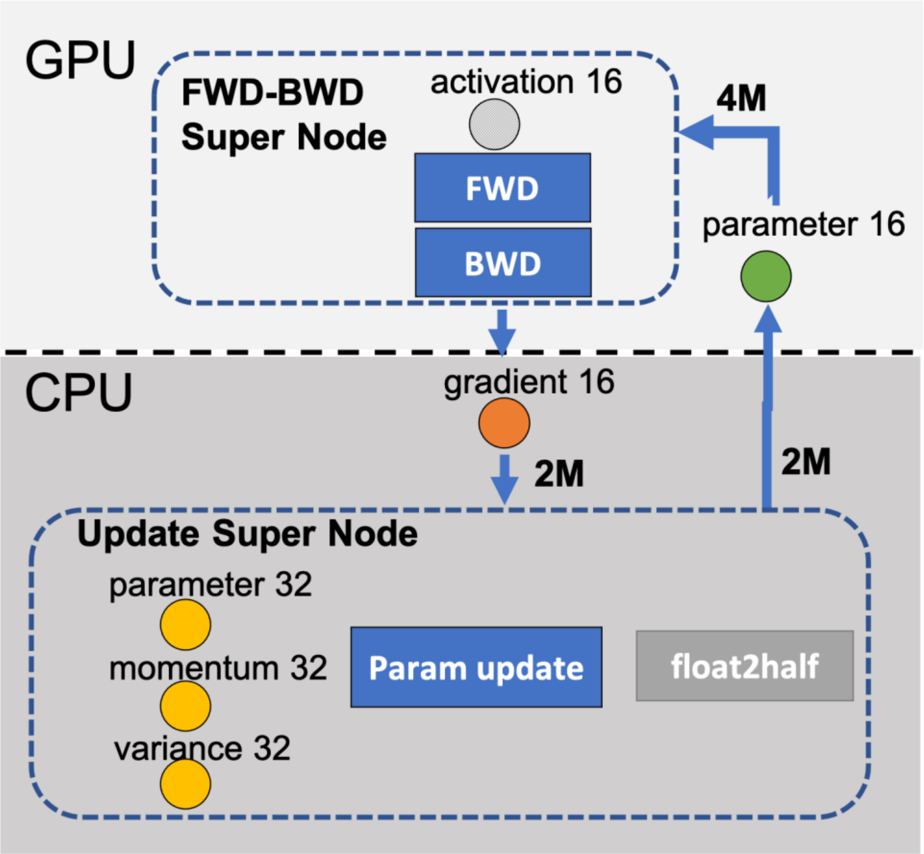
The dataflow of fully connected neural networks with M parameters.

ZeRO-Offload Enables Large Model Training by Offloading Data and Compute to CPU



Offloading fp16 gradients and updating super node on CPU

Unique Optimal Offload Strategy

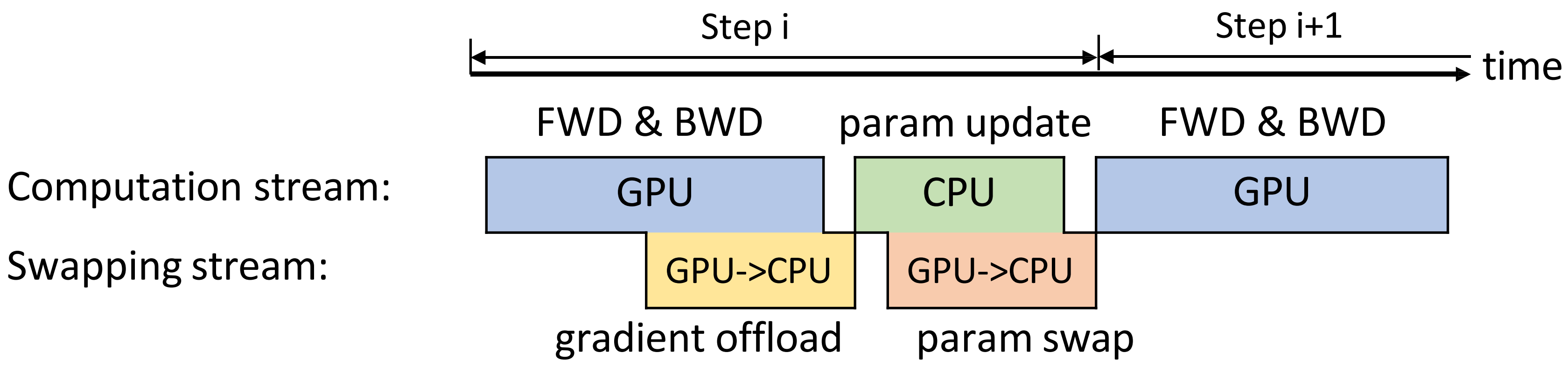


FWD-BWD	param16	gradient16	Update	Memory	Reduction
GPU	GPU	GPU	GPU	16M	1x(baseline)
GPU	GPU	CPU	GPU	14M	1.14x
GPU	GPU	GPU	CPU	4M	4x
GPU	GPU	CPU	CPU	4M	8x

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

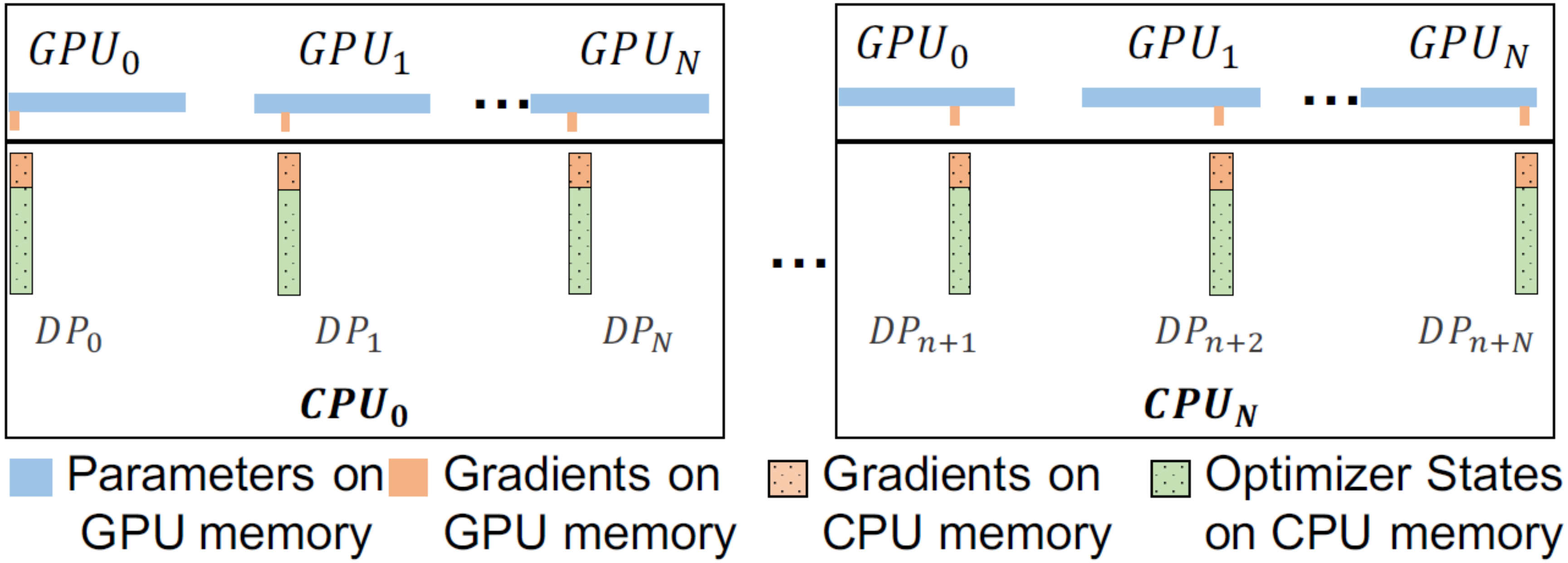
Offloading fp16 gradients and updating super node on CPU

ZeRO-Offload Single GPU Schedule



ZeRO-Offload training process on a single GPU.

Scaling with ZeRO-2



Partitioning based on ZeRO* before offloading

* ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. SC'20

Optimized CPU Execution

- Highly parallelized CPU optimizer implementation

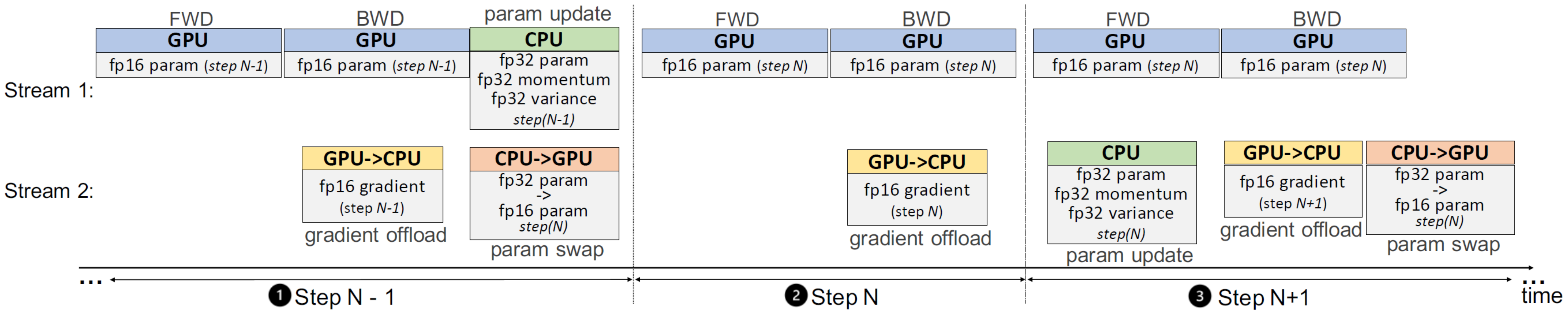
1) SIMD vector instruction for fully exploiting the hardware parallelism supported on CPU architectures.

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

- One-Step delayed parameter update



Evaluation Setup

- Hardware Overview

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

- Workloads

GPT-2, BERT-Large, Transformer based models

# params	batch size per GPU	MP setting in ZeRO-Offload	# layer	hidden size
1, 2 billion	32	1	20, 40	2048
4 billion	32	1	64	2304
6, 8 billion	16	1	53, 72	3072
10,11 billion	10,8	1	50,55	4096
12, 13 billion	4	1	60, 65	4096
15 billion	8	2	78	4096
20,40,60 billion	8	2	25,50,75	8192
70 billion	8	8	69	9216

- Baseline

PyTorch DDP, Megatron, SwapAdvisor, L2L, ZeRO-2

Evaluation Results

- How does ZeRO-Offload scale the trainable model size compared to existing multi-billion parameter training solutions on a single GPU/DGX-2 node?

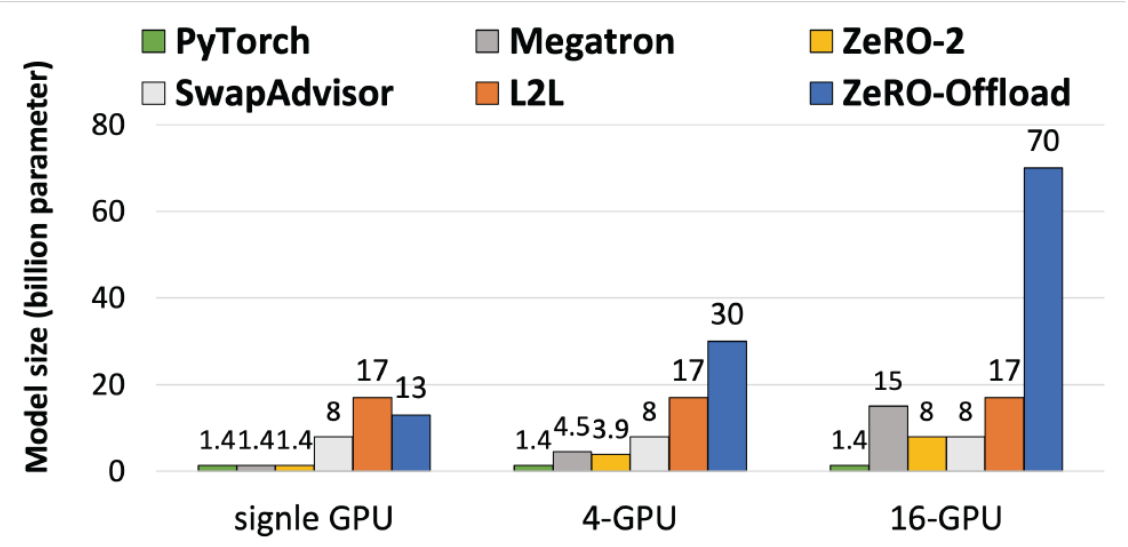


Figure 7: The size of the biggest model that can be trained on single GPU, 4 and 16 GPUs (one DGX-2 node).

Evaluation Results

- What is the training throughput of ZeRO-Offload on single GPU/DGX-2 node?

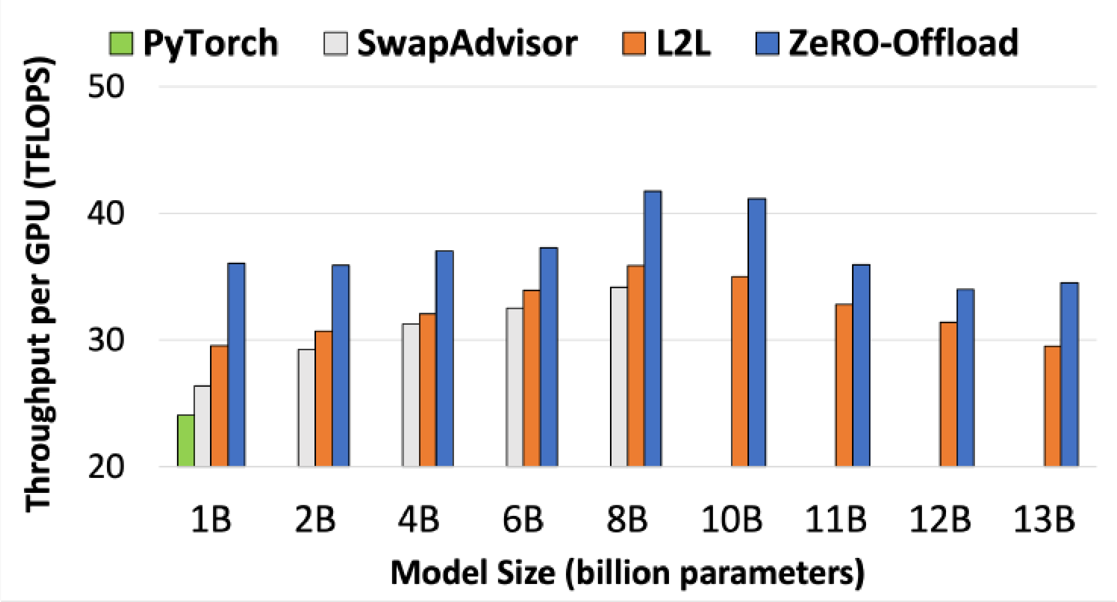


Figure 8: The training throughput with PyTorch, L2L, SwapAdvisor and ZeRO-Offload on a single GPU with a batch size of 512.

Discussion

- Strengths of this work:

1. **Efficiency**: It achieves very impressive memory saving while training large models on a single GPU.

2. **Availability**: It's already well implemented in the DeepSpeed library, can be directly used by data scientists.

3. **Scalability**: With the help of ZeRO-2, it can be applied to multi-GPU clusters.

4. What else…?

Discussion

- Potential issues & improvements of this work:

1. The motivation of this work mentions that it want to utilize CPU for computation (not only for memory expansion). Is the idea of offloading computations to CPU still good today?

1. For a GPU with 40GB Mem (A100), it needs 120GB Mem of Host CPU. Is CPU mem enough for scaling multiple GPUs?
2. Will the optimizer computation affect CPU’s other applications?
3. GPU computation power is growing very fast, today’s GPU may be much faster. How to avoid CPU computation to be the performance bottleneck?

Discussion

- Potential issues & improvements of this work:

1. Can the offload strategy and scheduling do better?

1. In this work, the parameters and activations are still always on GPU. Can these also be offloaded? What if the batch size is large?
2. Can we do a more fine-grained (e.g., OP level, tensor-level) optimized tensor migration scheduling, to best utilize the limited CPU-GPU interconnection bandwidth?

Thank You!

Haoyang Zhang

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN