ZeRO-Offload: Democratizing Billion-Scale Model Training

Jie Ren∗ , Samyam Rajbhandari† , Reza Yazdani Aminabadi† , Olatunji Ruwase† Shuangyan Yang∗ , Minjia Zhang† , Dong Li∗ , Yuxiong He†

Presenter: Yueming Yuan, Ananth Madan

Background and Motivation

Large Models are Large

- Training large models consumes a lot of memory
  - Model states: parameters, gradients, optimizer states (i.e. momentum and variance in Adam)
  - Residual states: activations, temporary buffers, fragmented memory (unusable)
- Model states are the primary memory bottleneck in training
  - For Adam and other optimizers, optimizer state is in fp32
  - Overall, model state is 16M bytes with Adam, where M is the number of parameters
    - 2M bytes per parameter and gradient, and 4M bytes per parameter, variance, and momentum
- **High resource requirement -> Limited accessibility of large model training**

Existing work & why ZeRO-offload

Scaling out model training: Accessibility challenge

Existing distributed training technologies:

- Pipeline parallelism
- Model parallelism
- ZeRO

Distribute the model states across multiple GPU devices

Require enough GPU devices that many institutions can not access

Scaling up model training

- Activation checkpointing
  - Not applicable for large model states
- Compression
  - Accuracy loss
- **Using external memory, e.g. CPU**

Solution: Heterogeneous DL training

Exploit CPU resources to reduce GPU resources requirement

Existing work:

- Target activation memory bottleneck, not for attention-based model
- Exploit CPU memory, but not CPU compute
- Mostly designed for single GPU

Solution: Heterogeneous DL training

Exploit CPU resources to reduce GPU resources requirement

Existing work:

- Target activation memory bottleneck, not for attention-based model
- Exploit CPU memory, but not CPU compute
- Mostly designed for single GPU
- L2L

Need a better heterogeneous strategy to satisfying **efficiency, scalability and usability** for large model training

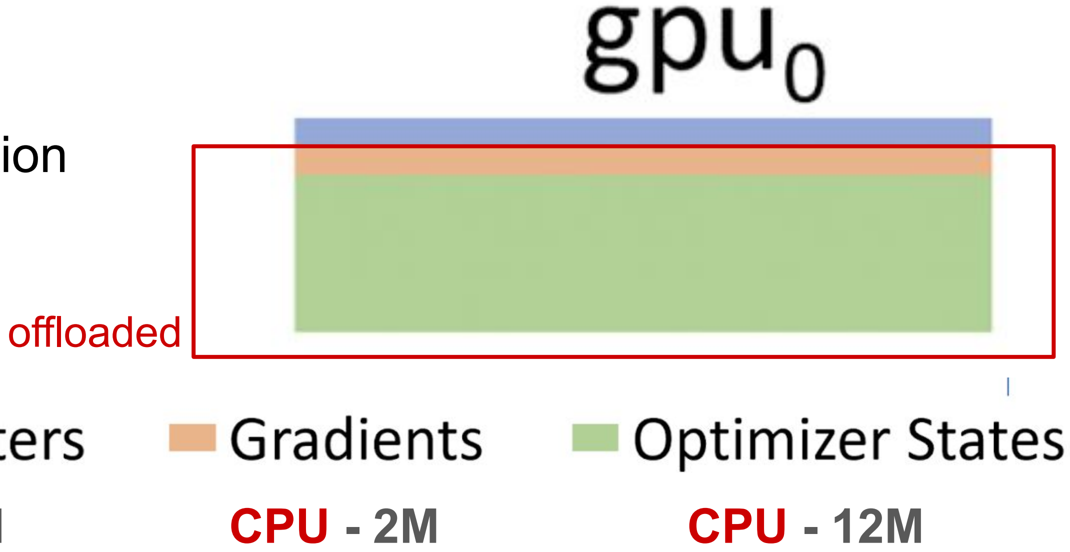
Key Idea & Designs

Part 1. Strategy choice
Part 2. Training Schedule
Part 3. CPU optimizations

Key Design

Offload the Optimizer states and gradients to the CPU memory, compute parameter updates at CPU

- Reduce 8x memory
- Lightweight CPU computation /communication burden



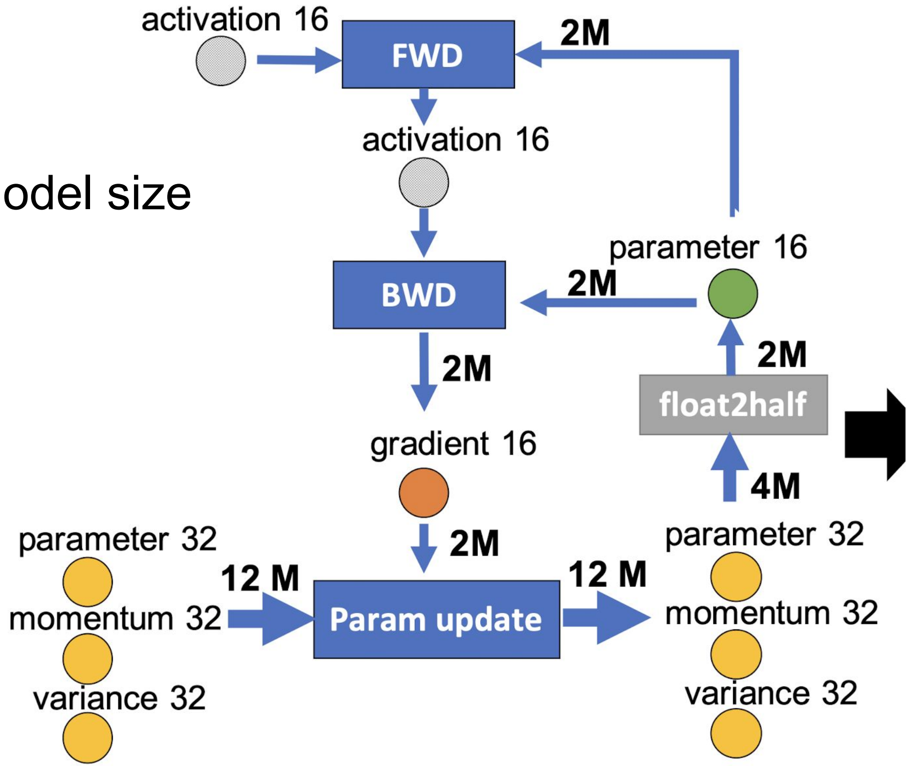
Part 1. How to choose the strategy?

Goal: design an optimal **CPU offloading strategy**, **offload part of the model states and computations to CPU**, to reduce the GPU memory requirement, while keeping the

- **Efficiency**
  - Avoid orders of magnitude performance reduction
- **Scalability**
  - Good performance on multi GPUs

Dataflow graph of mixed-precision training

M: model size



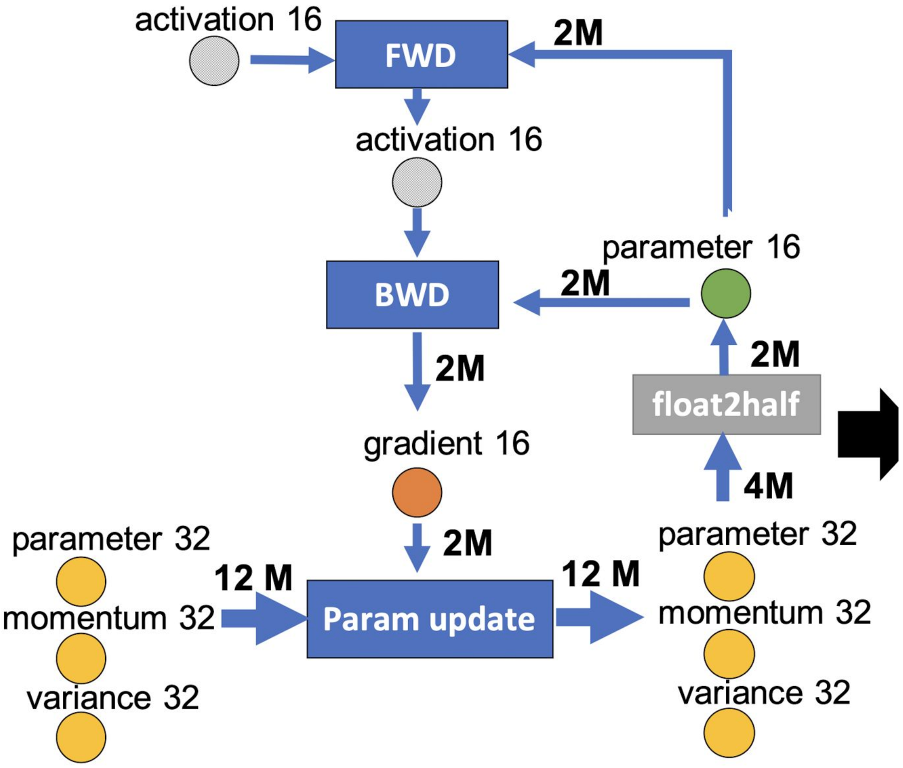
Model states:

- Parameter 16
- Gradient 16
- Parameter 32 + momentum 32 + variance 32

Computations:

- FWD
- BWD
- Param update
- float2half

Dataflow graph of mixed-precision training

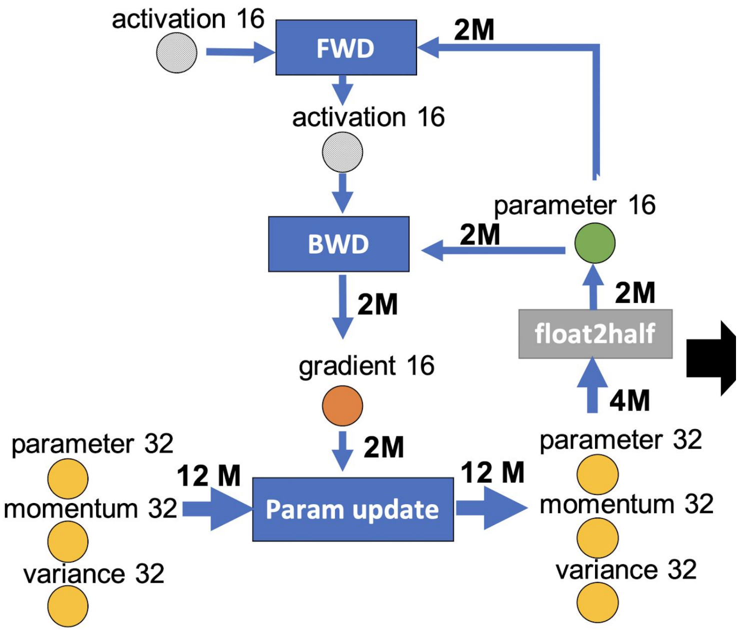


Need to offload part of the model states and computations

⬇️

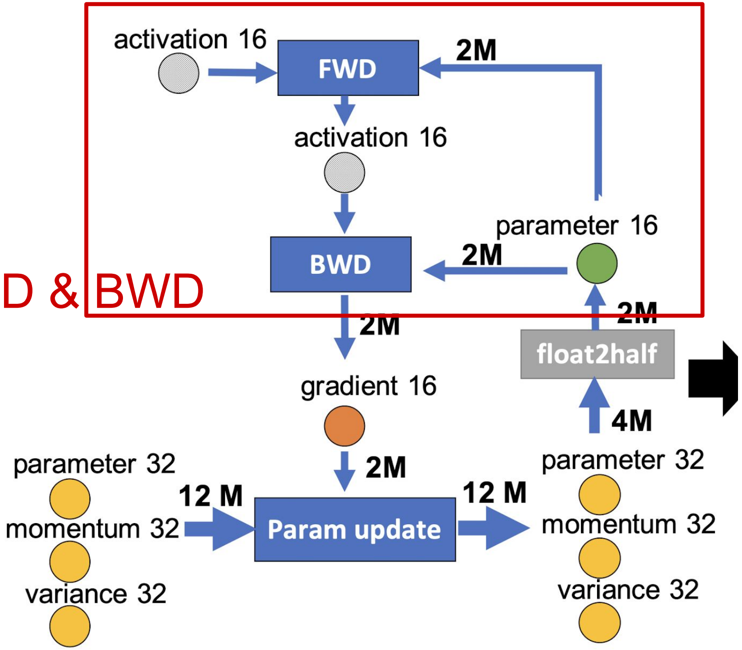
Find the best two-way partitioning of the graph

Design Considerations



1. **Minimize CPU computations**
   - CPU is multiple orders of magnitude slower than GPU
   - General compute complexity of a model: O(MB). Can only offload O(M) computations: norms, weight updates, …

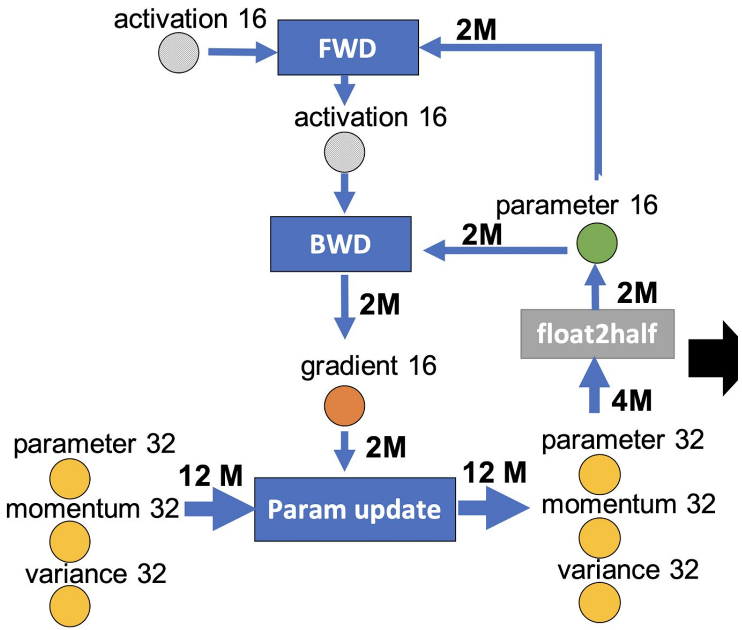
Design Considerations



1.FWD & BWD

1. **Minimize CPU computations**
   - CPU is multiple orders of magnitude slower than GPU
   - General compute complexity of a model: O(MB). Can only offload O(M) computations: norms, weight updates, …
   - **FWD-BWD must be assigned on the GPU.**

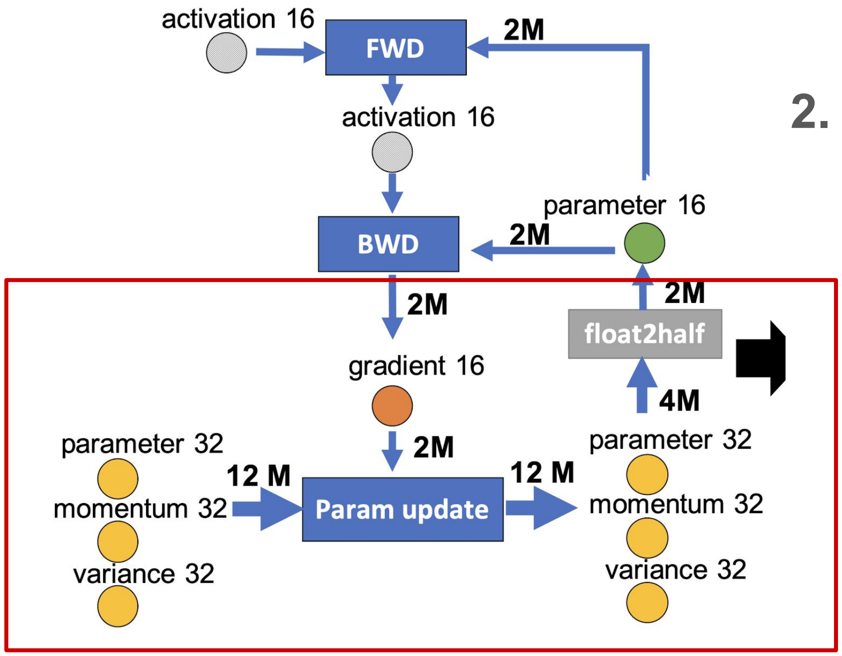
Design Considerations



2. Minimize GPU/CPU communication

- PCI-E is orders of magnitude slower than GPU memory bandwidth
- The minimum communication volume of the partition is 4M.

Design Considerations



2.Param update

2. Minimize GPU/CPU communication

- PCI-E is orders of magnitude slower than GPU memory bandwidth
- The minimum communication volume of the partition is 4M.
- **The fp32 model states must be co-located with the Param Update and the float2half computations.**

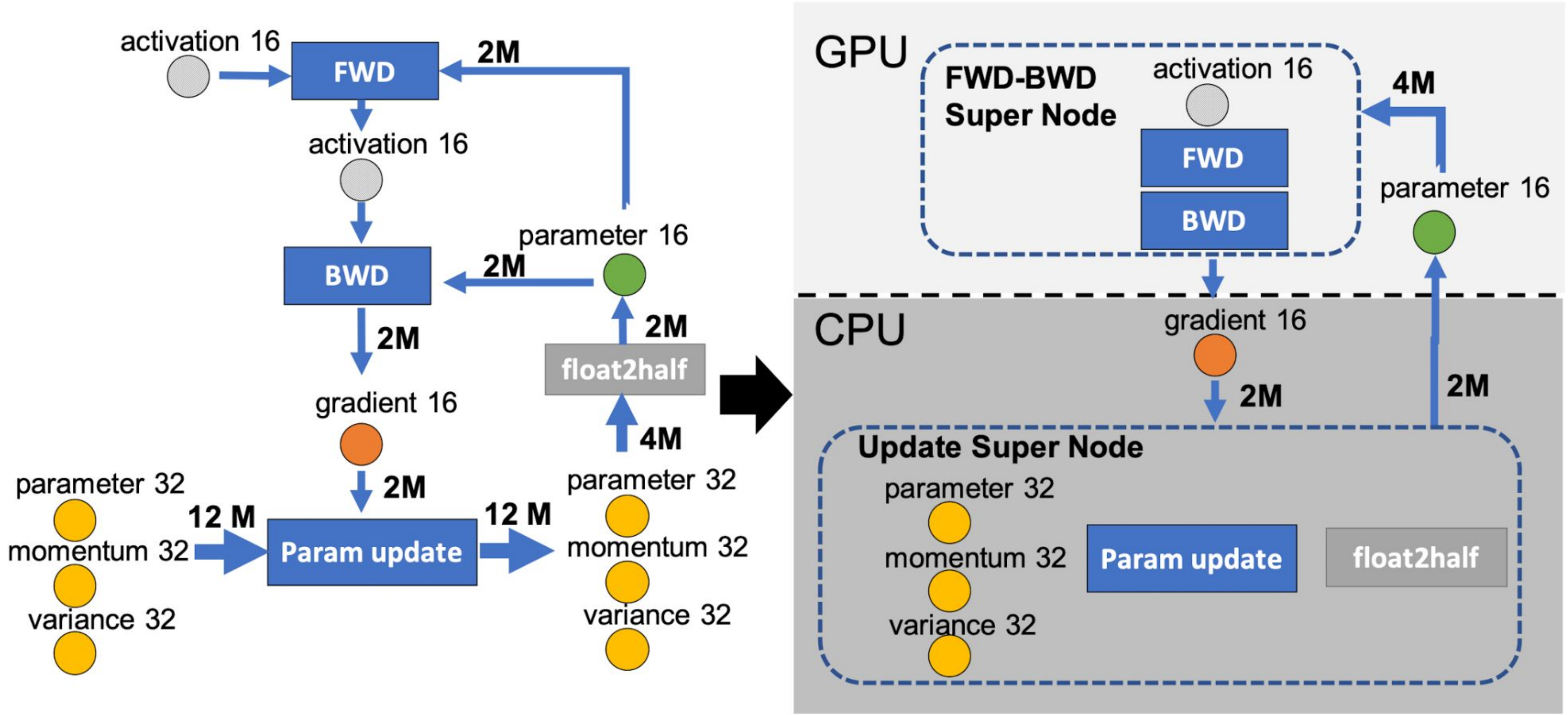
Design Considerations

1. **Minimize CPU computations**
   - FWD-BWD must be assigned on the GPU.
2. **Minimize GPU/CPU communication**
   - The fp32 model states must be co-located with the Param Update and the float2half computations.
3. **Maximize Memory savings**

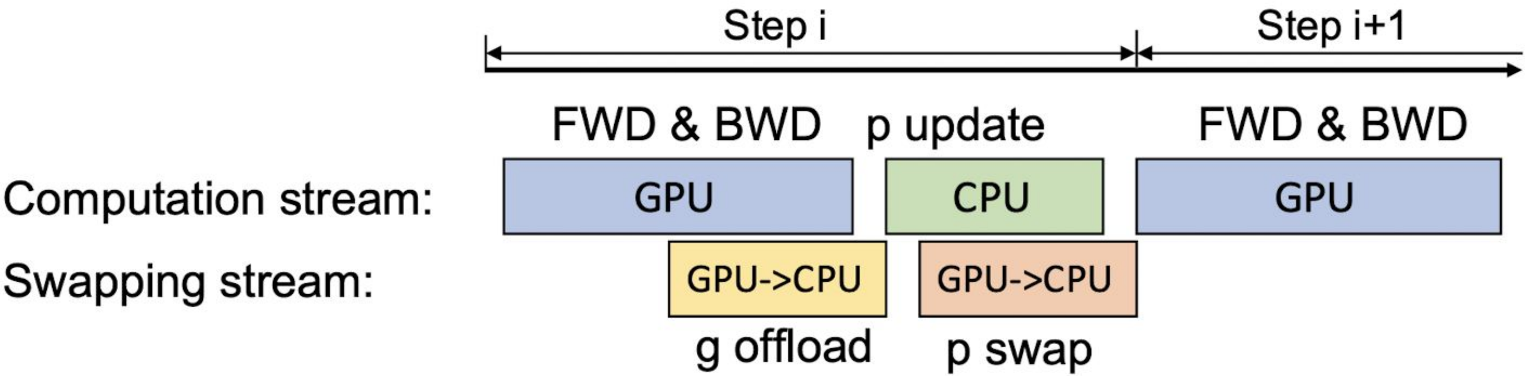
Table 1: Memory savings for offload strategies that minimizes communication volume compared to the baseline.

FWD-BWD	p16	g16	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

The unique and optimal offload strategy

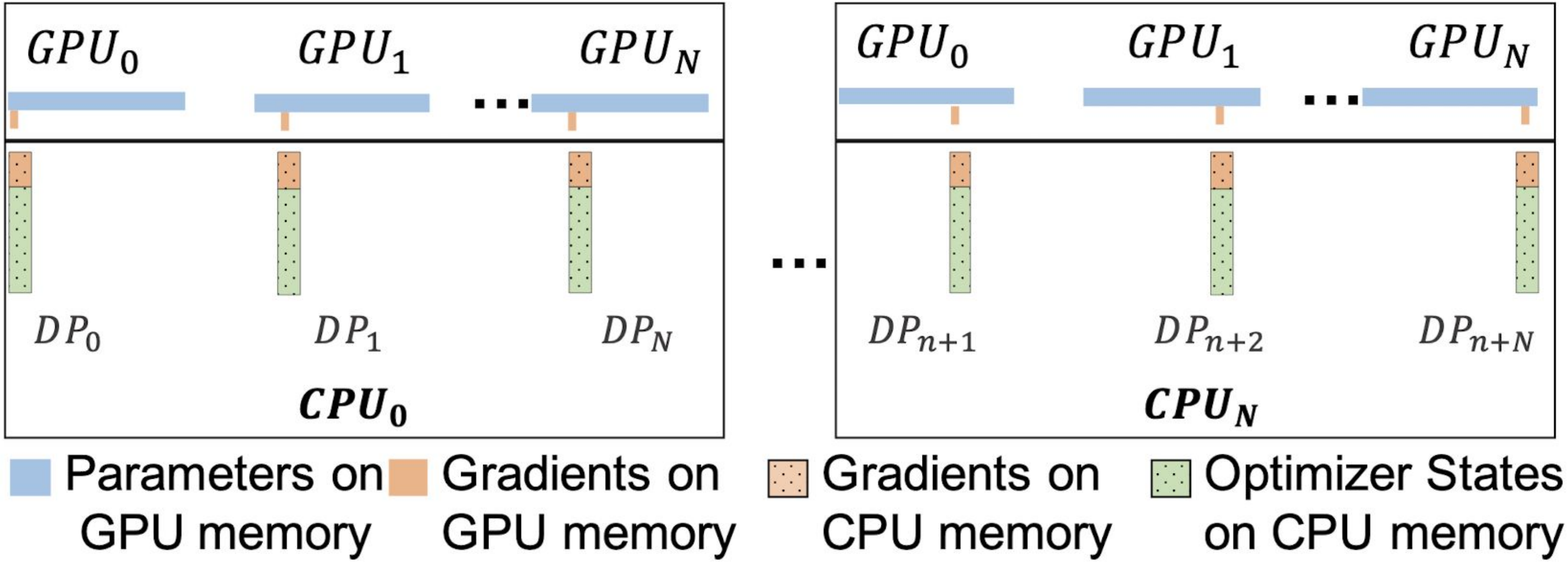


Part 2. ZeRO-offload Schedule - Single GPU



Gradients are transferred to CPU for each parameter/in small groups right after they are computed

ZeRO-offload Schedule - Multi-GPU, with ZeRO-2



CPU resources work in parallel to compute the weight update

Part 3. Optimized CPU execution - CPU Optimizer

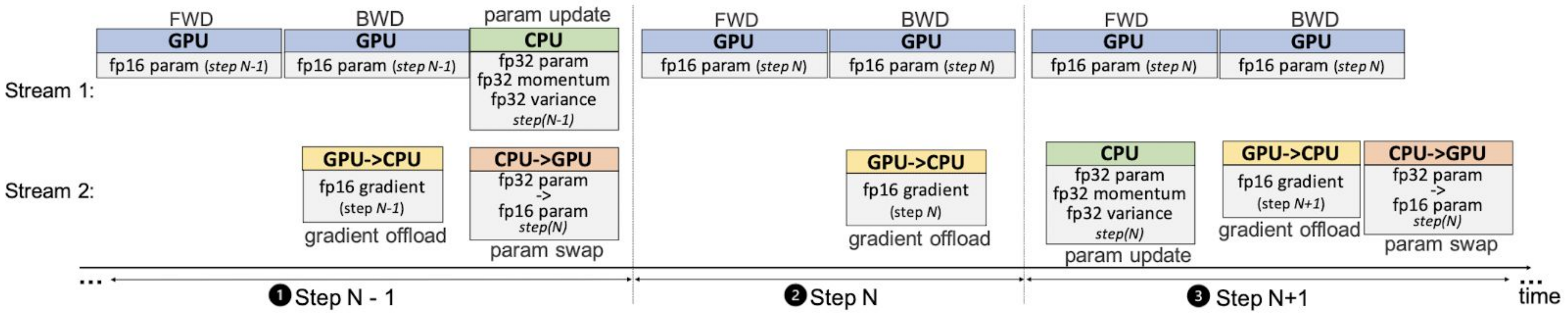
CPU Adam optimizer

- SIMD vector instruction
- Loop unrolling
- OMP multithreading

Tiled CPU-to-GPU 16FP parameter copy

Optimized CPU execution - DPU

One-step Delayed Parameter Update (DPU)



- Overlap the CPU computation with the GPU computation
- Do not apply in the first N-1 steps, apply from step N to avoid hurting convergence

Evaluation

Questions to Answer

- How does ZeRO-Offload trainable model size and throughput scale on a single GPU/node?
- How does ZeRO-Offload throughput scale to 128 GPUs
- How does the DPU and improved CPU-Adam affect throughput and model convergence?

Setup

- Single DGX-2 node vs. 8 connected DGX-2 nodes
  - Single- vs. multi-GPU training throughput evaluation
- GPT-2-like Transformer models
  - Additionally, BERT for evaluating convergent analysis
- Tested against the following frameworks
  - PyTorch DDP
  - Megatron
  - SwapAdvisor
  - L2L
  - ZeRO-2

Table 3: Model configuration in evaluation.

# 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

Evaluation (Model Size and Throughput)

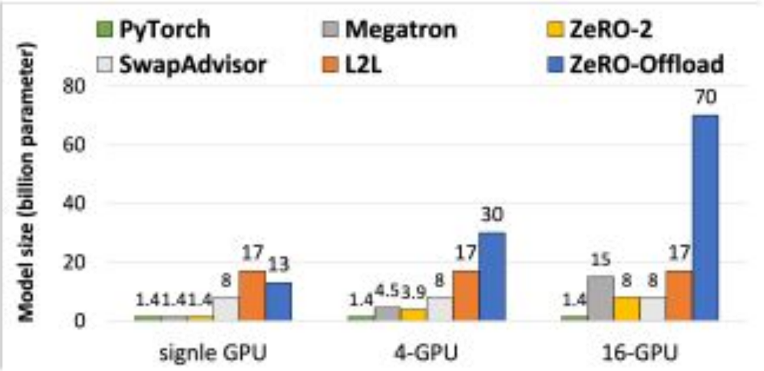


Figure 7: The size of the biggest model that can be trained on single GPU, 4 and 16 GPUs (one DGX-2 node).

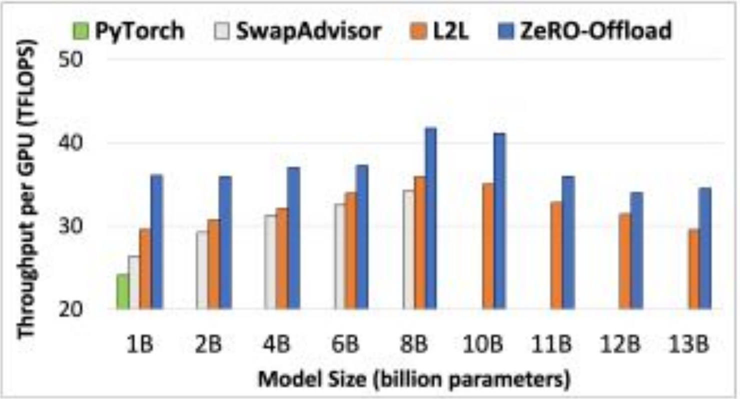


Figure 8: The training throughput with PyTorch, L2L, SwapAdvisor and ZeRO-Offload on a single GPU with a batch size of 512.

Multi-GPU (Single DGX-2) Training Throughput

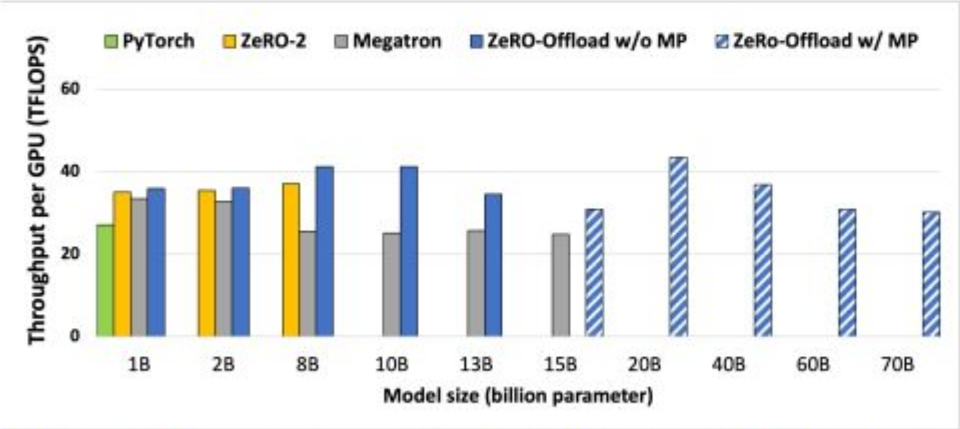


Figure 10: Training throughput with PyTorch, ZeRO-2, Megatron-LM, ZeRO-Offload without model parallelism and ZeRO-Offload with model parallelism.

Throughput Scalability to 128 GPUs

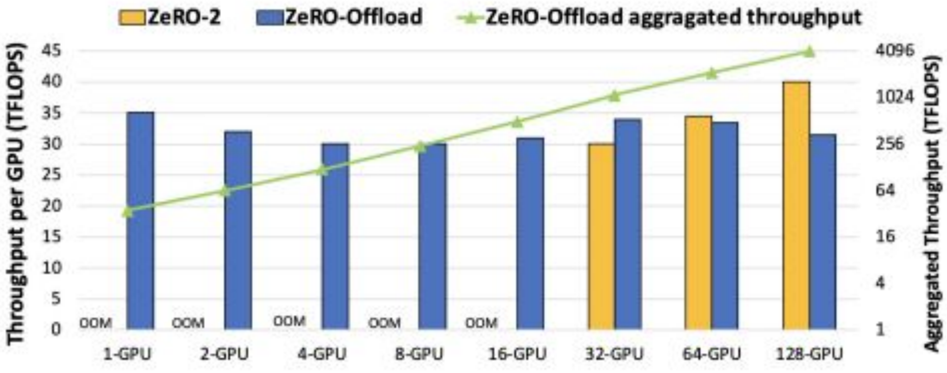


Figure 11: Comparison of training throughput between ZeRO-Offload and ZeRO-2 using 1–128 GPUs for a 10B parameter GPT2.

CPU-Adam

Table 4: Adam latency (s) for PyTorch (PT) and CPU-Adam.

#Parameter	CPU-Adam	PT-CPU	PT-GPU (L2L)
1 billion	0.22	1.39	0.10
2 billion	0.51	2.75	0.26
4 billion	1.03	5.71	0.64
8 billion	2.41	11.93	0.87
10 billion	2.57	14.76	1.00

Model Convergence with DPU

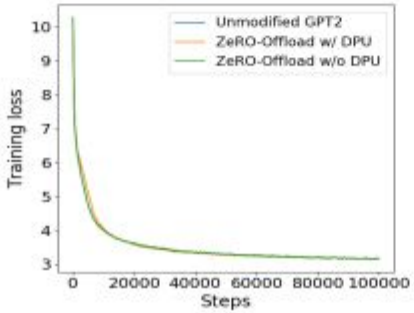


Figure 12: The training loss curve of unmodified GPT-2, ZeRO-Offload w/o DPU and ZeRO-Offload with DPU.

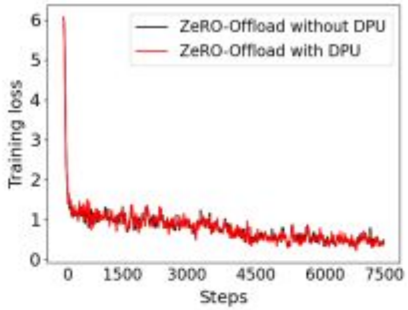


Figure 13: The fine-tuning loss curve of BERT, ZeRO-Offload w/o DPU and ZeRO-Offload with DPU.

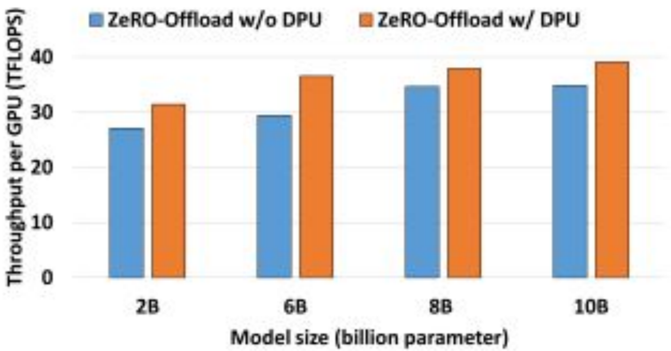


Figure 9: The training throughput is compared for w/o DPU and w/ DPU to GPT-2. Batch size is set to 8.

Holistic Analysis vs. PyTorch

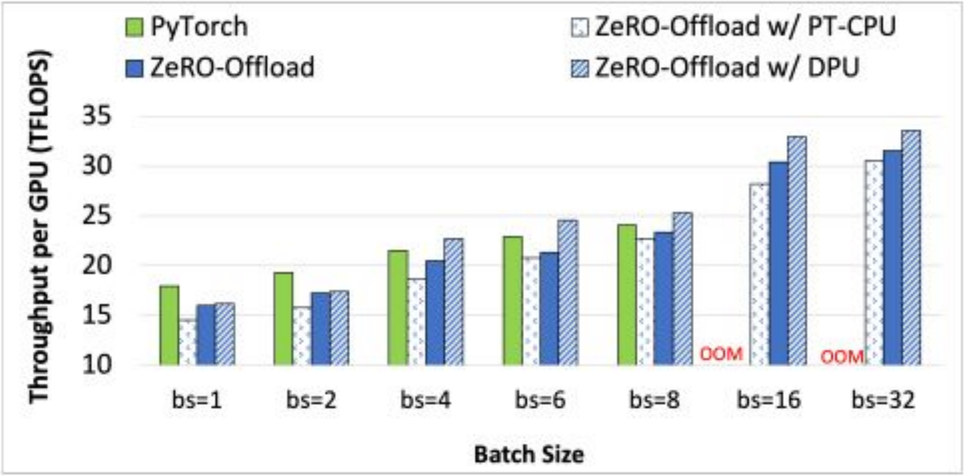


Figure 14: Comparison of training throughput with enabling offload strategies and optimization techniques step-by-step in ZeRO-Offload.

Strengths, Weaknesses, and Takeaways

Strengths

- Usability
  - Extremely easy to use: No code refactoring required
    - DeepSpeed library
  - Flexible configuration allows you to selectively turn off and on optimizations
- Scalability
  - Scalable GPU training via CPU offloading
    - Demonstrable improvements over regular ZeRO for larger batch sizes and model sizes
    - Helps with increasing throughput in environments without so many GPUs

Strengths, Weaknesses, and Takeaways

Weaknesses/Other Directions

- ZeRO-3 is not supported (at least in paper…)
- For even larger models, parameters memory become the new bottleneck
- The CPU parameter updating is too slow for a large model size
- Difficult to overlap the CPU computation and communication if the model size is too large

Takeaways

- Great step forward for making large model training more accessible offloading in heterogeneous systems
- Can we go bigger…?