

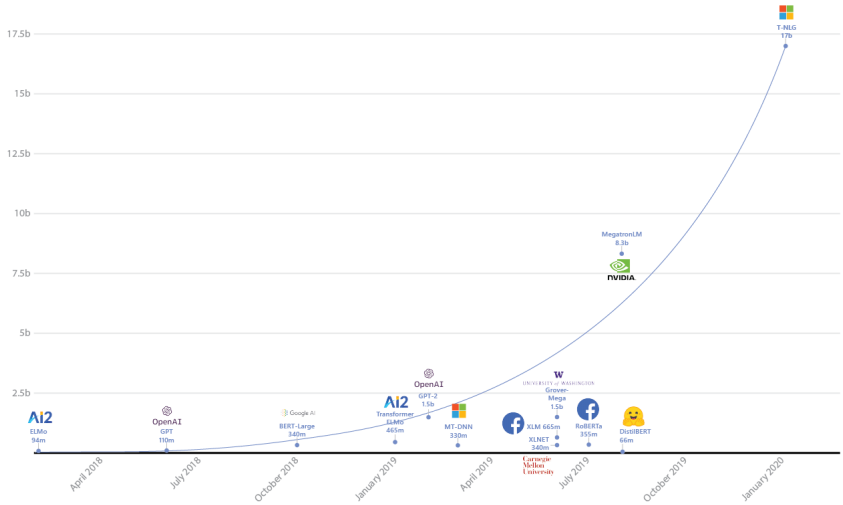
DeepSpeed ZeRO

Karol Kaczmarek

Adam Mickiewicz University
Poznań

Applica.ai
Warsaw

2021



Data Parallelism (DP)

- ▶ model parameters are replicated on each device
- ▶ at each step, a mini-batch is divided evenly across all the data parallel processes
- ▶ each process executes the forward and backward propagation on a different subset of data samples
- ▶ use of averaged gradients across processes to update the model locally
- ▶ `torch.nn.DataParallel`, `torch.nn.parallel.DistributedDataParallel`

Model Parallelism (MP)

- ▶ splits the model vertically, partitioning the computation and parameters in each layer across multiple devices
- ▶ requiring significant communication between each layer
- ▶ Mesh-Tensorflow [1], Megatron-LM [2] [3], L2L [4]

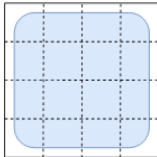
Data Parallelism (DP) and Model Parallelism (MP)

How the *model weights* are split over cores

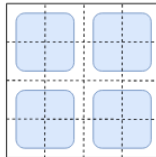
Data
Parallelism



Model
Parallelism

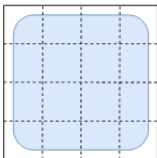


Model and Data
Parallelism



How the *data* is split over cores

Data
Parallelism



Model
Parallelism



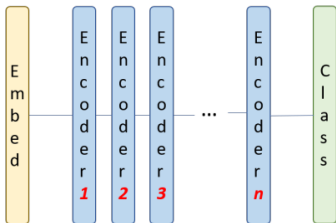
Model and Data
Parallelism



L2L (layer-to-layer)

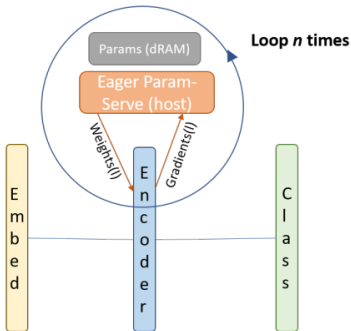
L2L enables training of networks by keeping one layer/block at a time in GPU memory and only moves tensors in the upcoming layer/block into GPU memory only if needed.

Conventional Execution n -layer NN



EPS - Eager Param-Server

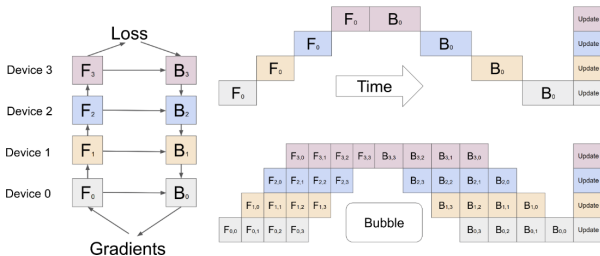
EPS with L2L execution



Pipeline Parallelism (PP)

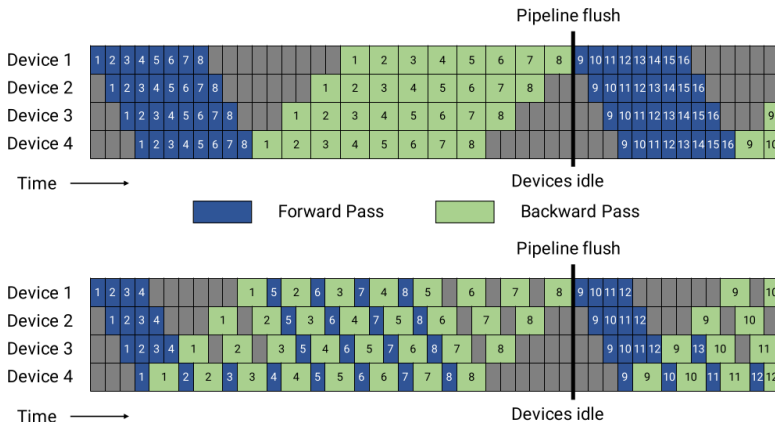
- ▶ split a model horizontally across layers running each partition on a different device
- ▶ use micro-batching (part of mini-batch) to hide the pipeline bubble
- ▶ PipeDream [5], GPipe [6], Megatron-LM [2] [3]

GPipe:



Pipeline Parallelism (PP)

Megatron-LM (April 2021) [3] - train 1 trillion parameters on 3072 GPUs (Nvidia A100 80GB - 384 DGX A100 nodes):



Other optimizations

- ▶ **Activation checkpointing** - reducing the memory footprint of activations - stored from forward pass and reused in backward pass
- ▶ **CPU-Offloading** - offloading model states to CPU memory
- ▶ **Adaptive optimization** - reducing memory consumption (use other optimizer, like Adafactor)

ZeRO [7]

- ▶ October 2019 (blogs - February 2020), Microsoft - PyTorch
- ▶ Turing-NLG - Turing Natural Language Generation - 17B parameters
- ▶ ZeRO (Zero Redundancy Optimizer) - optimize **memory**, vastly improving **training speed** while increasing **the model size** that can be efficiently trained
- ▶ runs 100B parameter models on a 400 Nvidia V100 GPU cluster (**8x** increase in model size and **10x** increase performance over state-of-the-art - T5 11B)
- ▶ share ZeRO as a part of our open source DL training optimization library called Deep-Speed - users do not need to modify their model - www.deepspeed.ai

ZeRO

ZeRO has two sets of optimizations:

- ▶ **ZeRO-DP** (ZeRO-powered data parallelism) aimed at reducing the memory footprint of the model states (removes the memory state redundancies across data-parallel processes by **partitioning** the model states instead of **replicating**)
- ▶ **ZeRO-R** targeted towards reducing the residual memory consumption

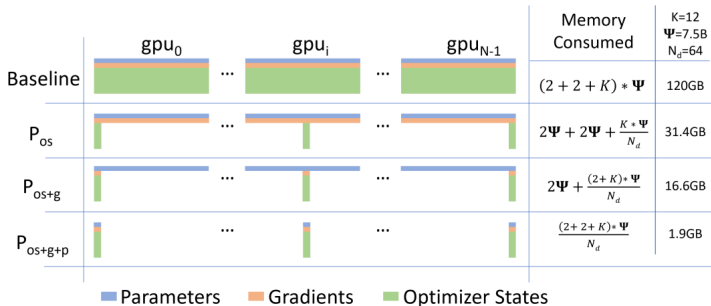
ZeRO-DP - Optimizing Model State Memory

ZeRO-DP eliminates memory redundancy by partitioning optimizer states (P_{os}), gradients (P_g) and parameters (P_p) across data parallel processes. **ZeRO-DP** has three main optimization stages:

- ▶ Optimizer State Partitioning (P_{os}): $4x$ memory reduction
- ▶ Add Gradient Partitioning (P_{os+g}): $8x$ memory reduction
- ▶ Add Parameter Partitioning (P_{os+g+p}): memory reduction is linear with N_d (N_d = DP degree/data parallelism degree, splitting across 64 GPUs - $N_d = 64$ will yield a $64x$ memory reduction), modest 50% increase in communication volume

With all three stages enabled, ZeRO can train a trillion-parameter model on just 1024 NVIDIA GPUs.

ZeRO-DP - Comparing stages



K - memory multiplier of optimizer states, ψ - model size, N_d = DP degree/data parallelism degree

ZeRO-DP - Memory consumption

- ▶ ZeRO-1 (P_{os}) partitions the optimizer states only
- ▶ ZeRO-2 (P_{os+g}) partitions gradients in addition to optimizer states
- ▶ ZeRO-3 (P_{os+g+p}) partitions all model states

DP	7.5B Model (GB)			128B Model (GB)			1T Model (GB)		
	P_{os}	P_{os+g}	P_{os+g+p}	P_{os}	P_{os+g}	P_{os+g+p}	P_{os}	P_{os+g}	P_{os+g+p}
1	120	120	120	2048	2048	2048	16000	16000	16000
4	52.5	41.3	30	896	704	512	7000	5500	4000
16	35.6	21.6	7.5	608	368	128	4750	2875	1000
64	31.4	16.6	1.88	536	284	32	4187	2218	250
256	30.4	15.4	0.47	518	263	8	4046	2054	62.5
1024	30.1	15.1	0.12	513	257	2	4011	2013	15.6

Bold-faced text are the combinations for which the model can fit into a cluster of 32GB V100 GPUs

ZeRO-R - Optimizing Residual State Memory

The rest of the memory is consumed by activations, temporary buffers, and unusable memory fragments. **ZeRO-R** optimize the residual memory consumed by:

- ▶ **activations** (stored from forward pass in order to perform backward pass) - optimize memory by activation partitioning or/and offloads to CPU when appropriate
- ▶ defines appropriate size for **temporary buffers** to strike for a balance of memory and computation efficiency
- ▶ proactively manages memory based on the different lifetime of tensors, preventing **memory fragmentation**

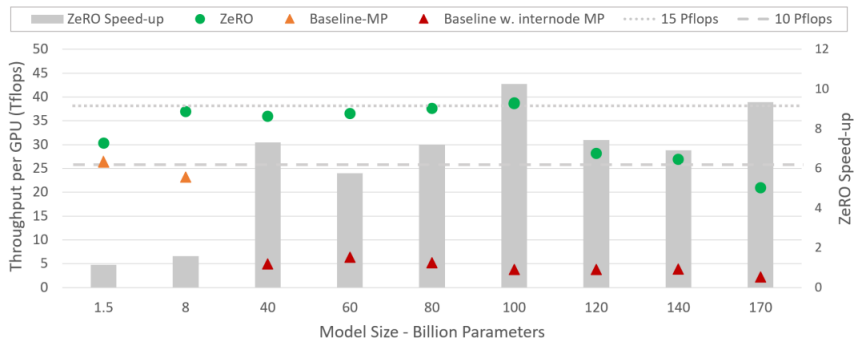
ZeRO-R - Residual Memory Consumption

Memory consumption for 1.5B parameter GPT-2 model:

- ▶ **Activations** - trained with sequence length of 1K and batch size of 32 requires about 60 GB of memory (proportional to the number of transformer layers * hidden dimensions * sequence length * batch size)
- ▶ **Temporary buffers** - fp32 buffer would required 6 GB of memory
- ▶ **Memory Fragmentation** - significant memory fragmentation when training very large models, resulting in out of memory issue with over 30% of memory still available in some extreme cases

ZeRO training

Baseline - 384 GPUs, ZeRO - 400 GPUs



More in paper: Memory/Communication Analysis, 1 Trillion Parameters, Comparing to Megatron-LM

ZeRO-Offload [8]

- ▶ January 2021, Microsoft - PyTorch
- ▶ train models with over 13 billion parameters on a single GPU (Nvidia V100 32GB)
- ▶ 10x increase size and without model change
- ▶ offloading data and compute to CPU
- ▶ designed to minimize the data movement to/from GPU
- ▶ reduce CPU compute time while maximizing memory savings on GPU to their existing training pipeline
- ▶ part of an Open-Source PyTorch library, DeepSpeed - www.deepspeed.ai
- ▶ does not require model refactoring to work - can enable with few lines of code change

ZeRO-Offload idea

- ▶ offload the gradients, optimizer states and optimizer computation to CPU, while keeping the parameters and forward and backward computation on GPU
- ▶ enables a 10x increase in model size, with minimum communication and limited CPU computation
- ▶ ZeRO-Offload works with ZeRO to scale deep learning training to multiple GPUs - ZeRO-Offload works symbiotically with ZeRO-2
 - ▶ ZeRO: ZeRO-1, ZeRO-2 and ZeRO-3 corresponding to the partitioning of the three different model states, optimizer states, gradients and parameters, respectively

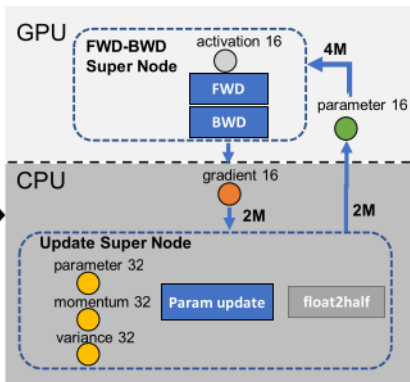
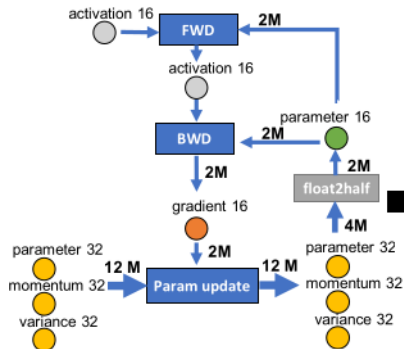
ZeRO-Offload idea

- ▶ compute complexity of training per iteration is generally given by $O(MB)$, where M is the model size and B is the batch size
- ▶ CPU computation should have a compute complexity lower than $O(MB)$ to avoid bottleneck
- ▶ forward and backward propagation ($O(MB)$ complexity) should be done on the GPU
- ▶ remaining computations (norm calculations, weight updates, etc. - $O(M)$ complexity) maybe offloaded and computed to the CPU

ZeRO-Offload design

- ▶ requires CPU to perform $O(M)$ computation compared to $O(MB)$ on GPU where M and B are the model size and batch sizes
- ▶ large batch sizes - CPU computation is not a bottleneck
- ▶ small batch sizes - CPU compute can be a bottleneck
- ▶ optimization to avoid bottleneck:
 - ▶ CPU optimizer that is up to 6x faster than the-state-of-art
 - ▶ one-step delayed parameter update (DPU) that allows overlapping the CPU optimizer step with GPU compute
- ▶ achieves excellent scalability on up to 128 GPUs (near linear speedup)

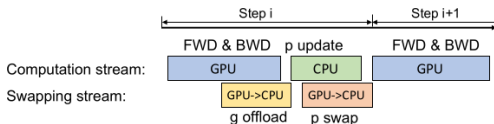
ZeRO-Offload dataflow



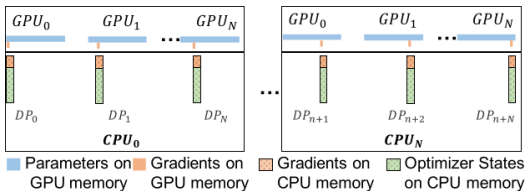
M - model parameters, 16 - fp16, 32 - fp32

ZeRO-Offload training process

Single GPU:



Multi GPUs:



Total CPU update time decreases with increased data parallelism

Fast CPU Adam optimizer

- ▶ SIMD vector instruction (best performance with AVX512 SIMD instruction set) for fully exploiting the hardware parallelism supported on CPU architectures
- ▶ loop unrolling, an effective technique for increasing instruction level parallelism that is crucial for better memory bandwidth utilization
- ▶ OMP multithreading for effective utilization of multiple cores and threads on the CPU in parallel
- ▶ Mixed Precision Training - casting fp16 <-> fp32

Fast CPU Adam optimizer - latency

#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

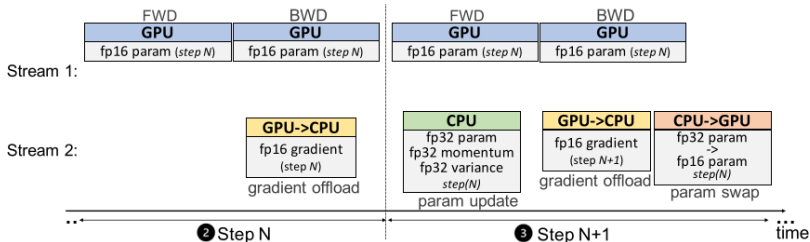
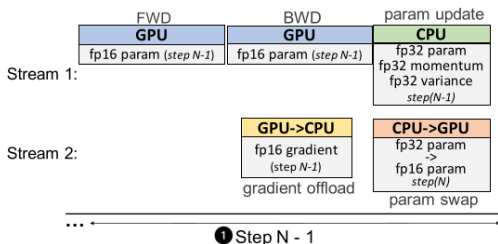
PT - PyTorch

One-Step Delayed Parameter Update

Delayed Parameter Update (DPU) training schedule:

- ▶ first $N - 1$ steps, are trained without DPU to avoid destabilizing
- ▶ step N , we obtain the gradients from the GPU, but we skip the CPU optimizer step, and do not update the fp16 parameters on the GPU either
- ▶ step $N + 1$, we compute the parameter updates on the CPU using gradients from step N , while computing the forward and backward pass on the GPU in parallel using parameters updated at step $N - 1$
- ▶ training with DPU achieves same model training accuracy with higher training throughput

One-Step Delayed Parameter Update



ZeRO-Offload training results

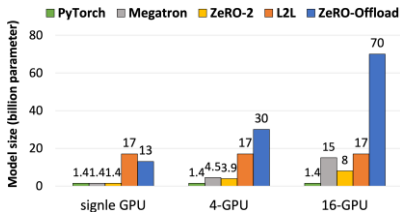
Train GPT-2 model on single DGX-2 (16 x NVIDIA V100 32GB, CPU Intel Xeon - support AVX512):

# 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

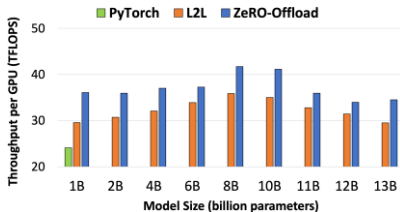
Compared with:

- ▶ **PyTorch DDP** - train with **DistributedDataParallel**
- ▶ **Megatron-LM** - train up to 8.3B parameter models using 512 GPUs
- ▶ **L2L** - keeping one Transformer block at a time in GPU
- ▶ **ZeRO** - extends data parallelism with memory redundancies

ZeRO-Offload training results



The size of the biggest model that can be trained

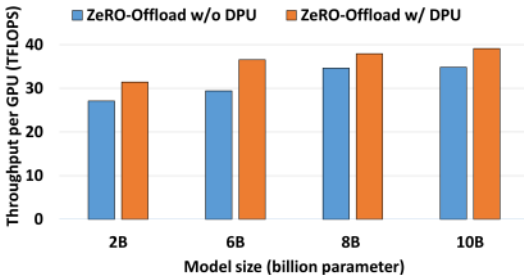


The training on a single GPU with a batch size of 512

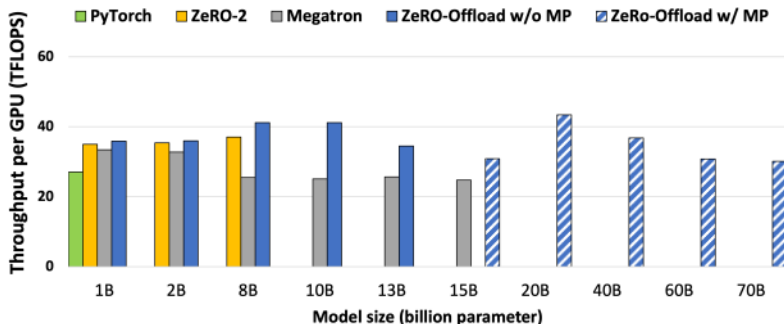
- ▶ ZeRO-Offload - 13B on a single GPU (9x larger than using PyTorch, Megatron, and ZeRO-2)
- ▶ ZeRO-Offload increases the model size by 50x (PyTorch), 4.5x (Megatron), 7.8x (ZeRO-2) and 4.2x (L2L) on 16 GPUs

ZeRO-Offload with/without DPU

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

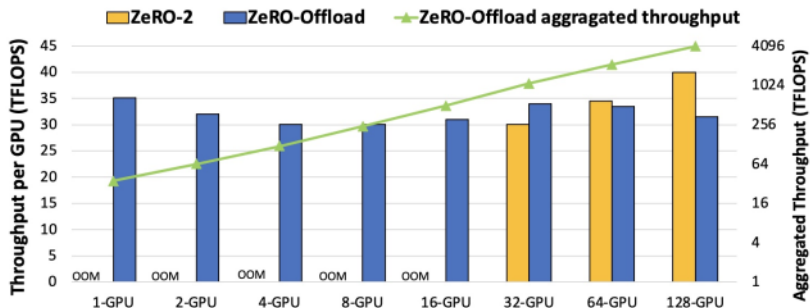


ZeRO-Offload with model parallelism



ZeRO-Offload scalability

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



ZeRO-Infinity [9]

- ▶ April 2021, Microsoft - PyTorch
- ▶ **ZeRO-Infinity** - heterogeneous system technology that leverages GPU, CPU, and NVMe memory to allow for unprecedented model scale on limited resources without requiring model code refactoring
- ▶ available through DeepSpeed (ZeRO-Infinity extends the ZeRO with new innovations in heterogeneous memory access called the **infinity offload engine**)
- ▶ running 32 trillion parameters on 32 NVIDIA DGX-2 nodes (512 V100 GPUs)
- ▶ supports 1 trillion parameters per NVIDIA V100 DGX-2 node - 50x increase over 3D parallelism

Memory/Bandwidth requirements (more analysis in paper)

Memory requirements for models:

Params (Trillions)	Layers	Hidden Size	Attn Heads	Model States (TB/Model)	TB/Node		Working Mem. per GPU (GB)	
					Act.	Act. Ckpt.	Model State	Act.
0.10	80	10K	128	1.83	2.03	0.05	1.95	1.63
0.50	100	20K	160	9.16	3.91	0.12	6.25	2.50
1.01	128	25K	256	18.31	7.13	0.20	9.77	3.56
10.05	195	64K	512	182.81	24.38	0.76	64.00	8.00
101.47	315	160K	1024	1845.70	88.59	3.08	400.00	18.00

Available memory and achievable bandwidth (NVIDIA V100 DGX-2 Cluster):

Nodes	GPUs	Aggregate Memory (TB)			GPU-GPU Bandwidth (GB/s)	Memory Bandwidth/GPU (GB/s)		
		GPU	CPU	NVMe		GPU	CPU	NVMe
1	1	0.032	1.5	28.0	N/A	600-900	12.0	12.0
1	16	0.5	1.5	28.0	150-300	600-900	3.0	1.6
4	64	2.0	6.0	112.0	60-100	600-900	3.0	1.6
16	256	8.0	24.0	448.0	60-100	600-900	3.0	1.6
64	1024	32.0	96.0	1792.0	60-100	600-900	3.0	1.6
96	1536	48.0	144.0	2688.0	60-100	600-900	3.0	1.6

ZeRO-Infinity

ZeRO-Infinity consisting five technologies:

- ▶ **infinity offload engine** - fully leverage heterogeneous architecture on modern clusters by simultaneously exploiting GPU, CPU and NVMe memory, and GPU and CPU compute
- ▶ **memory-centric tiling** - handle massive operators without requiring model parallelism
- ▶ **bandwidth-centric partitioning** - leveraging aggregate memory bandwidth across all parallel devices
- ▶ **overlap-centric design** - overlapping compute and communication
- ▶ **ease-inspired implementation** - avoid model code refactoring

Design for unprecedented scale

NVMe storage that is over **50x** larger than the GPU memory and nearly **20x** larger than CPU memory.

- ▶ **Infinity offload engine for model states** - built on top of ZeRO-3 which partitions all model states to remove memory redundancy refactoring (DeepNVMe - C++ NVMe read/write library - capable of achieving near peak read and write bandwidths on the NVMe)
- ▶ **CPU Offload for activations** - offload activation memory to CPU memory, when necessary (0.76 TB for 10 trillion parameter model)
- ▶ **Memory-centric tiling for working memory** - breaking down a large operator into smaller tiles that can be executed sequentially (automatically)

Design for excellent training efficiency

- ▶ **bandwidth-centric partitioning** - data mapping and parallel data retrieval strategy for offloaded parameters and gradients that allows ZeRO-Infinity to achieve virtually unlimited heterogeneous memory bandwidth (using **allgather** instead of a **broadcast** when a parameter needs to be accessed)
- ▶ **overlap centric design** - allows to overlap not only GPU-GPU communication with computation but also NVMe-CPU and CPU-GPU communications (traces the forward and backward computation on that fly and prefetches the parameter requires by the future operators)

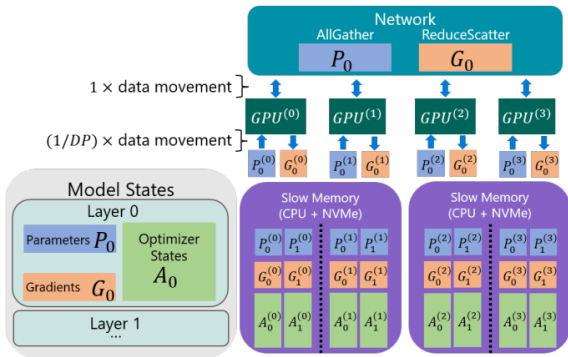
Design for ease of use

Eliminates the need for manual model code refactoring even when scaling to trillions of parameters.

Ease-inspired implementation with two automated features:

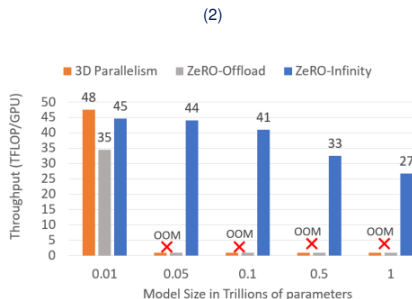
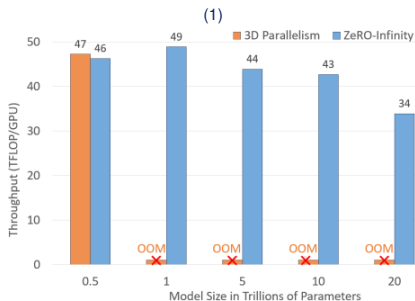
- ▶ **automated data movement** - gather and partition parameters right before and after they are required during the training (pre/post forward/backward hooks into PyTorch submodules)
- ▶ **automated model partitioning during initialization** - wrapping the constructor of all module classes so that parameters of each submodule are partitioned and offloaded immediately

Training overview



- model with two layers on four data parallel (DP) ranks
- partitioned parameters (P_n) are moved from slow memory to GPU and then collected to form the full layer
- after gradients are computed, they are aggregated, repartitioned, and then offloaded to slow memory
- $P_0^{(2)}$ is the portion of 0-layer parameters owned by $GPU^{(2)}$

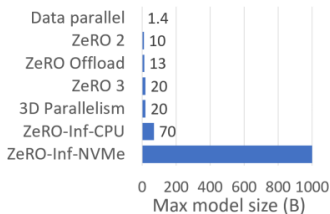
Efficiently training



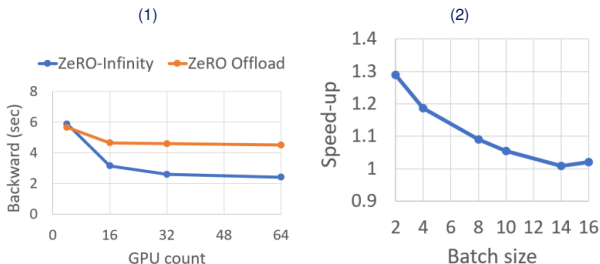
- 1 ZeRO-Infinity efficiently trains 40x larger models than 3D parallelism on 512 GPUs
- 2 ZeRO-Infinity can train up to 1T model on a DGX-2 node (16 GPUs) without model parallelism

Different device placement and partitioning strategies

Name	Optimizer + Grad (devices/partitioned)	Parameters (devices/partitioned)
Data parallel	[GPU] / ✗	[GPU] / ✗
ZeRO 2	[GPU] / ✓	[GPU] / ✗
ZeRO-Offload	[CPU,GPU] / ✓	[GPU] / ✗
3D Parallelism	[GPU] / ✓	[GPU] / ✓
ZeRO 3	[GPU] / ✓	[GPU] / ✓
ZeRO-Inf-CPU	[CPU, GPU] / ✓	[CPU,GPU] / ✓
ZeRO-Inf-NVMe	[NVMe,CPU,GPU] / ✓	[NVMe,CPU,GPU] / ✓



Impact of system features



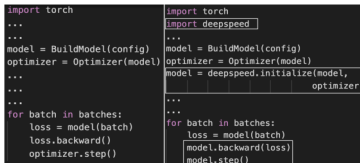
- 1 ZeRO-Infinity speedup of nearly 2x at 64 GPUs compared to ZeRO-Offload
- 2 **Prefetching** and **overlapping** are crucial to achieving good performance at small batch sizes per GPU - 8B parameter model with 64 GPUs

More analysis in paper

DeepSpeed

▶ DeepSpeed

- ▶ works best with fp16 and Fused Adam/DeepSpeed CPU-Adam
- ▶ <https://www.deepspeed.ai>



The image shows two side-by-side code snippets in a dark-themed editor. The left snippet is standard PyTorch code for training a model. The right snippet is the equivalent code using DeepSpeed, with additional imports and initialization steps. Red boxes highlight the differences: 'import deepspeed' and 'deepspeed.initialize(model, [devices], [num_gpus], [num_nodes])' on the right, and 'model.backward(loss)' and 'model.step()' on the right, which replace the standard PyTorch 'loss.backward()' and 'optimizer.step()'.

```
import torch
...
...
model = BuildModel(config)
optimizer = Optimizer(model)
...
...
for batch in batches:
    loss = model(batch)
    loss.backward()
    optimizer.step()
```

```
import torch
import deepspeed
...
...
model = BuildModel(config)
optimizer = Optimizer(model)
model = deepspeed.initialize(model,
                             [devices], [num_gpus], [num_nodes],
                             optimizer)
...
...
for batch in batches:
    loss = model(batch)
    model.backward(loss)
    model.step()
```

▶ DeepSpeed with PyTorch Lightning

- ▶ https://pytorch-lightning.readthedocs.io/en/stable/advanced/advanced_gpu.html?highlight=DeepSpeed#deepspeed

▶ DeepSpeed with transformers trainer

- ▶ https://huggingface.co/transformers/main_classes/trainer.html

References I

- [1] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. Hechtman, "Mesh-tensorflow: deep learning for supercomputers," 2018.
- [2] M. Shoeybi, M. Patwary, R. Puri, and et al., "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2019.
- [3] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, and et al., "Efficient large-scale language model training on gpu clusters," 2021.
- [4] B. Pudipeddi, M. Maral, J. Xi, and S. Bharadwaj, "Training large neural networks with constant memory using a new execution algorithm," 2020.
- [5] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. Devanur, G. Granger, P. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," 2019.
- [6] Y. Huang, C. Yonglong, D. Chen, H. Lee, J. Ngiam, Q. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," 2020.
- [7] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," 2019.
- [8] J. Ren, S. Rajbhandari, R. Aminabadi, Yazdani, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "Zero-offload: Democratizing billion-scale model training," 2021.
- [9] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," 2021.