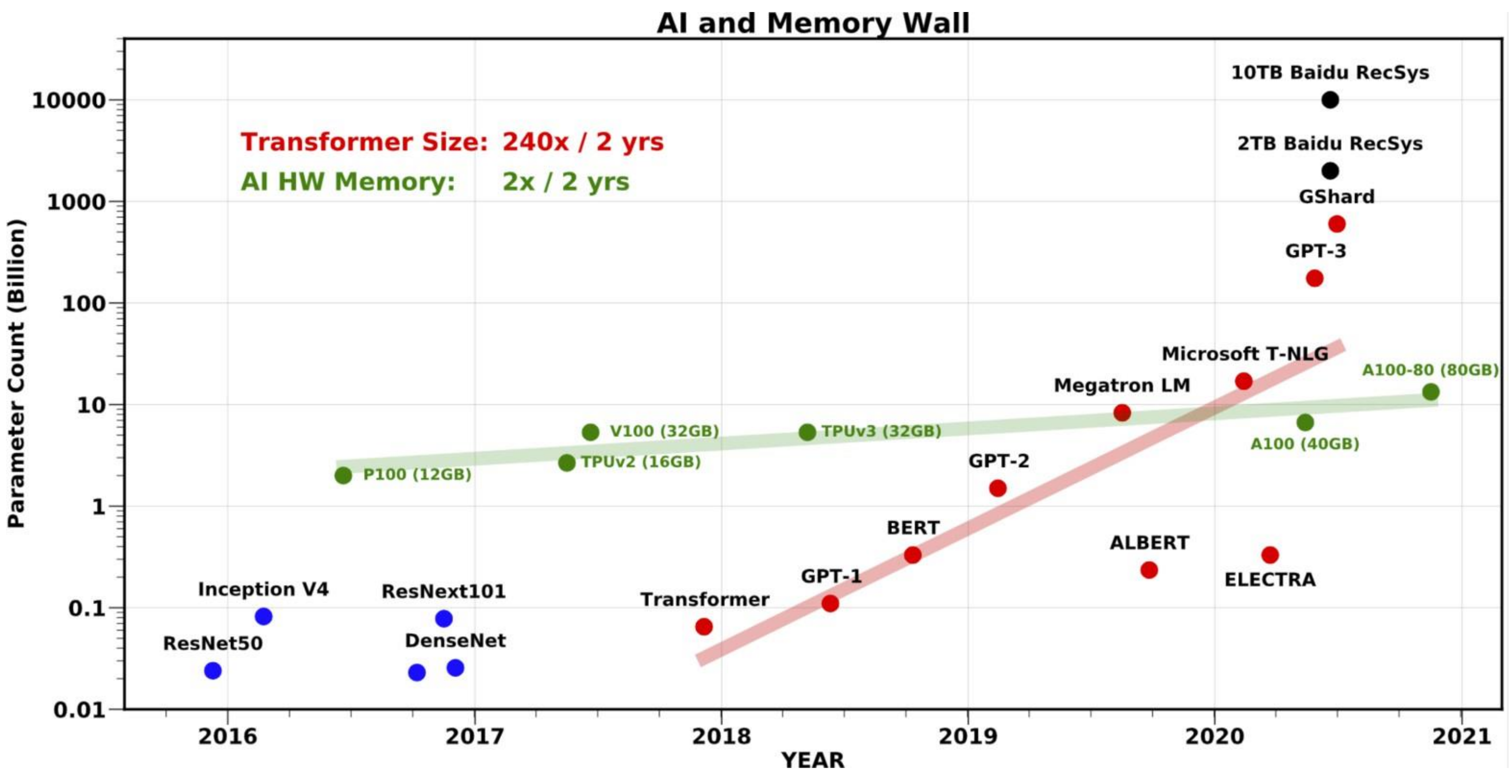
ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning

Authors: Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, Yuxiong He

Large model training landscape

- GPU Memory Wall
  - 1T (10T) params: 800 (8K) V100 GPUs
  - How do we support the growth in model size?
- Accessibility to large model training
  - 256 GPUs to fine-tune GPT-3
  - Limited access to such resources
- Model code refactoring
  - Re-writing the model using 3D parallelism (tensor-slicing + pipeline parallelism)
  - Painful and error prone

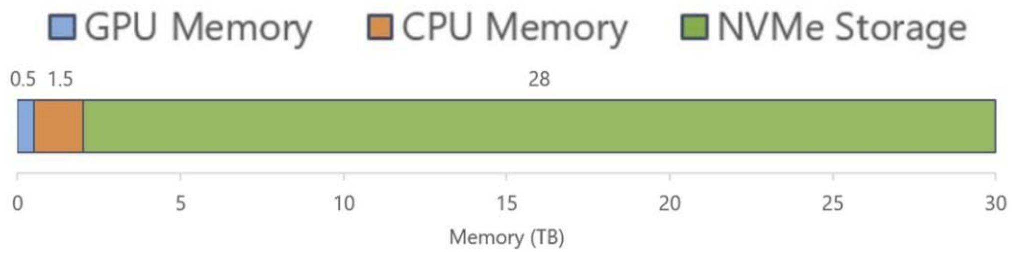


*AI and Memory Wall. (This blogpost has been written in... | by Amir Gholami| riselab | Medium

Beyond the GPU Memory

- Modern clusters have heterogeneous memory systems.
- GPU memory comprises a small fraction
- Leverages GPU/CPU/NVMe memory
  - 32T params on 32 nodes
  - 1T params on a single node
- GPT-3 can be fine-tuned on a single node

Memory available on a Single DGX-2 Node



How to leverage non-GPU memory?

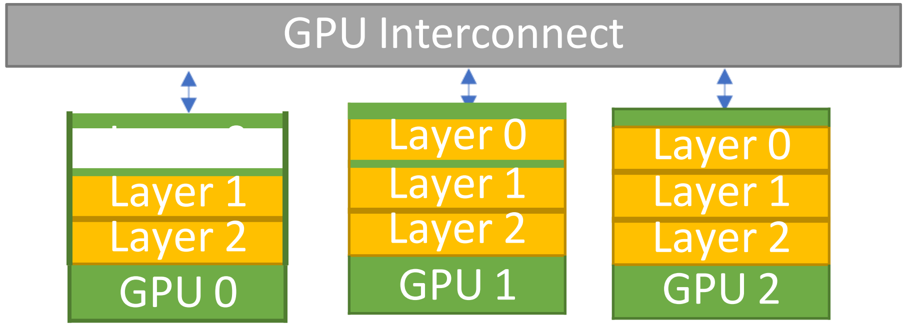
- Can we extend an existing parallel training technology to use CPU/NVMe memory?

- Data Parallelism : Replication causes memory explosion
- Tensor-Slicing: scaling challenge for multi-GPU
- Pipeline-Parallelism: Requires significant code refactoring

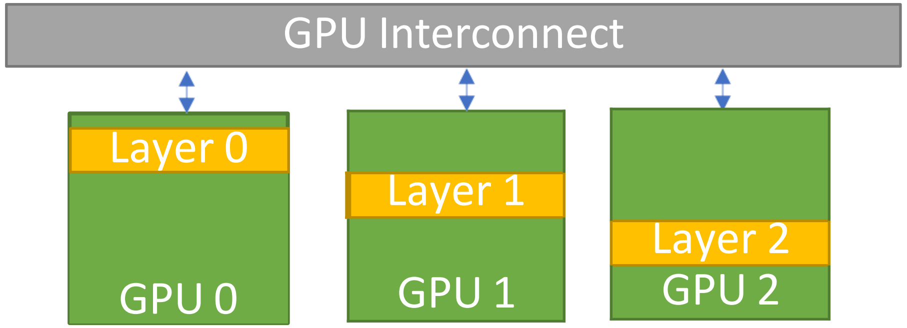
- What about Zero Redundancy Optimizer (ZeRO)?
  - Efficiently scale across nodes – trillions of parameters
  - No model code refactoring necessary

ZeRO: Zero Redundancy Optimizer

- Memory efficient form of data parallelism
- Each GPU stores a mutually exclusive subset of the parameters
- Broadcast parameters from owner to all the GPUs as needed



Model States mapping in **Data Parallel** Training



Model States mapping in **ZeRO** Training

Zero Infinity Overview

- Infinity offload engine
  - Based on GPU memory,
  - Offload partitioned model states -> CPU/NVMe
  - Fetch back at time of needed
- Optimization: memory centric tiling
  - Breakdown large linear operator –> small sequential ones
  - Reduce required working memory

ZeRO with CPU/NVME Offload

- Store in CPU/NVME instead of GPU
- Send from CPU/NVMe to GPU
- Broadcast or reduce as ZeRO

- Is NVME←→ GPU bandwidth sufficient?
  - Efficiency analysis based on bandwidth

$$efficiency = \frac{compute\_time}{compute\_time + communication\_time}$$

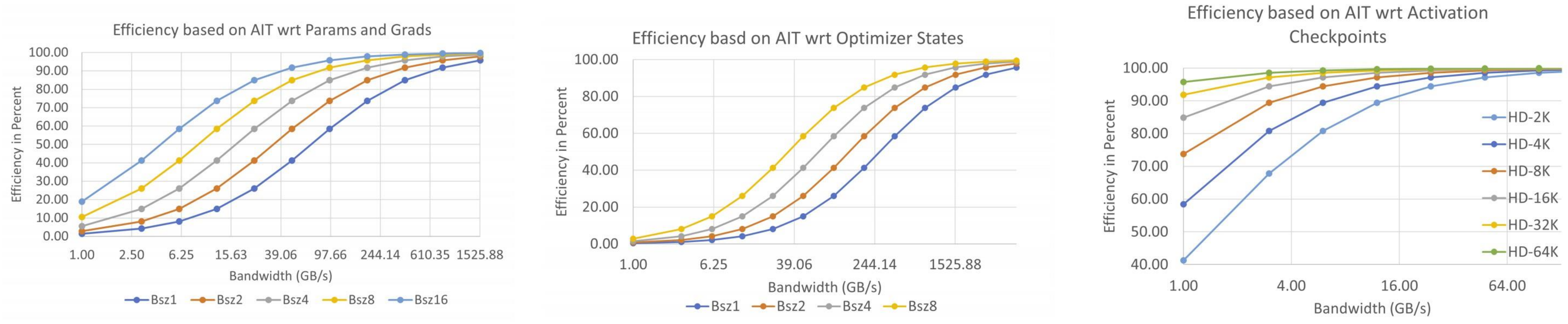
$$compute\_time = \frac{total\_computation}{peak_{tp}}$$

$$ait = \frac{total\_computation}{total\_data\_movement}$$

$$communication\_time = \frac{total\_data\_movement}{bw} = \frac{total\_computation}{ait \times bw}$$

$$efficiency = \frac{ait \times bw}{ait \times bw + peak_{tp}}$$

Efficiency as a function of bandwidth



(a) Parameter and Gradient Bandwidth **(b) Optimizer States bandwidth** **(c) Activation Checkpoint Bandwidth**

Figure 3: Impact of bandwidth on efficiency assuming an accelerator with 70 TFlops of single GPU peak achievable throughput.

Data Type	Overlap	Requirement
Params/Grads	Yes	70 GB/s
Optimizer States	No	1500 GB/s
Activations	Yes	1-4 GB/s

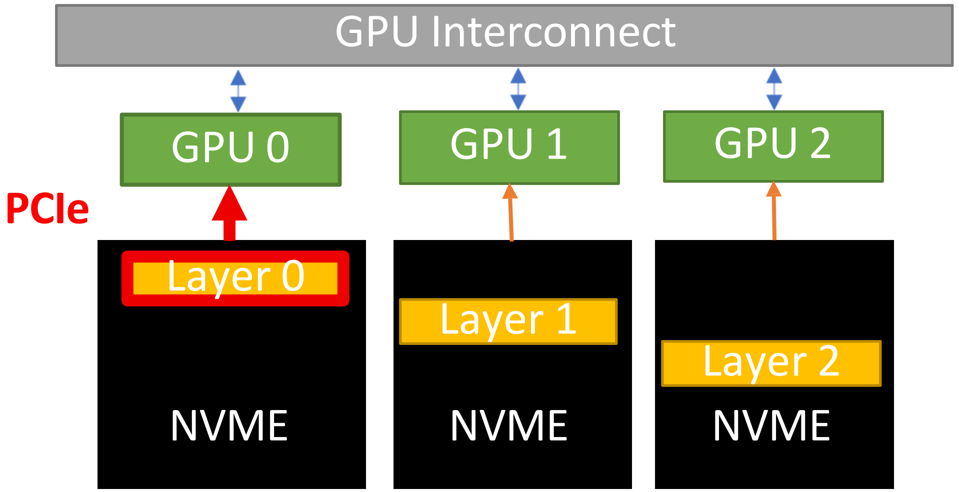
Overlap: prefetch data from CPU to GPU before computation.
Need BW to achieve at least 50% efficiency.

ZeRO with CPU/NVME Offload

Example: Training using ZeRO with Offload on 64x DGX-2 nodes.

ZeRO with non-GPU memory

GPUs	Data Type	Required
1024	Params/Grads	70 GB/s
1024	Optimizer States	1500 GB/s
1024	Activations	1-4 GB/s

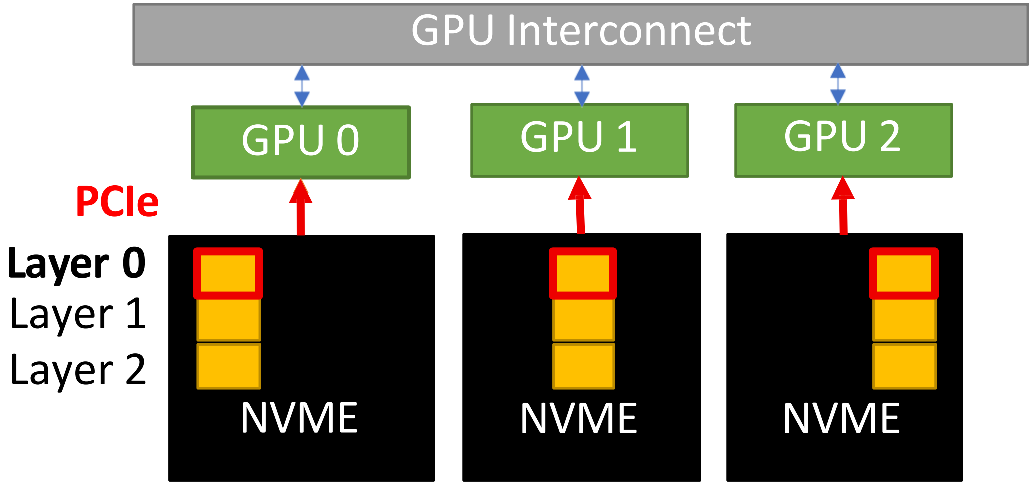


- Is CPU/NVME←→ GPU bandwidth sufficient?
  - Params/grads: PCIe bottleneck 12 GB/s
  - Optimizer States: More than needed
  - Activations: CPU Memory bandwidth sufficient

Efficiency Design Choice

- Require 70GB/s
  - GPU-GPU BW can satisfy
  - But not PCIE’s 12GB/s BW
  - Zero-Offload, CPU-> owner GPU then broadcast
    - Require larger batch size
    - Activation memory too large for CPU memory
    - May not lead to effective convergence

BW-centric Partition



- Partition each parmaeter across GPUs
- Send from NVMe to GPU in parallel

- Bandwidth Increases linearly with devices
  - #gpus x host-to-device bandwidth
  - CPU -> GPU: 64 GB/s – 4 TB/s (1-64 nodes)
  - NVMe -> GPU: 28 GB/s – 1.8 TB/s (1-64 nodes)

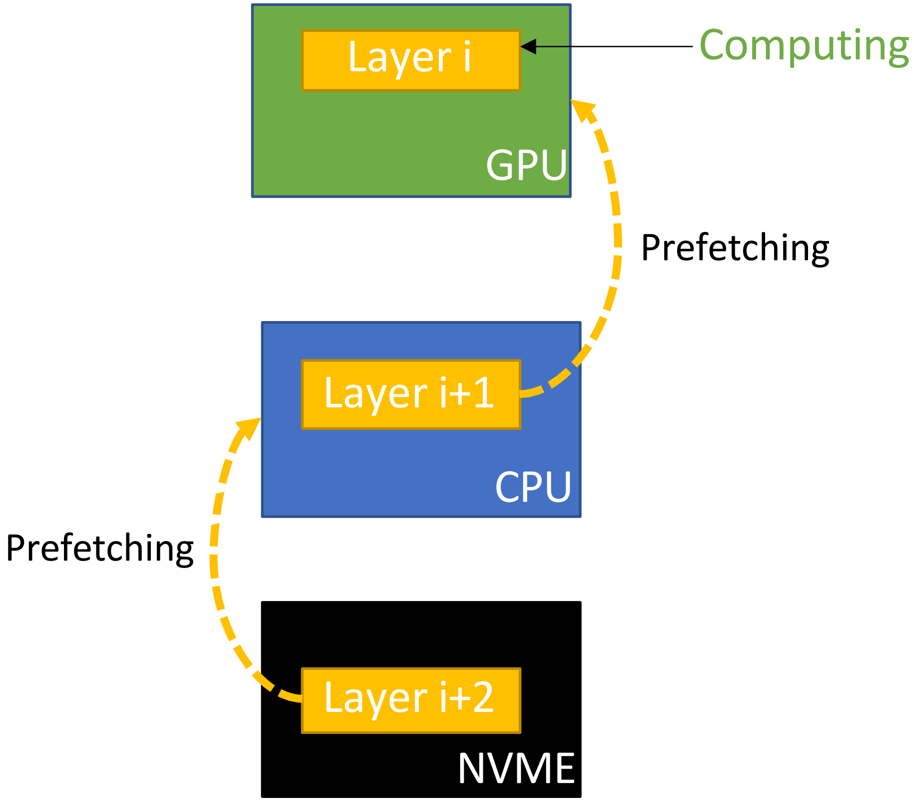
- Limited by GPU←→GPU bw
  - min (#gpus x host-device bw, gpu-gpu bw)
  - 70 GB/s

ZeRO Infinity

GPUs	Data Type	Required	NVMe memory	CPU Memory
1024	Params/Grads	70 GB/s	70 GB/s	70 GB/s
1024	Optimizer States	1500 GB/s	1792 GB/s	4096 GB/s
1024	Activations	4 GB/s	1.75GB/s	4GB/s

Overlap-Centric Design

- Data movement flow
  - NVMe -> CPU
  - CPU -> GPU
  - GPU <> GPU (all gather)
- Prefetch required data before consumption
  - While executing ith operator, fetch i + 1, i +2 …



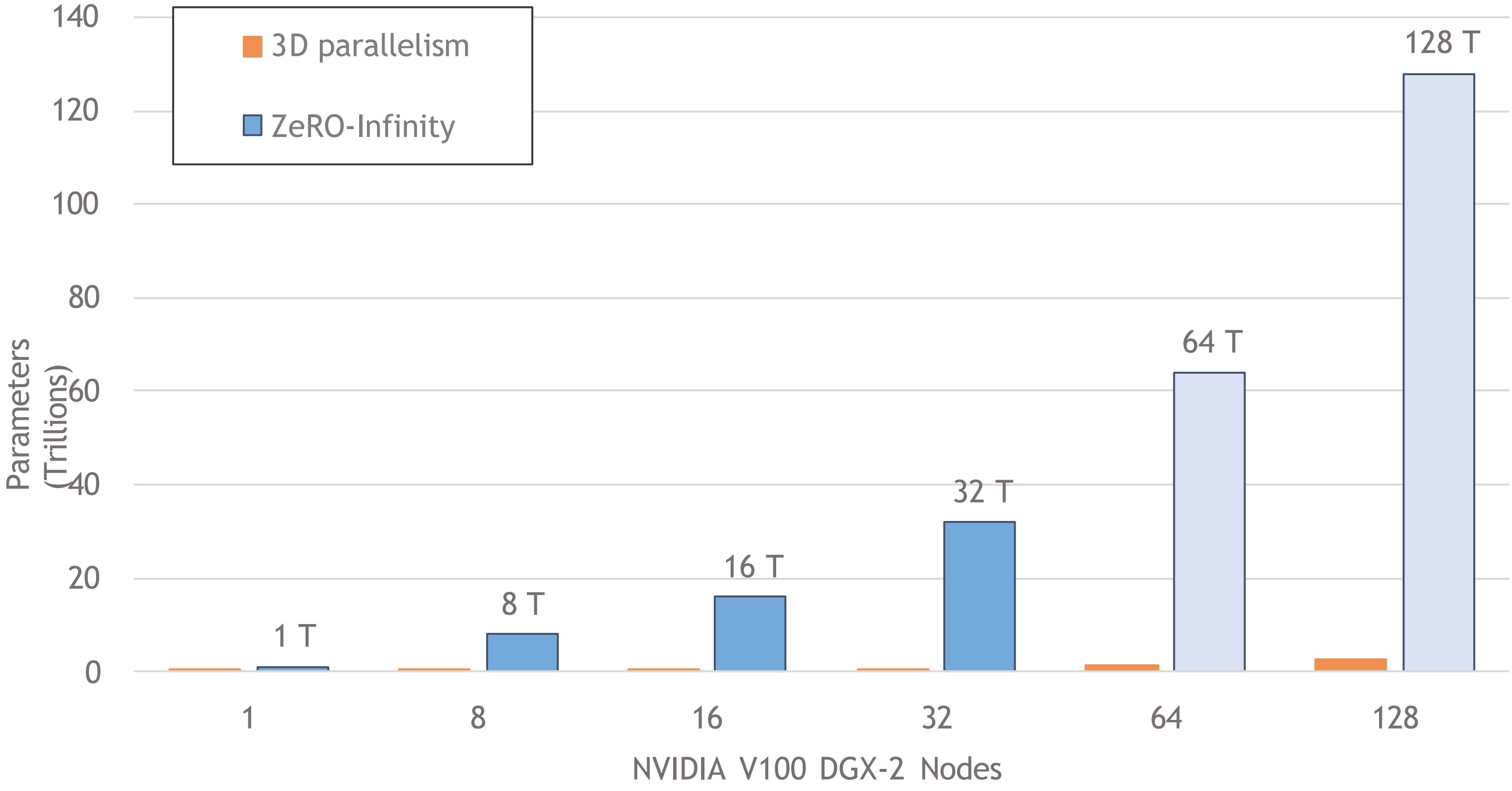
Overlapped layer prefetching during forward pass

Ease Inspired Implementation

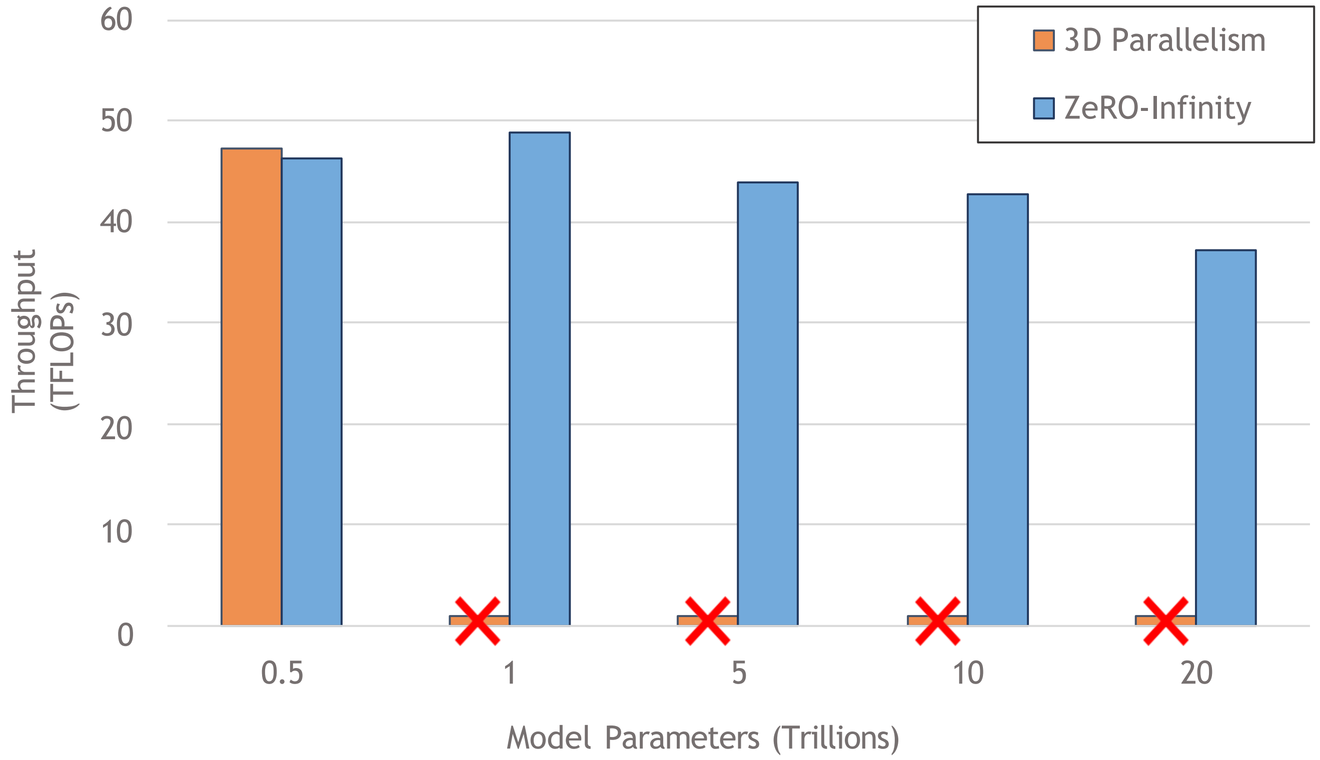
- Automatic Data Movement
  - Auto registration of all parameters
  - Intercepting parameter access to automate communication
- Automatic Model Partitioning during Initialization
  - Initializing models that are larger than GPU/CPU memory
  - Automatically partitioning parmaeters as they are created

Evaluation

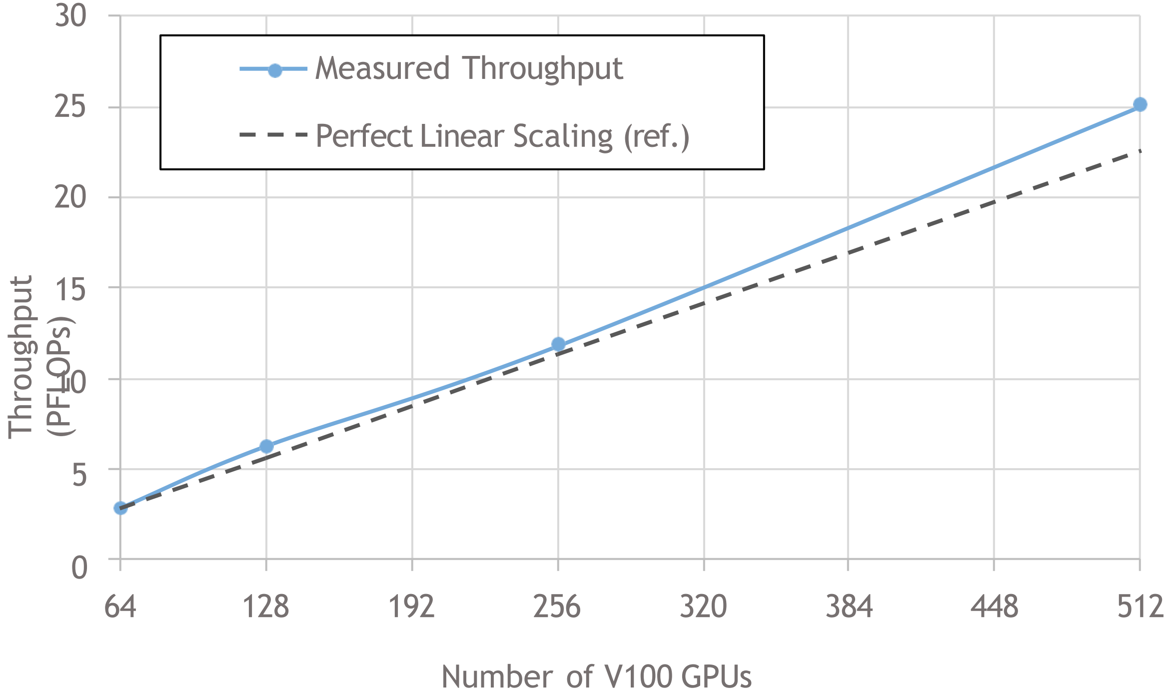
Massive model scale



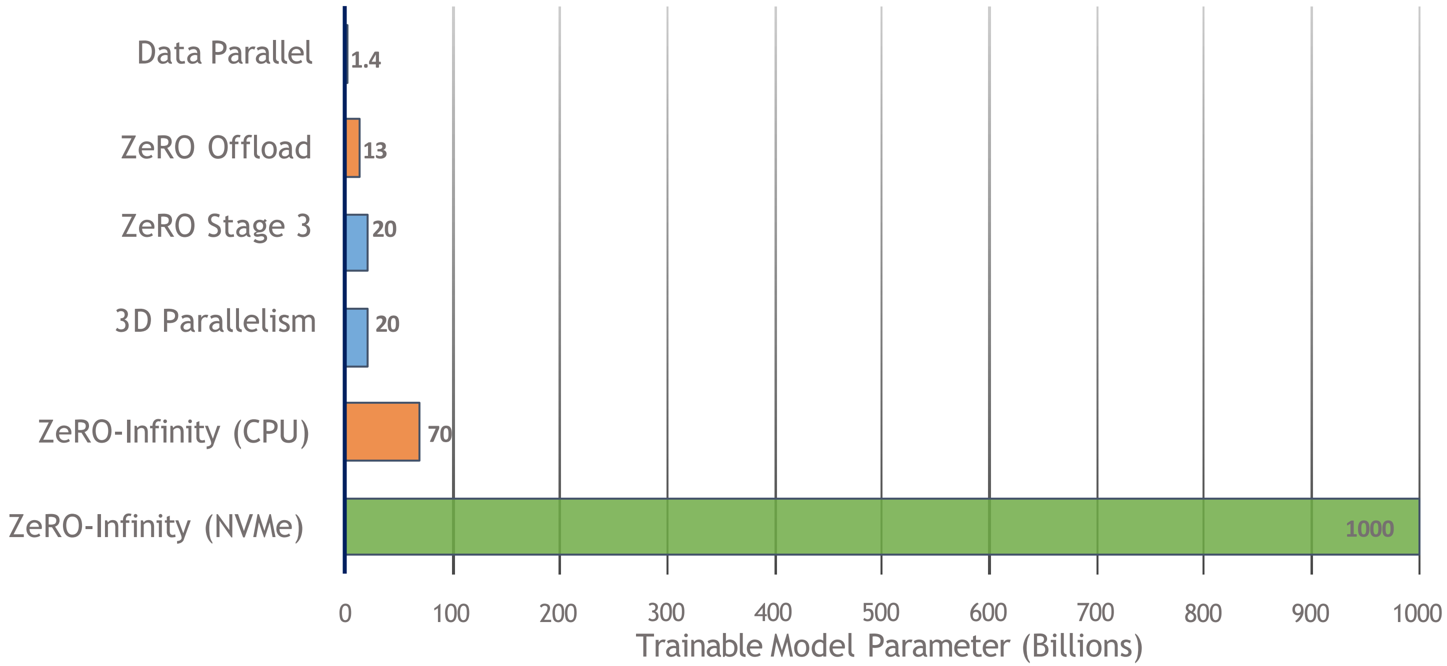
Excellent Efficiency



Super-linear Scalability

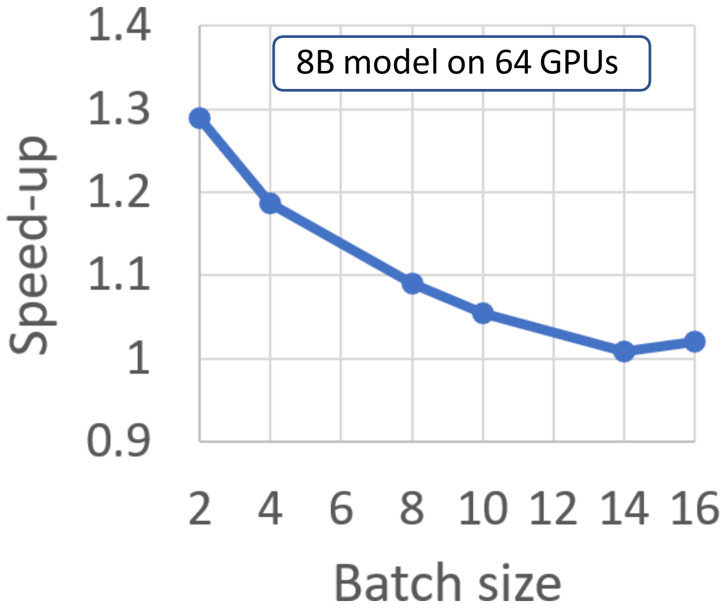


Democratizing Large Model Training



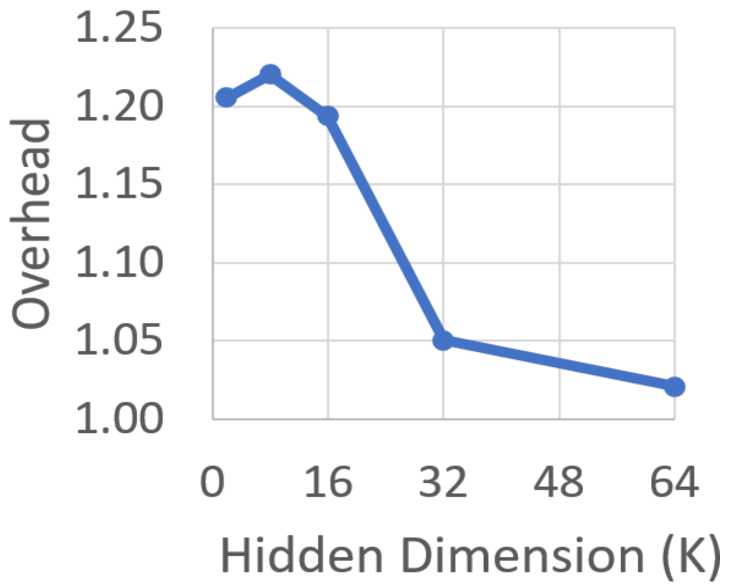
Impact of System Features on Performance

- Prefetching and Overlapping



More effective for smaller batch sizes

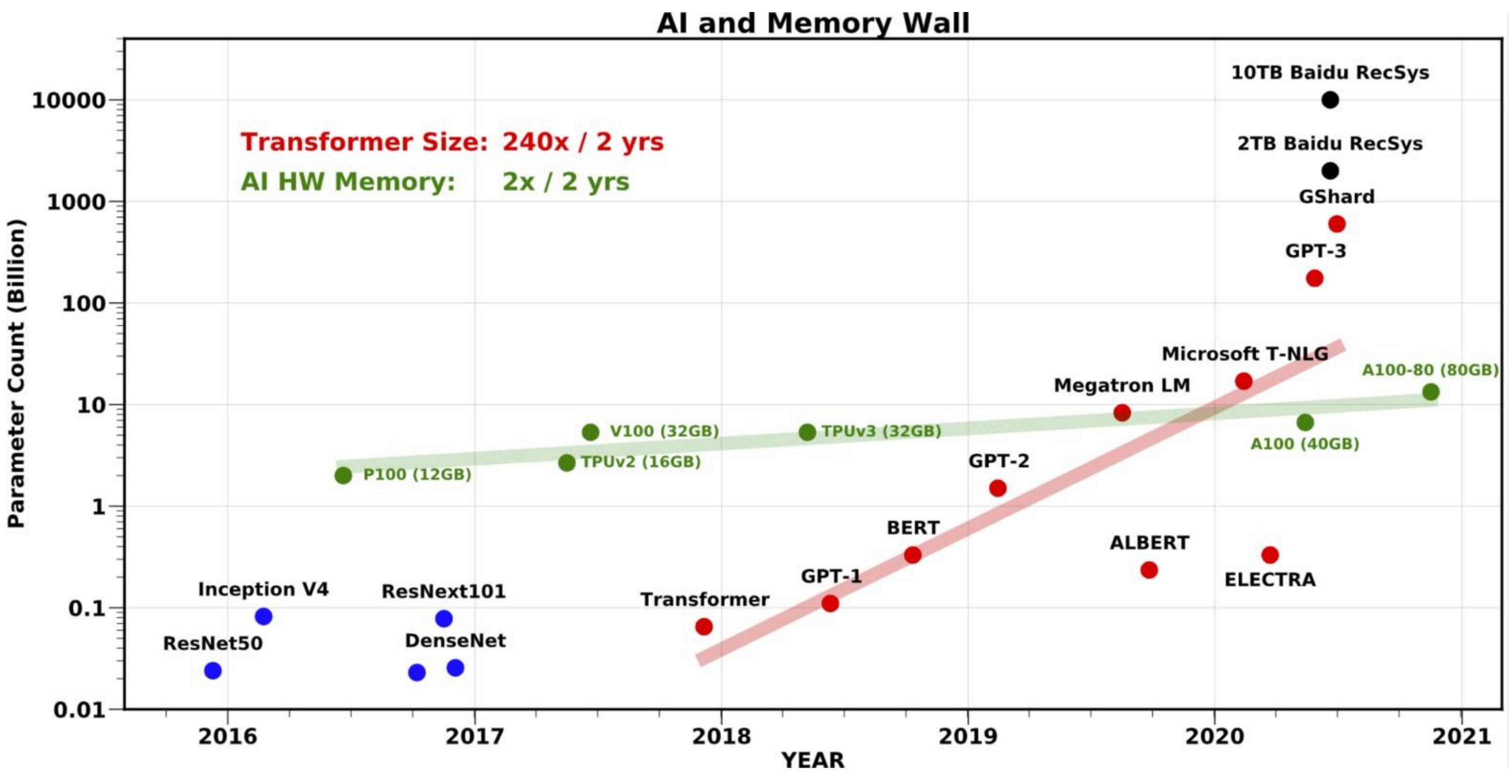
- Activation checkpoint offload



Overhead is negligible for large hidden dims

Large model training landscape today

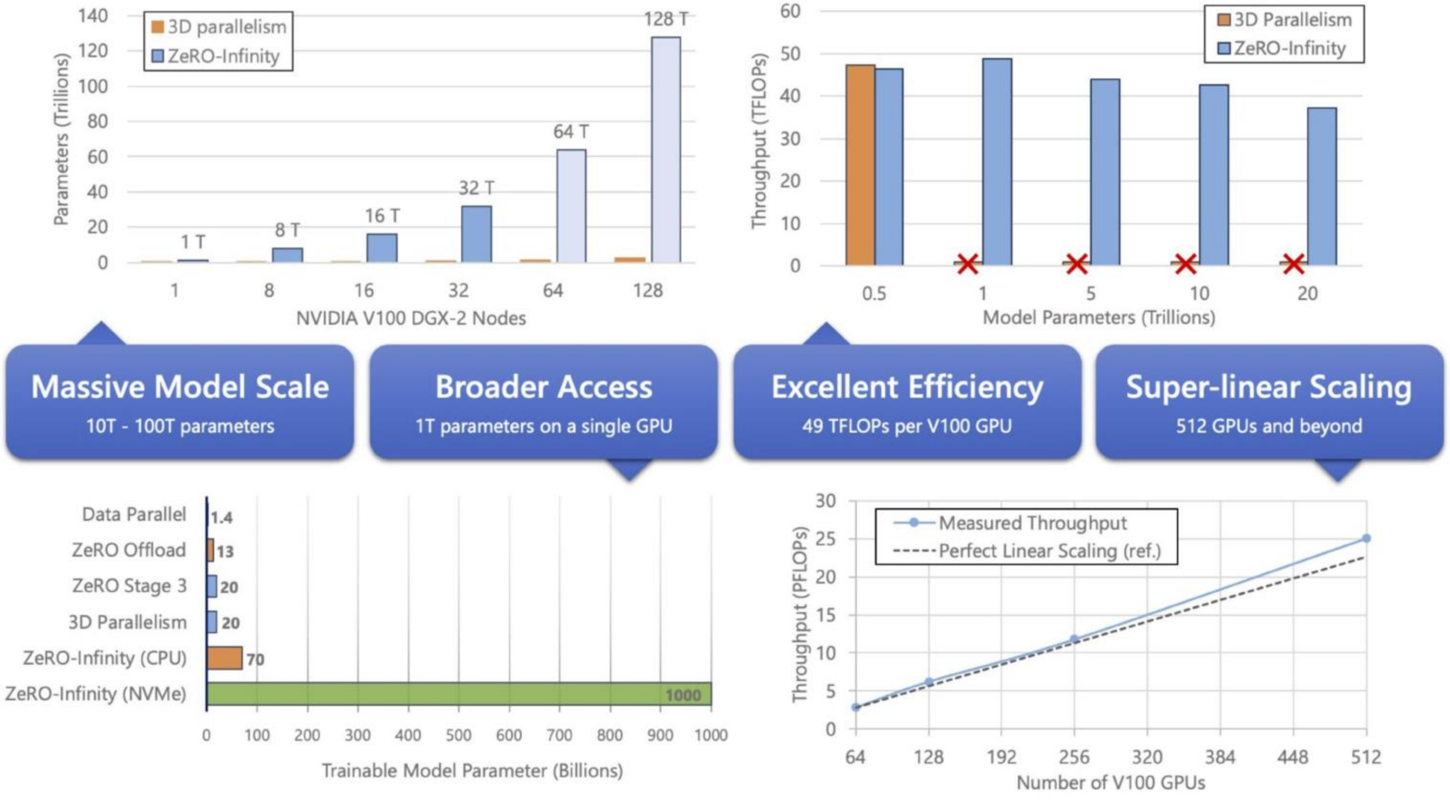
- **GPU Memory Wall**
  - 1T (10T) params: 800 (8K) V100 GPUs
  - How do we support the growth in model size?
- **Accessibility to large model training**
  - 256 GPUs to fine-tune GPT-3
  - Limited access to such resources
- **Model code refactoring**
  - Re-writing the model using 3D parallelism (tensor-slicing + pipeline parallelism)
  - Painful and error prone



*AI and Memory Wall. (This blogpost has been written in… | by Amir Gholami| riselab | Medium

Redefining the landscape with ZeRO-Infinity

- Beyond GPU Memory
  - 50x larger models
  - 32T params on 512 GPUs (instead of 25K)
- Broader access to large model training
  - GPT-3 sized fine-tuning on a single node/GPU (instead of 16 nodes)
- Excellent Throughput and Scalability
  - Comparable to 3D-parallelism
- Ease of Use
  - No model refactoring necessary



Plus and Minus

- Clear analysis on BW requirement
  - Clear illustration on why Offloading can achieve high efficiency
- Leveraging huge NVMe room
  - Much larger capacity for ML models

- Data placement
  - Activation memory on CPU memory
  - But other states, CPU becomes cache of NVMe
  - Can have some pre knowledge of hotness of data

Discussion

- CPU by passing?
  - NVMe -> CPU -> GPU
  - GPU direct accessing NVMe, greatly cutdown GPU fetching time