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
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

\[
\text{efficiency} = \frac{\text{compute\_time}}{\text{compute\_time} + \text{communication\_time}}
\]

\[
\text{compute\_time} = \frac{\text{total\_computation}}{\text{peak}_{tp}}
\]

\[
\text{communication\_time} = \frac{\text{total\_computation}}{\text{total\_data\_movement}}
\]

\[
\text{ait} = \frac{\text{total\_computation}}{\text{total\_data\_movement} \cdot \text{bw}}
\]

\[
\text{ait} \times \text{bw} = \frac{\text{total\_computation}}{\text{ait} \times \text{bw}}
\]

\[
\text{efficiency} = \frac{\text{ait} \times \text{bw}}{\text{ait} \times \text{bw} + \text{peak}_{tp}}
\]
Efficiency as a function of bandwidth

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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Overlap</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Params/Grads</td>
<td>Yes</td>
<td>70 GB/s</td>
</tr>
<tr>
<td>Optimizer States</td>
<td>No</td>
<td>1500 GB/s</td>
</tr>
<tr>
<td>Activations</td>
<td>Yes</td>
<td>1-4 GB/s</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>GPUs</th>
<th>Data Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>Params/Grads</td>
<td>70 GB/s</td>
</tr>
<tr>
<td>1024</td>
<td>Optimizer States</td>
<td>1500 GB/s</td>
</tr>
<tr>
<td>1024</td>
<td>Activations</td>
<td>1-4 GB/s</td>
</tr>
</tbody>
</table>

- 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 parameter across GPUs
- Send from NVMe to GPU in parallel

- Bandwidth Increases linearly with devices
  - $\#\text{gpus} \times \text{host-to-device bandwidth}$
  - 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 $\leftarrow \rightarrow$ GPU bw
  - $\min (\#\text{gpus} \times \text{host-device bw, gpu-gpu bw})$
  - 70 GB/s

---

<table>
<thead>
<tr>
<th>GPUs</th>
<th>Data Type</th>
<th>Required</th>
<th>NVMe memory</th>
<th>CPU Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>Params/Grads</td>
<td>70 GB/s</td>
<td>70 GB/s</td>
<td>70 GB/s</td>
</tr>
<tr>
<td>1024</td>
<td>Optimizer States</td>
<td>1500 GB/s</td>
<td>1792 GB/s</td>
<td>4096 GB/s</td>
</tr>
<tr>
<td>1024</td>
<td>Activations</td>
<td>4 GB/s</td>
<td>1.75GB/s</td>
<td>4GB/s</td>
</tr>
</tbody>
</table>
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 parameters as they are created
Evaluation
Massive model scale

![Graph showing parameters in billions for different NVIDIA V100 DGX-2 nodes, with 3D parallelism and ZeRO-Infinity representations.](image)
Excellent Efficiency

![Diagram showing throughput vs. model parameters for 3D Parallelism and ZeRO-Infinity.](image)
Super-linear Scalability

![Graph showing measured throughput and perfect linear scaling. The x-axis represents the number of V100 GPUs, ranging from 64 to 512. The y-axis represents throughput (PFLOPs), ranging from 0 to 30. The blue line represents measured throughput, and the black dashed line represents perfect linear scaling.](image)
Democratizing Large Model Training

- Data Parallel: 1.4
- ZeRO Offload: 13
- ZeRO Stage 3: 20
- 3D Parallelism: 20
- ZeRO-Infinity (CPU): 70
- ZeRO-Infinity (NVMe): 1000

Trainable Model Parameter (Billions)
Impact of System Features on Performance

• Prefetching and Overlapping

![Graph showing speed-up vs batch size](image)

- More effective for smaller batch sizes

• Activation checkpoint offload

![Graph showing overhead vs hidden dimension](image)

- 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...](https://medium.com) 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