Memory Footprint Reduction Techniques for DNN Training: An Overview

Episode III. Weight Placement for EX-Large Models

Bojian Zheng

2022/2/18
Outline

• EX-Large Models: Bring us better accuracy but also challenges.
• Existing Solutions: Data and Model Parallelism
• ZeRO-Infinity: Data-Parallelism-based Solution for EX-Large Training
Recall: A History of Prior Works

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Weight Pruning for Efficient Inference</strong></td>
<td><strong>Feature Maps Reduction for Efficient Training</strong></td>
<td><strong>Weight Placement for EX-Large Models</strong></td>
</tr>
</tbody>
</table>

**2015-2016**

3. S. Han et al. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. ICLR 2015

**2016-2019**


**2019-2020**


**Topic for Today!**
Recall: GPU Memory Allocations in DNN Training

- Major GPU memory consumers: **Model States** & Feature Maps
  - **Model States**: weights (①), gradients (②), optimizer states (③)

---

Why EX-Large Models?

• Rule of Thumb in Deep Learning:

Larger & Deeper Models ⇒ Better Model Accuracy\(^{[1, 2]}\)

• The amount of computation resources consumed by a job can reject the problem scale and may also indicate the commercial value of the workload\(^{[3]}\).

But at what cost?

---

Challenges with Training EX-Large Models

• BERT[1]
  • State-of-the-Art Natural Language Processing Model (2018)
  • 340M parameters
  • Cannot fit into the GPU memory even with a batch size of 1[2].
  • Addressable with offloading[3], checkpointing[4] etc.

[2] https://github.com/google-research/bert
Challenges with Training EX-Large Models

• GPT-3[1]

  • State-of-the-Art Natural Language Processing Model (2020)
  • 175B parameters (470× LARGER)
  • 700 GB storage ≫ modern GPU memory capacity
  • 3×700 GB = 2.1 TB more for gradients & optimizer states

Challenges with Training EX-Large Models

• Multi-Interests\([1, 2, 3]\)
  • Industry-scale recommender
  • Encodes every object of interest into a vector, which assembles into an embedding table.
  • Up to 240 GB storage \(\gg\) modern GPU memory capacity
  • Cannot fit a single layer into the GPU memory!

Today’s DNN Training In A Nutshell
Data and Model Parallelism

• Partition into multiple devices? E.g., Data and/or Model Parallelism?

[Diagram showing a deep neural network with inputs X and W, and outputs Y, with RTX 3090 graphics cards distributed across the diagram, and an equation $P[\text{Cool Dog}] = 100\%$.]
Data Parallelism

• Partitions the **data** into multiple devices.
Model Parallelism

• Partitions the **model** into multiple devices.
Data vs. Model Parallelism

Data Parallelism

+ Need to replicate the entire model. Challenges of the EX-large models persist!

Model Parallelism

− Need to significantly refactor the code for load-balancing.
− Hard to support models with complex dependencies.

Could we have the merits from both? i.e., generic support for EX-Large models with minor code changes?
ZeRO\textsuperscript{[1]}

• A solution that is based on data-parallelism.

• **Key Idea** Partitions model states among devices and use efficient communication primitives to gather them.

• However, hungry DNN models need much more memory!

\textsuperscript{[1]} S. Rajbhandari et al. ZeRO: Memory Optimizations toward Training Trillion Parameter Models. SC 2020
ZeRO-Infinity\textsuperscript{[1]}

- ZeRO\textsuperscript{[2]} + Infinity Offload Engine
- **Observation** GPUs are also accompanied with a CPU and NVMe SSDs

<table>
<thead>
<tr>
<th>Memory (TB)</th>
<th>GPUs</th>
<th>CPU</th>
<th>NVMe SSDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td>1.5</td>
<td>28</td>
</tr>
</tbody>
</table>

- **Key Idea** Partitions model states among (GPUs + CPU + SSD) and Keeps most of the states to the latter two.

\[1\] S. Rajbhandari et al. ZeRO: Memory Optimizations toward Training Trillion Parameter Models. SC 2020
Challenges

• **Challenge #1** How to efficiently get the weights from CPU/SSDs?

• **Challenge #2** How to handle cases when the weight of a single layer cannot fit into the GPU memory?
Key Idea #1. Weight Gathering

- **Challenge #1** How to efficiently get the weights from CPU/SSDs?
- **Optimization #1** \(\forall\) weight, each GPU worker holds a portion of it and **gathers** the rest from others.
  - Better utilization than having a single worker broadcast the weight to others.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>GPUs</th>
<th>Aggregate GPU Memory (TBs)</th>
<th>Aggregate CPU Memory (TBs)</th>
<th>Aggregate NVME Memory (TBs)</th>
<th>GPU Memory Bandwidth/GPU (GB/s)</th>
<th>GPU-GPU Bandwidth (GB/s)</th>
<th>CPU-GPU Memory Bandwidth/GPU (GB/s)</th>
<th>NVME-GPU Memory Bandwidth/GPU (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.032</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>N/A</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>0.5</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>150-200</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>2.0</td>
<td>6.0</td>
<td>112.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>8.0</td>
<td>24.0</td>
<td>448.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>32.0</td>
<td>96.0</td>
<td>1792.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Key Idea #1. Weight Gathering

• **Challenge #1**  How to efficiently get the weights from CPU/SSDs?

• **Optimization #1**  \( \forall \)weight, each GPU worker holds a portion of it and **gathers** the rest from others.

• Better utilization than having a single worker broadcast the weight to others.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>GPUs</th>
<th>Aggregate GPU Memory (TBs)</th>
<th>Aggregate CPU Memory (TBs)</th>
<th>Aggregate NVME Memory (TBs)</th>
<th>GPU Memory Bandwidth/GPU (GB/s)</th>
<th>GPU-GPU Bandwidth (GB/s)</th>
<th>CPU-GPU Memory Bandwidth/GPU (GB/s)</th>
<th>NVME-GPU Memory Bandwidth/GPU (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.032</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>N/A</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>0.5</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>150-200</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>2.0</td>
<td>6.0</td>
<td>112.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>8.0</td>
<td>24.0</td>
<td>448.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>32.0</td>
<td>96.0</td>
<td>1792.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Key Idea #1. Weight Gathering

- **Challenge #1** How to efficiently get the weights from CPU/SSDs?
- **Optimization #1** \(\forall\) weight, each GPU worker holds a portion of it and *gathers* the rest from others.
  
  - Better utilization than having a single worker *broadcast* the weight to others.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>GPUs</th>
<th>Aggregate GPU Memory (TBs)</th>
<th>Aggregate CPU Memory (TBs)</th>
<th>Aggregate NVME Memory (TBs)</th>
<th>GPU Memory Bandwidth/GPU (GB/s)</th>
<th>GPU-GPU Bandwidth (GB/s)</th>
<th>CPU-GPU Memory Bandwidth/GPU (GB/s)</th>
<th>NVME-GPU Memory Bandwidth/GPU (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.032</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>N/A</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>0.5</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>150-200</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>2.0</td>
<td>6.0</td>
<td>112.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>8.0</td>
<td>24.0</td>
<td>448.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>32.0</td>
<td>96.0</td>
<td>1792.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Key Idea #1. Weight Gathering

- **Challenge #1** How to efficiently get the weights from CPU/SSDs?
- **Optimization #1** ∀ weight, each GPU worker holds a portion of it and **gathers** the rest from others.

- Better utilization than having a single worker *broadcast* the weight to others.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>GPUs</th>
<th>Aggregate GPU Memory (TBs)</th>
<th>Aggregate CPU Memory (TBs)</th>
<th>Aggregate NVME Memory (TBs)</th>
<th>GPU Memory Bandwidth/GPU (GB/s)</th>
<th>GPU-GPU Bandwidth (GB/s)</th>
<th>CPU-GPU Memory Bandwidth/GPU (GB/s)</th>
<th>NVME-GPU Memory Bandwidth/GPU (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.032</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>N/A</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>0.5</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>150-200</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>2.0</td>
<td>6.0</td>
<td>112.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>8.0</td>
<td>24.0</td>
<td>448.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>32.0</td>
<td>96.0</td>
<td>1792.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
</tbody>
</table>

**Same effective bandwidth even if 1024 GPUs are reading in parallel**
Key Idea #1. Weight Gathering

• **Challenge #1**  How to efficiently get the weights from CPU/SSDs?

• **Optimization #1**  ∀ weight, each GPU worker holds a portion of it and **gathers** the rest from others.

• Better utilization than having a single worker **broadcast** the weight to others.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>GPUs</th>
<th>Aggregate GPU Memory (TBs)</th>
<th>Aggregate CPU Memory (TBs)</th>
<th>Aggregate NVME Memory (TBs)</th>
<th>GPU Memory Bandwidth/GPU (GB/s)</th>
<th>GPU-GPU Bandwidth (GB/s)</th>
<th>CPU-GPU Memory Bandwidth/GPU (GB/s)</th>
<th>NVME-GPU Memory Bandwidth/GPU (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.032</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>N/A</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>0.5</td>
<td>1.5</td>
<td>28.0</td>
<td>600-900</td>
<td>150-200</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>2.0</td>
<td>6.0</td>
<td>112.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>8.0</td>
<td>24.0</td>
<td>448.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>32.0</td>
<td>96.0</td>
<td>1792.0</td>
<td>600-900</td>
<td>60-100</td>
<td>4.1</td>
<td>1.6</td>
</tr>
</tbody>
</table>

4/1.6 TB/s (Gather) vs. 12/12 GB/s (Broadcast)
Key Idea #1. Pipelined Prefetch

- **Challenge #1** How to efficiently get the weights from CPU/SSDs?
- **Optimization #1** ∀weight, each GPU worker holds a portion of it and gathers the rest from others.

- **Optimization #2** Pipelined prefetch from SSD→CPU→GPU.

<table>
<thead>
<tr>
<th>i</th>
<th>SSD</th>
<th>CPU</th>
<th>GPU</th>
<th>Compute</th>
</tr>
</thead>
<tbody>
<tr>
<td>i + 1</td>
<td>SSD</td>
<td>CPU</td>
<td>GPU</td>
<td></td>
</tr>
<tr>
<td>i + 2</td>
<td>SSD</td>
<td>CPU</td>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>i + 3</td>
<td>SSD</td>
<td>SSD</td>
<td>SSD</td>
<td></td>
</tr>
</tbody>
</table>

Read the weight slice from SSD for layer $i + 3$ while layer $i$ is computing its results.
Key Idea #3. Tiled Computation

• **Challenge #2** How to handle cases when the weight of a single layer cannot fit into the GPU memory?

• **Key Idea** Some layers can evaluate their results in **tiles** and some do not need the whole weight for the full results\(^1\).
  
  • E.g., Dense Layer \(Y = XW^T\)

Key Results

The diagram shows the throughput (TFLOP/GPU) for different model sizes in trillions of parameters, comparing 3D Parallelism and ZeRO-Infinity. The x-axis represents model size, and the y-axis shows throughput. The bars indicate the performance for each model size:

- At 0.5 trillion parameters, the throughput is 47 TFLOPs for 3D Parallelism and 46 TFLOPs for ZeRO-Infinity.
- At 1 trillion parameters, the throughput is 49 TFLOPs for 3D Parallelism and 44 TFLOPs for ZeRO-Infinity.
- At 5 trillion parameters, the throughput is 44 TFLOPs for 3D Parallelism and 43 TFLOPs for ZeRO-Infinity.
- At 10 trillion parameters, the throughput is 43 TFLOPs for 3D Parallelism and 34 TFLOPs for ZeRO-Infinity.
- At 20 trillion parameters, the throughput is 34 TFLOPs for ZeRO-Infinity.

The performance of 3D Parallelism is consistently higher than ZeRO-Infinity across all model sizes shown in the diagram.
Key Results

Baseline with Data + Model Parallelism

Throughput (TFLOP/GPU)

<table>
<thead>
<tr>
<th>Model Size in Trillions of Parameters</th>
<th>3D Parallelism</th>
<th>ZeRO-Infinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>34</td>
<td></td>
</tr>
</tbody>
</table>

OOM
Key Results

Offloading to CPU/SSDs cause 2% performance degradation
Key Results

Support for models that have trillion parameters
Conclusions

• EX-Large Models: Bring us better accuracy but also challenges.

• Existing Solutions: Data and Model Parallelism
  • Can’t make EX-large training practical while being easily programmable.

• ZeRO-Infinity: Data-Parallelism-based Solution for EX-Large Training
  • Key Ideas: Weight Gathering, Pipelined Prefetch, Tiled Computation
  • Makes it possible to train models with trillion parameters with small performance overhead (2%).

• https://github.com/microsoft/DeepSpeed
Memory Footprint Reduction Techniques for DNN Training: An Overview

Episode III. Weight Placement for EX-Large Models

Bojian Zheng

2022/2/18
Why Not?

• Reduce the precision of the weight parameters$^1$? E.g., FP32→FP16?

• **Challenge**: Weights need to be in high-precision format to accommodate for incremental gradient updates.

• E.g., Type equation here.
$P[\text{Cool Dog}] = 100\%$
$P[\text{Cool Dog}] = 100\%$
ZeRO-Infinity\textsuperscript{[1]}

- ZeRO\textsuperscript{[2]} + Infinity Offload Engine
- Observation GPUs are accompanied with large CPU memory and
- **Key Idea** Partitions model states among workers (GPUs + CPU + NVMe SSD). Offload the states in the latter two
