GPipe and GShard

Kaixin Luo
Motivation

- The more Computational power you spent, the better model you get
GPipe’s Idea

• Commonly known parallelisms:
  • Data Parallelism
  • Model Parallelism

• Proposed:
  • Pipeline parallelism
What is Pipeline parallelism?
Formally speaking

- Partition mini batch into micro batches
- For forward pass:
  - Executes the model as usual
- For backward pass:
  - Sum the gradient of micro batch into mini batch gradient
Introducing GPipe Library

- Open source
- Implemented under Lingvo
GPipe Interface

- Any model could be treated as a sequence of layers
- Each layer has a forward function $f$, with weights $w$ and optional cost function $c$
- Given $K$ devices, the forward function $F$ for each partition is the combination of $f[i]$ to $f[j]$
- The back propagation could be computed by symbolic differentiation
- The cost of each partition could be the sum of the layer cost
Algorithm

• Given K as the number of accelerators:
  • The network will be partitioned into K “pieces”
  • Communication primitive inserted for each partition
  • The partition algorithm will minimize the variance of estimated communication cost.

• Given N as the size of the batch and M as the number of the micro batches:
  • Forward pass computes as normal
  • In the backward pass, sum up the gradient of micro batch into mini batch gradients
Performance

• Tested model:
  • AmoebaNet
  • Transformer

• Measures
  • Scalability
  • Efficiency
  • Communication cost
## Scalability

<table>
<thead>
<tr>
<th>NVIDIA GPUs (8GB each)</th>
<th>Naive-1</th>
<th>Pipeline-1</th>
<th>Pipeline-2</th>
<th>Pipeline-4</th>
<th>Pipeline-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmoebaNet-D (L, D)</td>
<td>(18, 208)</td>
<td>(18, 416)</td>
<td>(18, 544)</td>
<td>(36, 544)</td>
<td>(72, 512)</td>
</tr>
<tr>
<td># of Model Parameters</td>
<td>82M</td>
<td>318M</td>
<td>542M</td>
<td>1.05B</td>
<td>1.8B</td>
</tr>
<tr>
<td>Total Model Parameter Memory</td>
<td>1.05GB</td>
<td>3.8GB</td>
<td>6.45GB</td>
<td>12.53GB</td>
<td>24.62GB</td>
</tr>
<tr>
<td>Peak Activation Memory</td>
<td>6.26GB</td>
<td>3.46GB</td>
<td>8.11GB</td>
<td>15.21GB</td>
<td>26.24GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cloud TPUv3 (16GB each)</th>
<th>Naive-1</th>
<th>Pipeline-1</th>
<th>Pipeline-8</th>
<th>Pipeline-32</th>
<th>Pipeline-128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-L</td>
<td>3</td>
<td>13</td>
<td>103</td>
<td>415</td>
<td>1663</td>
</tr>
<tr>
<td># of Model Parameters</td>
<td>282.2M</td>
<td>785.8M</td>
<td>5.3B</td>
<td>21.0B</td>
<td>83.9B</td>
</tr>
<tr>
<td>Total Model Parameter Memory</td>
<td>11.7G</td>
<td>8.8G</td>
<td>59.5G</td>
<td>235.1G</td>
<td>937.9G</td>
</tr>
<tr>
<td>Peak Activation Memory</td>
<td>3.15G</td>
<td>6.4G</td>
<td>50.9G</td>
<td>199.9G</td>
<td>796.1G</td>
</tr>
</tbody>
</table>
### Efficiency

Table 2: Normalized training throughput using GPipe with different # of partitions $K$ and different # of micro-batches $M$ on TPUs. Performance increases with more micro-batches. There is an almost linear speedup with the number of accelerators for Transformer model when $M \gg K$. Batch size was adjusted to fit memory if necessary.

<table>
<thead>
<tr>
<th></th>
<th>TPU</th>
<th>AmoebaNet</th>
<th>Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$K = 2$</td>
<td>$K = 4$</td>
<td>$K = 8$</td>
</tr>
<tr>
<td>$M = 1$</td>
<td>1</td>
<td>1.13</td>
<td>1.38</td>
</tr>
<tr>
<td>$M = 4$</td>
<td>1.07</td>
<td>1.26</td>
<td>1.72</td>
</tr>
<tr>
<td>$M = 32$</td>
<td>1.21</td>
<td>1.84</td>
<td>3.48</td>
</tr>
</tbody>
</table>
Communication Cost

Table 3: Normalized training throughput using GPipe on GPUs without high-speed interconnect.

<table>
<thead>
<tr>
<th>GPU</th>
<th>AmoebaNet</th>
<th>Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K ) = 2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>( M = 32 )</td>
<td>1</td>
<td>1.7</td>
</tr>
</tbody>
</table>
## Test result for AmoebaNet

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Train</th>
<th># Test</th>
<th># Classes</th>
<th>Accuracy (%)</th>
<th>Previous Best (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet-2012</td>
<td>1,281,167</td>
<td>50,000</td>
<td>1000</td>
<td>84.4</td>
<td>83.9 [12] (85.4* [27])</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>50,000</td>
<td>10,000</td>
<td>10</td>
<td>99.0</td>
<td>98.5 [26]</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>50,000</td>
<td>10,000</td>
<td>100</td>
<td>91.3</td>
<td>89.3 [26]</td>
</tr>
<tr>
<td>Stanford Cars</td>
<td>8,144</td>
<td>8,041</td>
<td>196</td>
<td>94.6</td>
<td>94.8* [26]</td>
</tr>
<tr>
<td>Oxford Pets</td>
<td>3,680</td>
<td>3,369</td>
<td>37</td>
<td>95.9</td>
<td>93.8* [29]</td>
</tr>
<tr>
<td>Food-101</td>
<td>75,750</td>
<td>25,250</td>
<td>101</td>
<td>93.0</td>
<td>90.4* [30]</td>
</tr>
<tr>
<td>FGVC Aircraft</td>
<td>6,667</td>
<td>3,333</td>
<td>100</td>
<td>92.7</td>
<td>92.9* [31]</td>
</tr>
<tr>
<td>Birdsnap</td>
<td>47,386</td>
<td>2,443</td>
<td>500</td>
<td>83.6</td>
<td>80.2* [32]</td>
</tr>
</tbody>
</table>
Test result for Transformer
GPipe is not panacea

• Need a smart partition for micro batch, sparse or imbalanced partition will hurt the overall performance
• Bubble time is an issue when the data set is too small (M>4K).
• The model partition is not flexible when the model is complex (GShard)
GShard
What is sharding?

• In database, it is breaking big table into pieces and store them in different place.
• But, how about neural networks?
Motivation

- scaling a model that is already big enough.
Challenges for scaling

• Architecture support
• Computation cost vs model size
• Model representation
Proposed design principle

• SPMD XLA compilers
• Sublinear scaling for model design
• Model Abstraction
SPMD Compiler

(a) MPMD Partition

(b) SPMD Partition
Sublinear model design
Model: Transformer with MoE Layer
Mixture of Expert Layer

• A group of parallel feed forward neural networks.
• The gating algorithm will dispatch the weight.
Algorithm 1: Group-level top-2 gating with auxiliary loss

**Data:** $x_S$, a group of tokens of size $S$

**Data:** $C$, Expert capacity allocated to this group

**Result:** $G_{S,E}$, group combine weights

**Result:** $l_{aux}$, group auxiliary loss

1. $c_E \leftarrow 0$  \quad \text{▷ gating decisions per expert}
2. $g_{S,E} \leftarrow \text{softmax}(wg \cdot x_S)$  \quad \text{▷ gates per token per expert, $wg$ are trainable weights}
3. $m_E \leftarrow \frac{1}{S} \sum_{s=1}^{S} g_{s,E}$  \quad \text{▷ mean gates per expert}
4. **for** $s \leftarrow 1$ **to** $S$ **do**
5. \quad $g_1, e_1, g_2, e_2 = \text{top}_2(g_{s,E})$  \quad \text{▷ top-2 gates and expert indices}
6. \quad $g_1 \leftarrow g_1 / (g_1 + g_2)$  \quad \text{▷ normalized $g_1$}
7. \quad $c \leftarrow c_{e_1}$  \quad \text{▷ position in $e_1$ expert buffer}
8. \quad **if** $c_{e_1} < C$ **then**
9. \quad \quad $G_{s,e_1} \leftarrow g_1$  \quad \text{▷ $e_1$ expert combine weight for $x_s$}
10. \quad **end**
11. \quad $c_{e_1} \leftarrow c + 1$  \quad \text{▷ incrementing $e_1$ expert decisions count}
12. **end**
13. $l_{aux} = \frac{1}{E} \sum_{e=1}^{E} c_E \cdot m_e$  \quad \text{▷ top-2 gates and expert indices}
14. **for** $s \leftarrow 1$ **to** $S$ **do**
15. \quad $g_1, e_1, g_2, e_2 = \text{top}_2(g_{s,E})$  \quad \text{▷ top-2 gates and expert indices}
16. \quad $g_2 \leftarrow g_2 / (g_1 + g_2)$  \quad \text{▷ normalized $g_2$}
17. \quad $\text{rnd} \leftarrow \text{uniform}(0, 1)$  \quad \text{dispatch to second-best expert with probability $\propto 2 \cdot g_2$}
18. \quad $c \leftarrow c_{e_2}$  \quad \text{▷ position in $e_2$ expert buffer}
19. \quad **if** $c < C \land 2 \cdot g_2 > \text{rnd}$ **then**
20. \quad \quad $G_{s,e_2} \leftarrow g_2$  \quad \text{▷ $e_2$ expert combine weight for $x_s$}
21. \quad **end**
22. \quad $c_{e_2} \leftarrow c + 1$
23. **end**
Algorithm 2: Forward pass of the Positions-wise MoE layer. The underscored letter (e.g., G and E) indicates the dimension along which a tensor will be partitioned.

1. gates = softmax(einsum("GSM,ME->GSE", inputs, wg))
2. combine_weights, dispatch_mask = Top2Gating(gates)
3. dispatched_expert_inputs = einsum("GSEC,GSM->EGCM", dispatch_mask, reshaped_inputs)
4. h = einsum("EGCM,EMH->EGCH", dispatched_expert_inputs, wi)
5. h = relu(h)
6. expert_outputs = einsum("EGCH,EHM->GECM", h, wo)
7. outputs = einsum("GSEC,GECM->GSM", combine_weights, expert_outputs)
Shallow dip about einsum

- **Matrix multiplication:**
  - `Einsum("ab,ba->aa", mat1, mat2)`

- **Matrix Dot:**
  - `Einsum("ab,ab->ab", mat1, mat2)`

- **Matrix transpose:**
  - `Einsum("ab->ba", mat1)`

- **Matrix sum:**
  - `Einsum("ab->", mat1)`
FLOPS Analysis

• Assumption:
  • Given G as the group number, D as the device number, E as the expert number, the rest are constances.
  • number of tokens per device \( \frac{N}{D} = O(1) \) is constant
  • \( G = O(D) \), \( S = O(1) \) and \( N = O(GS) = O(D) \)
  • \( M = O(1) \), \( H = O(1) \)
  • \( E = O(D) \)
  • \( C = O(2S/E) = O(1/D) \), \( D < S \) and is a positive integer
FLOPS Analysis

$$FLOPS_{\text{Softmax}} + FLOPS_{\text{Top2Gating}} + FLOPS_{\text{DispatchCombine}} + FLOPS_{\text{FFN}} =$$

$$O(GSME) + O(GSEC) + O(GSMEC) + O(EGCHM) =$$

$$O(D \cdot 1 \cdot 1 \cdot D) + O(D \cdot 1 \cdot D \cdot \frac{1}{D}) + O(D \cdot 1 \cdot 1 \cdot D \cdot \frac{1}{D}) + O(D \cdot D \cdot \frac{1}{D} \cdot 1 \cdot 1) =$$

$$O(D^2) + O(D) + O(D) + O(D)$$
GShard APIs

• Replicate(tensor): replicate the tensor across partitions
• Split(tensor, split_dimension, num_partitions): split tensor according to split_dimensions into num_partitions portions
• Shard(tensor, device_assignment): generalized split and allowing multi-dimension split and specifying the placement of each dimension
# MoE forward computation using GShard

```python
# Partition inputs along group (G) dim.
+ inputs = split(inputs, 0, D)

# Replicate the gating weights
+ wg = replicate(wg)

gates = softmax(einsum("GSM,ME->GSE", inputs, wg))
combine_weights, dispatch_mask = Top2Gating(gating_logits)
dispatched_expert_inputs = einsum(
    "GSEC,GSM->EGCM", dispatch_mask, reshaped_inputs)

# Partition dispatched inputs along expert (E) dim.
+ dispatched_expert_inputs = split(dispatched_expert_inputs, 0, D)

h = einsum("EGCM,EMH->EGCH", dispatched_expert_inputs, wi)
...```

...
GShard communication APIs

- **CollectivePermute:**
  - Change sharded tensor device order among partitions
- **AllGather**
  - Concatinates tensors
- **AllReduce**
  - Elementwise reduction among partitions
- **AllToAll**
  - Logically split the tensor according to the dimension and send to different participants, it is efficient in TPU device network
## Results

<table>
<thead>
<tr>
<th>Id</th>
<th>Model</th>
<th>Experts Per-layer</th>
<th>Experts total</th>
<th>TPU v3 Cores</th>
<th>Enc+Dec layers</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>MoE(2048E, 36L)</td>
<td>2048</td>
<td>36684</td>
<td>2048</td>
<td>36</td>
<td>600B</td>
</tr>
<tr>
<td>(2)</td>
<td>MoE(2048E, 12L)</td>
<td>2048</td>
<td>12228</td>
<td>2048</td>
<td>12</td>
<td>200B</td>
</tr>
<tr>
<td>(3)</td>
<td>MoE(512E, 36L)</td>
<td>512</td>
<td>9216</td>
<td>512</td>
<td>36</td>
<td>150B</td>
</tr>
<tr>
<td>(4)</td>
<td>MoE(512E, 12L)</td>
<td>512</td>
<td>3072</td>
<td>512</td>
<td>12</td>
<td>50B</td>
</tr>
<tr>
<td>(5)</td>
<td>MoE(128E, 36L)</td>
<td>128</td>
<td>2304</td>
<td>128</td>
<td>36</td>
<td>37B</td>
</tr>
<tr>
<td>(6)</td>
<td>MoE(128E, 12L)</td>
<td>128</td>
<td>768</td>
<td>128</td>
<td>12</td>
<td>12.5B</td>
</tr>
<tr>
<td></td>
<td>MoE(2048E, 60L)</td>
<td>2048</td>
<td>61440</td>
<td>2048</td>
<td>60</td>
<td>1T</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>Id</th>
<th>Model</th>
<th>Cores</th>
<th>Steps per sec.</th>
<th>Batch sz. (Tokens)</th>
<th>TPU core years</th>
<th>Training time (days)</th>
<th>BLEU avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>MoE(2048E, 36L)</td>
<td>2048</td>
<td>0.72</td>
<td>4M</td>
<td>22.4</td>
<td>4.0</td>
<td>44.3</td>
</tr>
<tr>
<td>(2)</td>
<td>MoE(2048E, 12L)</td>
<td>2048</td>
<td>2.15</td>
<td>4M</td>
<td>7.5</td>
<td>1.4</td>
<td>41.3</td>
</tr>
<tr>
<td>(3)</td>
<td>MoE(512E, 36L)</td>
<td>512</td>
<td>1.05</td>
<td>1M</td>
<td>15.5</td>
<td>11.0</td>
<td>43.7</td>
</tr>
<tr>
<td>(4)</td>
<td>MoE(512E, 12L)</td>
<td>512</td>
<td>3.28</td>
<td>1M</td>
<td>4.9</td>
<td>3.5</td>
<td>40.0</td>
</tr>
<tr>
<td>(5)</td>
<td>MoE(128E, 36L)</td>
<td>128</td>
<td>0.67</td>
<td>1M</td>
<td>6.1</td>
<td>17.3</td>
<td>39.0</td>
</tr>
<tr>
<td>(6)</td>
<td>MoE(128E, 12L)</td>
<td>128</td>
<td>2.16</td>
<td>1M</td>
<td>1.9</td>
<td>5.4</td>
<td>36.7</td>
</tr>
<tr>
<td>*</td>
<td>T(96L)</td>
<td>2048</td>
<td>-</td>
<td>4M</td>
<td>~235.5</td>
<td>~42</td>
<td>36.9</td>
</tr>
</tbody>
</table>

Table 3: Performance of MoE models with different number of experts and layers.
Reference:

• Huang et al., GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism
• Lepikhin et al., GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding
• https://www.youtube.com/watch?v=9s2cum25Kkc
• https://www.youtube.com/watch?v=1VdEw_mGjFk
Take Care and Thanks