Accelerating Distributed MoE Training and Inference with Lina

Jiamin Li\textsuperscript{1}, Yimin Jiang\textsuperscript{2}, Yibo Zhu, Cong Wang\textsuperscript{1}, Hong Xu\textsuperscript{2}

\textsuperscript{1}City University of Hong Kong, \textsuperscript{2}ByteDance Inc., \textsuperscript{3}The Chinese University of Hong Kong
Sparsely-Activated Mixture-of-Experts (MoE)

MoE architecture
An ensemble of experts.
Sparsely-Activated Mixture-of-Experts (MoE)

- **Sparsely-activated MoE**: each input selects just a few (1 or 2) experts for processing
- Benefit: sub-linear scaling of FLOPS with model size

MoE architecture
An ensemble of experts.

Figure credit to Anatomical and Functional Plasticity in Early Blind Individuals and the Mixture of Experts Architecture
Sparsely-Activated Mixture-of-Experts (MoE)

- **Sparsely-activated** MoE: each input selects just a few (1 or 2) experts for processing
- Benefit: sub-linear scaling of FLOPS with model size

Massive model parameters with constant computation cost.

Figure credit to Anatomical and Functional Plasticity in Early Blind Individuals and the Mixture of Experts Architecture
Potential of MoE in Transformer Models

- GLaM by Google
  - GLaM outperforms GPT-3 on 29 tasks

- DeepSpeed MoE models
  - Model quality: 6.7B-parameter dense = 1.3B-parameter MoE - 128
  - Training compute reduction of 5x

<table>
<thead>
<tr>
<th>cost</th>
<th>FLOPs / token (G)</th>
<th>GPT-3</th>
<th>GLaM</th>
<th>relative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train energy (MWh)</td>
<td>350</td>
<td>180</td>
<td>-48.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1287</td>
<td>456</td>
<td>-64.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>accuracy on average</th>
<th>Zero-shot</th>
<th>One-shot</th>
<th>Few-shot</th>
<th>GPT-3</th>
<th>GLaM</th>
<th>relative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>56.9</td>
<td>62.7</td>
<td>65.5</td>
<td>61.6</td>
<td>68.1</td>
<td>+10.2%</td>
</tr>
<tr>
<td></td>
<td>+10.2%</td>
<td>+6.3%</td>
<td>+6.3%</td>
<td>+4.4%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure credit to GLaM and DeepSpeed MoE.
MoE in Transformer-based Language Models

- Feed forward layers (FFN) are replaced with MoE layers.
- MoE layer = gate + experts
  - Expert: feed forward neural network
    => *same* architecture, *different* parameters
  - Gate: a *trainable* matrix to select expert for each data sample
    - Top-2 in training, Top-1 in inference
MoE in Transformer-based Language Models

- Feed forward layers (FFN) are replaced with MoE layers.
- MoE layer = gate + experts
  - Expert: feed forward neural network
    => *same* architecture, *different* parameters
  - Gate: a *trainable* matrix to select expert for each data sample
  - Top-2 in training, Top-1 in inference
MoE in Transformer-based Language Models

- Feed forward layers (FFN) are replaced with MoE layers.
- MoE layer = gate + experts
  - Expert: feed forward neural network
    - Same architecture, different parameters
  - Gate: a trainable matrix to select expert for each data sample
    - Top-2 in training, Top-1 in inference

Transformer block

Self Attention
Add & Norm
Gate
FFN0 FFN1 FFN2 FFN3
MoE
Add & Norm
MoE in Transformer-based Language Models

- Feed forward layers (FFN) are replaced with MoE layers.
- MoE layer = gate + experts
  - Expert: feed forward neural network => *same* architecture, *different* parameters
  - Gate: a *trainable* matrix to select expert for each data sample
    - Top-2 in training, Top-1 in inference
  - Load balancing loss during training: *even* distribution among experts.
Distributed MoE

• Hybrid parallelism:
  • Expert parallelism: each device hosts one unique expert
  • Data parallelism: replicate non-expert parameter on each device
Distributed MoE

• Hybrid parallelism:
  • Expert parallelism: each device hosts one unique expert
  • Data parallelism: replicate non-expert parameter on each device
Distributed MoE

- Hybrid parallelism:
  - Expert parallelism: each device hosts one unique expert
  - Data parallelism: replicate non-expert parameter on each device

- All-to-all communication
  - 1st: send data samples to experts.
  - 2nd: restore data samples back
Distributed MoE

• Hybrid parallelism:
  • Expert parallelism: each device hosts one unique expert
  • Data parallelism: replicate non-expert parameter on each device

• All-to-all communication
  • 1st: send data samples to experts.
  • 2nd: restore data samples back
Distributed MoE

• Hybrid parallelism:
  • Expert parallelism: each device hosts one unique expert
  • Data parallelism: replicate non-expert parameter on each device

• All-to-all communication
  • 1st: send data samples to experts.
  • 2nd: restore data samples back
  • Same data transfer size
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training (ms)</th>
<th>Inference (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All-to-all</td>
<td>Ratio</td>
</tr>
<tr>
<td>#Layers &amp; Params</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12L + 117M</td>
<td>259</td>
<td>36.7%</td>
</tr>
<tr>
<td>24L + 233M</td>
<td>589</td>
<td>35.4%</td>
</tr>
<tr>
<td>36L + 349M</td>
<td>979</td>
<td>38.2%</td>
</tr>
<tr>
<td>12L + 419M</td>
<td>333</td>
<td>39.5%</td>
</tr>
<tr>
<td>24L + 838M</td>
<td>715</td>
<td>37.6%</td>
</tr>
<tr>
<td>36L + 1.2B</td>
<td>1145</td>
<td>36.8%</td>
</tr>
</tbody>
</table>
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

In MoE Transformer:
• Attention -> a complete sequence
• FFN expert -> one single token
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

In MoE Transformer:

• Attention -> a complete sequence
• FFN expert -> one single token
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

In MoE Transformer:
• Attention -> a complete sequence
• FFN expert -> one single token

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.4%</td>
<td></td>
</tr>
<tr>
<td>26.2%</td>
<td></td>
</tr>
<tr>
<td>28.3%</td>
<td></td>
</tr>
<tr>
<td>32.5%</td>
<td></td>
</tr>
<tr>
<td>31.7%</td>
<td></td>
</tr>
<tr>
<td>27.4%</td>
<td></td>
</tr>
</tbody>
</table>
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

In MoE Transformer:
• Attention -> a complete sequence
• FFN expert -> one single token
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

In MoE Transformer:
- Attention -> a complete sequence
- FFN expert -> one single token

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.4%</td>
<td></td>
</tr>
<tr>
<td>26.2%</td>
<td></td>
</tr>
<tr>
<td>28.3%</td>
<td></td>
</tr>
<tr>
<td>32.5%</td>
<td></td>
</tr>
<tr>
<td>31.7%</td>
<td></td>
</tr>
<tr>
<td>27.4%</td>
<td></td>
</tr>
</tbody>
</table>

Block $i$

Attention

Gate

FFN 0 FFN 1 FFN 2 FFN 3

ATC is good

Attention

...
Distributed MoE is not efficient

All-to-all takes an average of 34.1% of the step time.

In MoE Transformer:
- Attention -> a complete sequence
- FFN expert -> one single token

To restore to a sequence, we must wait for the processing of all tokens.
All-to-all is the bottleneck

• Synchronous and blocking operation & large amounts of data transfer.
All-to-all is the bottleneck

- Synchronous and blocking operation & large amounts of data transfer.

![Diagram showing Forward Pass with Stream a and Stream b, gates, FFNs, combines, and All-to-all operations with time in ms (0, 3.1, 10.7, 12.1, 18.7, 20.3).]
All-to-all is the bottleneck

- Synchronous and blocking operation & large amounts of data transfer.

![Diagram showing stream a and stream b with Forward Pass, Gate, FFN, and Combine stages with timing in milliseconds](Image)
All-to-all is the bottleneck

- Synchronous and blocking operation & large amounts of data transfer.

Is it the only cause of all-to-all being the bottleneck?
All-to-all is the bottleneck

- Synchronous and blocking operation & large amounts of data transfer.

Is it the only cause of all-to-all being the bottleneck?

- MoE training and inference have their unique problems.
All-to-all is the bottleneck

- Synchronous and blocking operation & large amounts of data transfer.

Is it the only cause of all-to-all being the bottleneck?

- MoE training and inference have their unique problems.
  - Training has backward pass
  - Inference is purely workload-driven

![Diagram showing forward pass time and ratio for different experts.](image)
In backward pass,

• Allreduce: asynchronously aggregate *non-expert* gradients in data parallel.

• All-to-all: exchange token gradients to compute *expert* gradients.
In backward pass,
• Allreduce: asynchronously aggregate *non-expert* gradients in data parallel.
• All-to-all: exchange token gradients to compute *expert* gradients.

![Backward Pass Diagram]
In backward pass,

- **Allreduce**: asynchronously aggregate *non-expert* gradients in data parallel.
- **All-to-all**: exchange token gradients to compute *expert* gradients.

All-to-all is prolonged when it overlaps with allreduce and directly impacts step time.
In backward pass,

- **Allreduce**: asynchronously aggregate *non-expert* gradients in data parallel.
- **All-to-all**: exchange token gradients to compute *expert* gradients.

All-to-all is prolonged when it overlaps with allreduce and directly impacts step time.

Slowdown of all-to-all varies:

- Median: 2x; Maximum: ~4x
Expert Popularity in MoE Inference

- Inference: no load balancing constraints => expert selection is workload-driven, therefore, much more *biased*.
Expert Popularity in MoE Inference

- Inference: no load balancing constraints => expert selection is workload-driven, therefore, much more *biased*.

Devices with unpopular experts have to wait for those with popular experts.
Expert Popularity in MoE Inference

- Inference: no load balancing constraints => expert selection is workload-driven, therefore, much more *biased*.

Maximum idle time of the least popular expert => 29.4% of the inference time.

Devices with unpopular experts have to wait for those with popular experts.
Training: Challenges

Intuition: always prioritise all-to-all and avoid bandwidth sharing.

- Minimise the blocking period incurred by all-to-all
Intuition: always prioritise all-to-all and avoid bandwidth sharing.

• Minimise the blocking period incurred by all-to-all
Training: Challenges

Intuition: always prioritise all-to-all and avoid bandwidth sharing.

- Minimise the blocking period incurred by all-to-all

Backward pass

Challenges:

1. NCCL Communication primitives cannot be preempted.
2. No control knob to adjust resource sharing (GPU SM, network bandwidth...).
Training: Micro-op Scheduling

- Tensor Partitioning
  - Partition allreduce into micro-ops
  - Prioritise all-to-all whenever possible

Baseline
Training: Micro-op Scheduling

- Tensor Partitioning
  - Partition allreduce into micro-ops
  - Prioritise all-to-all whenever possible

Baseline

Prioritise all-to-all
Training: Micro-op Scheduling

- Tensor Partitioning
  - Partition allreduce into micro-ops
  - Prioritise all-to-all whenever possible

Baseline

- Partition all-to-all into micro-ops
- Pipelining computation and all-to-all
Training: Micro-op Scheduling

- Tensor Partitioning
  - Partition allreduce into micro-ops
  - Prioritise all-to-all whenever possible

Baseline

- Partition all-to-all into micro-ops
- Pipelining computation and all-to-all
Inference: Challenges

- MoE inference: no load balancing constraints \(\Rightarrow\) expert selection is workload-driven, therefore, much more *biased*.

![Graphs showing popularity distribution](image)

- Maximum idle time of the least popular expert \(\Rightarrow\) 29.4% of the inference time.

*Devices with unpopular experts have to wait for those with popular experts.*
Inference: Challenges

Q: How to deal with imbalance computation load?

- MoE inference: no load balancing constraints ⇒ expert selection is workload-driven, therefore, much more biased.

Maximum idle time of the least popular expert ⇒ 29.4% of the inference time.

Devices with unpopular experts have to wait for those with popular experts.
Q: How to deal with imbalance computation load?
Allocate resources based on expert popularity:
Popular experts => more resources.
Q: How to deal with imbalance computation load?
Allocate resources based on expert popularity:
Popular experts => more resources.

Token’s expert selection cannot be determined a priori to the actual gating computation.
Q: How to deal with imbalance computation load?
Allocate resources based on expert popularity:
Popular experts => more resources.

Token’s expert selection cannot be determined a priori to the actual gating computation.

Resource scheduling after gating network.

_Inefficient practice for latency-sensitive tasks!_

<table>
<thead>
<tr>
<th>Model &amp; Dataset</th>
<th>Layer</th>
<th>Top-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-XL &amp; Enwik8 (Text generation)</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Transformer-XL &amp; Enwik8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>BERT-Large &amp; WMT En-De (Translation)</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>BERT-Large &amp; WMT En-De</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>BERT-Large &amp; WMT En-De (Translation)</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>BERT-Large &amp; WMT En-De (Translation)</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>
Q: How to deal with imbalance computation load? Allocate resources based on expert popularity: Popular experts => more resources.

Token’s expert selection cannot be determined a priori to the actual gating computation.

Resource scheduling after gating network. *Inefficient practice for latency-sensitive tasks!*

How to achieve low-overhead resource scheduling to balance device load?
Inference: Pattern in Expert Selection

- Findings: similar tokens tend to be processed by the same or similar experts in each layer.
- Tokens selecting the same expert in layer $i$ tend to select the same expert again in layer $i + 1$. 
Inference: Pattern in Expert Selection

- Findings: similar tokens tend to be processed by the same or similar experts in each layer.
- Tokens selecting the same expert in layer $i$ tend to select the same expert again in layer $i + 1$. 

![Graph showing expert selection trend across layers](chart.png)
• Findings: similar tokens tend to be processed by the same or similar experts in each layer.

• Tokens selecting the same expert in layer i tend to select the same expert again in layer i + 1.

41.94% tokens when k is 1
54.59% tokens when k is 2
Inference: Pattern in Expert Selection

• Findings: similar tokens tend to be processed by the same or similar experts in each layer.

• Tokens selecting the same expert in layer i tend to select the same expert again in layer i + 1.

41.94% tokens when k is 1
54.59% tokens when k is 2

Exploit this pattern to estimate the overall expert popularity
Inference: Expert Popularity Estimation
Idea: Collect the expert selection distribution during training after the load balancing loss is minimised.
Inference: Expert Popularity Estimation

• Idea: Collect the expert selection distribution during training after the load balancing loss is minimised.

Tokens that select the same experts from layer $i - 1$ to layer $i$. Expert selection path $j$
Inference: Expert Popularity Estimation

- Idea: Collect the expert selection distribution during training after the load balancing loss is minimised.

Tokens that select the same experts from layer $i-l$ to layer $i$. Expert selection path $j$

Expert popularity distribution of path $j$
Inference: Expert Popularity Estimation

- Idea: Collect the expert selection distribution during training after the load balancing loss is minimised.

Top-$k$ Expert $e$ popularity: $\{P_{i+1}^{j(t)}\}$

Layers:
- Layer $i-2$: E0, E1, E2, E3
- Layer $i-1$: E0, E1, E2, E3
- Layer $i$: E0, E1, E2, E3
- Layer $i+1$: E0, E1, E2, E3

Expert selection path $j$:
- Tokens that select the same experts from layer $i - 1$ to layer $i$.

Expert popularity distribution of path $j$:
Inference: Two-phase Scheduling

• Phase 1:
  • Compute resource allocation for expert $e$ based on popularity estimation:

$$n_e = N \times \sum_{t=1}^{N_t} \frac{P_{j(t)}^{i+1}(e)}{N_t}$$

- No. of tokens in a batch
- No. of GPUs

- No. of tokens in a batch
- No. of GPUs
Inference: Two-phase Scheduling

- Phase 1: **Pipelined with model computation => nearly zero overhead**
  - Compute resource allocation for expert $e$ based on popularity estimation:

$$n_e = N \times \sum_{i=1}^{N_t} \frac{P_{j(t)}^{i+1}(e) / N_t}{N}$$

- No. of tokens in a batch
- No. of GPUs
Inference: Two-phase Scheduling

- Phase 1: **Pipelined with model computation => nearly zero overhead**
  - Compute resource allocation for expert \( e \) based on popularity estimation:
    \[
    n_e = N \times \sum_{i=1}^{N_t} \frac{P_{ij(t)}(e)}{N_t} / N_t
    \]
  - No. of tokens in a batch
  - No. of GPUs

- Phase 2:
  - Fine-tune the allocation with the actual expert selection.
    - Re-compute the allocation when the actual selection deviates significantly from the estimation.
Evaluation

• Testbed: Our testbed has four worker nodes. Each node has 4 Ampere A100 GPUs with 40GB memory and is equipped with 100Gbps InfiniBand.

• Every FFN layer in Transformer is replaced with the MoE layer.

• Training models:
  • Transformer-XL: a 24-layer encoder model.
  • BERT2GPT2: a 12-layer encoder-decoder model.
  • GPT-2: a 12-layer decoder model.

• Inference models:
  • Transformer-XL: text generation with Enwik8 test set.
  • BERT: a 12-layer decoder model for translation using WMT En-De test set.
Training step time speedup over Baseline (DeepSpeed) with different design choices.

- **Transformer-XL**
- **GPT-2**
- **BERT2GPT2**

Graphs show the step time speedup with different design choices:
- **Priority only**
- **Priority + Partition**
- **Priority + Partition + Pipeline**
MoE Layer in Training

MoE layer forward, backward, all-to-all running time speedup over Baseline.
Ideal: schedule resources assuming we have the prior knowledge of exact expert popularity.
MoE Layer in Inference

MoE layer running time speedup over Ideal.

All-to-all running time speedup over Ideal in selected layers.
Lina’s Contributions

- An in-depth empirical analysis of distributed MoE
  - Main causes for all-to-all to be the performance bottleneck in training and inference.
- [Training] A scheduler prioritises all-to-all over allreduce to improve its bandwidth and reduce its blocking period.
- [Inference] An estimation method of expert popularity to conduct two-phase resource scheduling.

Thanks!