Megatron-LM

Bryan Catanzaro, VP Applied Deep Learning Research
Incredible speed-ups take more than just powerful chips.

Full-stack invention: chips, systems, frameworks, compilers, algorithms, apps.

Entire stack must be co-optimized.

This is mostly software work.
The Soul of Megatron-LM
https://github.com/NVIDIA/Megatron-LM

• Today’s NLP models require a few million dollars to train so we must have:
  
  • **Efficiency:** we measure it as the percentage of theoretical peak FLOPs of a processor
    • Best ROI
    • Up to 56% MFU for Megatron-LM

  • **Scalability:** Efficient scaling of both model size (weak scaling) and number of GPUs (strong scaling)
    • Biggest model & dataset

  • **Simplicity:** Simple yet efficient algorithms mostly in Python, with no fancy compiler
    • Model innovation & agility
Data and Model Parallelism

Data Parallelism (DP)

Device 1

Device 2

$n$ copies of model parameters

Model Parallelism (MP)

Device 1

Device 2

Tensor MP

Device 1

Device 2

Pipeline MP

Single copy of model parameters
Efficiency and Scalability

- Achieve scalability using data and model parallelism
  - Model parallelism:
    - Tensor parallelism
    - Sequence parallelism
    - Pipeline parallelism
- Challenge: how to achieve efficiency at scale

Almost linear scaling for models from 1B to 1T parameters (3 orders of magnitude) across 32 to 3K GPUs (2 orders of magnitude)
Simplicity

- The Megatron-LM project is built in PyTorch
- I love compilers! I think the world needs awesome compilers for AI
- But we have an urgent mission:
  - Accelerate Transformers
- Automatic parallel compilers for AI are hard
- We are doing this all by hand
- This shows us Speed-of-light
- Space is moving quickly
  - New ideas all the time
Model Parallel MLP

- **MLP:**
  
  \[ Y = \text{GeLU}(X A) \]
  \[ Z = \text{Dropout}(Y B) \]

- **Approach 1:** split X column-wise and A row-wise:
  
  \[ X = [X_1, X_2] \quad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \quad Y = \text{GeLU}(X_1 A_1 + X_2 A_2) \]
  
  - Before GeLU, we will need a synchronization point

- **Approach 2:** split A column-wise:
  
  \[ A = [A_1, A_2] \quad [Y_1, Y_2] = [\text{GeLU}(X A_1), \text{GeLU}(X A_2)] \]
  
  - no synchronization is required
A column-wise, B row-wise: \( \frac{1}{2} \) the communication

\[ Y = \text{GeLU}(XA) \]

\[ A = [A_1, A_2] \]

\[ Z = \text{Dropout}(YB) \]

\[ B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \]

\( f \) and \( g \) are conjugate, \( f \) is identity operator in the forward pass and all-reduce in the backward pass while \( g \) is all-reduce in forward and identity in backward.
Pipeline Parallelism

- Divides a batch size into micro-batches to keep the pipeline pressurized

- However, due to synchronous gradient updates, we have idle times (bubble) at the beginning and end of each iteration
## Interleaving Pipeline Schedule

<table>
<thead>
<tr>
<th>Device 1</th>
<th></th>
<th>Device 2</th>
<th></th>
<th>Device 3</th>
<th></th>
<th>Device 4</th>
<th></th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4</td>
<td>1 2 3</td>
<td>1 2 3 4</td>
<td>1 2 3</td>
<td>1 2 3 4</td>
<td>1 2 3</td>
<td>1 2 3 4</td>
<td>1 2 3</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>5 2 6 3</td>
<td>4 7 8 5</td>
<td>6 4 7 8</td>
<td>5 7 8 6</td>
<td>4 8 7 6</td>
<td>5 7 8 6</td>
<td>4 8 7 6</td>
<td>5 7 8 6</td>
<td></td>
</tr>
<tr>
<td>4 7 8 5</td>
<td>5 6 7 8</td>
<td>4 5 7 8</td>
<td>3 5 7 8</td>
<td>4 5 7 8</td>
<td>4 5 7 8</td>
<td>4 5 7 8</td>
<td>4 5 7 8</td>
<td></td>
</tr>
<tr>
<td>6 7 8 5</td>
<td>6 7 8 5</td>
<td>7 6 8 5</td>
<td>6 7 8 5</td>
<td>6 7 8 5</td>
<td>6 7 8 5</td>
<td>6 7 8 5</td>
<td>6 7 8 5</td>
<td></td>
</tr>
<tr>
<td>7 8 5 6</td>
<td>7 8 5 6</td>
<td>8 7 5 6</td>
<td>7 8 5 6</td>
<td>7 8 5 6</td>
<td>7 8 5 6</td>
<td>7 8 5 6</td>
<td>7 8 5 6</td>
<td></td>
</tr>
<tr>
<td>8 5 6 7</td>
<td>8 5 6 7</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td></td>
</tr>
<tr>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td></td>
</tr>
<tr>
<td>10 11 12</td>
<td>10 11 12</td>
<td>10 11 12</td>
<td>10 11 12</td>
<td>10 11 12</td>
<td>10 11 12</td>
<td>10 11 12</td>
<td>10 11 12</td>
<td></td>
</tr>
<tr>
<td>11 12 12</td>
<td>11 12 12</td>
<td>11 12 12</td>
<td>11 12 12</td>
<td>11 12 12</td>
<td>11 12 12</td>
<td>11 12 12</td>
<td>11 12 12</td>
<td></td>
</tr>
</tbody>
</table>

Assign multiple stages to each device (interleaved schedule)

- **Forward Pass:** Green
- **Backward Pass:** Blue
Interleaving Schedule Results

- Interleaving more effective at small batch sizes
- Good for strong scaling

175B GPT-3 model on 96 GPUs (no data parallelism)
Sequence Parallelism

- Activations require a substantial amount of memory for large models.
- Tensor parallelism can only reduce parts of activations memory (dropout and layernorms are duplicated)
- Standard full activation recomputation introduces 30-40% computational overhead

Red line shows A100/H100 memory

Required memory for tensor + pipeline parallelism
Solution

- Sequence parallelism + Selective activation recomputation

56.3% MFU for 1T parameter model on 512 A100 GPUs

Percentage of required activation memory compared to the tensor+pipeline parallel baseline.

Per-layer breakdown; baseline is the case with no activation recomputation or sequence parallelism
End-to-end Results: Measured Strong Scaling

32x increase in number of GPUs for fixed model size and batch size

More work to do here
And Beyond
Conclusion

• Language models are the biggest compute challenge of our time
• Megatron-LM is a research project for big transformers
• Megatron technologies productized as part of NVIDIA NeMo
• Current work focuses on multimodality and more complex training setups

• A golden age for AI systems: so much more than chips