Data-, model- and pipeline-parallelism and memory efficiency in DNN training

Henk Dreuning

University of Amsterdam, Vrije Universiteit Amsterdam

A16 - ASCI Winter School on Efficient Deep Learning
24-11-2021
Table of contents

DNN training process
The problem: DL model sizes
Multi-GPU and distributed training
  • Data- and model-parallelism
  • Hybrid- and pipeline-parallelism
Pipeline-parallelism implementations
  • PipeDream(-2BW), GPipe, DAPPLE
DNN training

![Deep neural network](https://www.ibm.com/cloud/learn/neural-networks)

DNN training

Figure source: https://developer.nvidia.com/blog/inference-next-step-gpu-accelerated-deep-learning/ (accessed November, 2021)
Terminology: efficiencies

Statistical efficiency
How well the model learns (convergence)

Hardware efficiency
How long it takes to train 1 epoch
The problem: DL model sizes

DL model sizes keep increasing

- Training times increase
- Models no longer fit in device (GPU) memory

Figure source: S. Bianco et al., "Benchmark analysis of representative deep neural network architectures," IEEE Access, 2018, 6: 64270-64277.
Solution: multi-GPU and distributed training

- Which GPU/node trains what?
- Which paradigm?

Single node, single GPU

Single node, multi-GPU

Multi-node, multi-GPU (distributed)
Data parallelism
Data parallelism (DP)

Central parameter server

Decentralized (all-to-all)

Decentralized (ring, Horovod)

Data parallelism: BSP vs ASP

**Bulk synchronous parallel**

- More communication overhead
- No staleness

**Asynchronous parallel**

- Less communication overhead
- Staleness

Figure source: PipeDream, arXiv preprint, 2018 [3]
Data parallelism: usefulness and limitations

Data parallelism:

- Suffers from communication overhead (grows with model size)
- Cannot handle models larger than 1 worker's memory capacity
- Is useful to speed up training with large amounts of data
Model parallelism
Model parallelism (MP)

Originally for models that don't fit in memory

Figure source: PipeDream, arXiv preprint, 2018 [3]
Model parallelism: (dis)advantages

Advantages:
- Increased total memory capacity -> model size
- PtP communication -> less expensive
- No large global minibatch size: better statistical efficiency

Disadvantages:
- Underutilization
- Cannot hide communication

How to partition the model over the workers?

Figure source: PipeDream, arXiv preprint, 2018 [3]
Hybrid-and pipeline-parallelism
Hybrid parallelism

Mesh-TensorFlow / FlexFlow:
• Split iterations along multiple dimensions
  • Input samples, operators, attributes, parameters

Issues:
• No pipelining: up to 90% performance missed
• Implementations lacking...
Pipeline parallelism (PP)

Like MP, but multiple batches processed in a pipelined fashion
Pipeline parallelism (PP)

Advantages:
- Better hardware utilization than MP
- Less communication than DP
- Can hide communication

**Intra-batch pipeline parallelism**

Pipeline microbatches inside a minibatch

<table>
<thead>
<tr>
<th>GPU 0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU 1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>U</td>
</tr>
<tr>
<td>GPU 2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>U</td>
</tr>
<tr>
<td>GPU 3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>U</td>
</tr>
</tbody>
</table>

Forward pass: 3
Backward pass: 3
Weight update: U

**No staleness (close to Batch SGD) but pipeline bubble**
Inter-batch pipeline parallelism

Pipeline minibatches

<table>
<thead>
<tr>
<th>GPU 0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>U</th>
<th>4</th>
<th>1</th>
<th>U</th>
<th>5</th>
<th>2</th>
<th>U</th>
<th>6</th>
<th>3</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU 1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>U</td>
<td>1</td>
<td>U</td>
<td>4</td>
<td>2</td>
<td>U</td>
<td>5</td>
<td>3</td>
<td>U</td>
<td>6</td>
</tr>
<tr>
<td>GPU 2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>U</td>
<td>3</td>
<td>1</td>
<td>U</td>
<td>2</td>
<td>U</td>
<td>4</td>
<td>3</td>
<td>U</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>GPU 3</td>
<td>0</td>
<td>0</td>
<td>U</td>
<td>1</td>
<td>1</td>
<td>U</td>
<td>2</td>
<td>2</td>
<td>U</td>
<td>3</td>
<td>3</td>
<td>U</td>
<td>4</td>
<td>4</td>
<td>U</td>
</tr>
</tbody>
</table>

No pipeline bubbles but staleness

- 3 Forward pass
- 3 Backward pass
- U Weight update
Pipeline-parallelism implementations
Intra-batch pipelining:
  - GPipe [1] (and TorchGPipe [2]) (single-node)
  - DAPPLE [6] (multi-node with replication)

Inter-batch pipelining:
  - PipeDream [4](-2BW [5]) (multi-node with replication)
Pipeline-parallelism: challenges

Work partitioning
  • Minimize load imbalance
  • Minimize communication across pipeline stages

Work (FW / BW) scheduling
  • Handle FW for new (micro)batch or BW for in-flight (micro)batch?

Ensuring effective learning
PipeDream: automated partitioning

Automated partitioning based on per-stage latency:

• To minimize overall training time: minimize time taken by slowest stage

Profiler:

• Obtains per-layer compute times and communication sizes

Optimizer:

• Dynamic programming approach to find partitioning that minimizes overall training time
• Is topology-aware

Simple static 1F1B work scheduling:

Alternate FW and BW passes

Keeps pipeline full after warmup

Static -> cheap

Figure source: PipeDream, arXiv preprint, 2018 [3]
PipeDream: effective learning

Problem:
- FW and BW pass for 1 minibatch performed with different weight versions
  - Hinders model convergence

Solution:
- Weight stashing
  - Keep multiple copies of weights

PipeDream-2BW limits number of weight copies to 2 at expense of staleness

Figure source: PipeDream, arXiv preprint, 2018 [3]
GPipe: partitioning and work scheduling

Partitioning:

- **GPipe**: manual
- **TorchGPipe**:
  - Per-layer FW+BW pass profiling
  - Searches partitioning with balanced compute time
  - (Remember: single node and no replication)

- Work scheduling (both):
  - FW for all microbatches
  - BW for all microbatches

Figure source: DAPPLE, PPoPP 2021 [6]
GPipe: effective learning

No staleness

• Semantically similar to DP with BSP
• No convergence issues / no measures needed
Per-stage latency approach (PipeDream) not suitable for intra-batch pipelining

Model expected (full) pipeline latency $T_{PL}$:

$$T_{PL}(N, G, \mathcal{G}) = \min_{1 \leq j < N} \min_{1 \leq m < G} \min_{g \in D(g, m)} T_{PL}(j, m, g)$$

- For all possible partitionings
- Choose partitioning with lowest $T_{PL}$
- Apply 3 device placement policies

Figure source: DAPPLE, PPoPP 2021 [6]
DAPPLE: work scheduling

Early backward scheduling:
• Bring BW passes forward
• Release FW activation memory earlier

Lower peak memory consumption

Figure 3. The different scheduling between GPipe(a) and DAPPLE(b) and their memory consumptions.
DAPPLE: effective learning

No staleness
• Semantically similar to DP with BSP
• No convergence issues / no measures needed
Conclusion

DL models keep increasing in size

Data parallelism reduces training times

Model parallelism can train larger models than a single worker

Pipeline parallelism combines best of both

• But still suffers from bubbles or weight copies
Referenced papers


