Hippie: A Data-Paralleled Pipeline Approach to Improve Memory-Efficiency and Scalability for Large DNN Training

Xiangyu Ye, Zhiquan Lai, Shengwei Li, Lei Cai, Ding Sun, Linbo Qiao, Dongsheng Li

National Key Laboratory of Parallel and Distributed Processing
National University of Defence Technology
Contents

• Background & Motivation

• The Hippie approach

• Evaluation

• Conclusion
DNN (Deep Neural Network) models continue to grow

• Challenges of large DNN training
  • Memory limitation
    • Increase of model parameters
    • Increase of training data
  • Low scalability

https://openai.com/blog/ai-and-compute
Motivation

- **Two challenges of parallelizing DNN Training:**
  - High scalability
  - Low memory overhead

- **Define a new index to measure the efficiency:**
  - Memory Efficiency (ME) = \( \frac{\text{Scalability} \times \text{Throughput}}{\text{Memory}} \)
Contents

• Motivation & Background

• The Hippie approach
  • Overview
  • Communication Schedule
  • Last-stage Schedule
  • Pipeline Plan

• Evaluation

• Conclusion
Overview

- Improve scalability
  - Communication Schedule
- Reduce memory overhead
  - Last-stage Schedule
  - Pipeline Planner
Contents

• Motivation & Background

• The Hippie approach
  • Overview
  • Communication Schedule
  • Last-stage Schedule
  • Pipeline Plan

• Evaluation

• Conclusion
• Stage0 starts to perform AllReduce:

Get gradients

Gradient Buffer
---

**Communication Schedule**

- **Stage 1** starts to perform **AllReduce**:

  - **Stage 3**:
    - Forward
    - Backward
    - AllReduce and Update
    - Idle
    - Communication

  - **Stage 2**:
    - Forward
    - Backward
    - AllReduce and Update
    - Idle
    - Communication

  - **Stage 1**:
    - Forward
    - Backward
    - AllReduce and Update
    - Update
    - Communication

  - **Stage 0**:
    - Forward
    - Backward
    - AllReduce
    - Update
    - Communication

---

**Get gradients**
Pipeline starts to perform backward:

- Stage 0: AllReduce
- Stage 1: Update
- Stage 2: AllReduce and Update
- Stage 3: AllReduce and Update

Gradient Buffer
Communication Schedule

- **Stage1 ends the AllReduce:**

  - **Forward**
  - **Backward**
  - **AllReduce gradient or update weight**
  - **Idle**
  - **communication**

  ![Diagram](communication_schedule_diagram.png)

  - Stage3: F0, F1, F2, F3, B0, B1, B2, B3
  - Stage2: F0, F1, F2, F3, Idle, B0, B1, B2, B3
  - Stage1: F0, F1, F2, F3, AllReduce, B0, B1, B2, B3
  - Stage0: F0, F1, F2, F3, AllReduce, B0, B1, B2, B3

  **Store gradients**

  **Gradient Buffer**

  - AllReduce and Update
  - Forward or Backward
  - Communication

  - Stage3 ends with AllReduce and Update
  - Stage1 ends with AllReduce
  - Stage0 ends with AllReduce

  - Stage1 ends the AllReduce:
    - Store gradients
Communication Schedule

- **Stage0 ends the AllReduce:**

  ![Diagram showing communication schedule with stages and AllReduce events]

  **Stage3**
  - Forward: F0, F1, F2, F3
  - Backward: B0, B1, B2
  - AllReduce: B3
  - AllReduce and Update: F0, F1, F2, F3

  **Stage2**
  - Idle: B0
  - B3
  - AllReduce and Update: F0, F1, F2, F3

  **Stage1**
  - AllReduce: B0
  - B1, B2, B3
  - Update: F0, F1, F2, F3
  - AllReduce: B0, B1, B2, B3

  **Stage0**
  - AllReduce
  - B0, B1, B2, B3
  - Update: F0, F1, F2, F3
  - AllReduce: B0, B1, B2, B3

  **Store gradients**

  **Gradient Buffer**
Communication Schedule

- Stage 3 starts to perform AllReduce and update:

Stage3: F0, F1, F2, F3, B0, B1, B2, B3
Stage2: F0, F1, F2, F3, B0, B1, B2, B3
Stage1: F0, F1, F2, F3, AllReduce, B0, B1
Stage0: F0, F1, F2, F3, AllReduce, B0

Gradient Buffer
• Stage2 starts to perform AllReduce and update:
• Stage 1 starts to perform update:

- Store gradients
- Get gradients

Gradient Buffer
Communication Schedule

- **Stage1 starts to perform update:**

  - **Stage3**
    - Forward
    - Backward
    - AllReduce and Update
  - **Stage2**
    - Forward
    - Backward
    - AllReduce and Update
  - **Stage1**
    - Forward
    - Backward
    - AllReduce
    - Update
  - **Stage0**
    - Forward
    - Backward
    - AllReduce

**Gradient Buffer**

- Store gradients
- Get gradients

• Forward
• Backward
• AllReduce and Update
• Idle
• Communication
- Pipeline starts to perform next step:
Contents

• Motivation & Background

• The Hippie approach
  • Overview
  • Communication Schedule
  • Last-stage Schedule
  • Pipeline Plan

• Evaluation

• Conclusion
Last-stage Schedule

1. Store the data to pass
2. Release the graph
3. Release intermediate results
Pipeline Planner

- **Aim:**
  - Generate an efficient pipeline

- **Optimization goal:**
  - Memory efficiency (ME)

- **Process:**
  - Partition the model
  - Select specific layers to apply re-computation
Contents

• Background & Motivation
• The Hippie approach

• Evaluation
  • Experimental setup
  • Memory efficiency
  • Scalability
  • Convergence

• Conclusion
**Experimental setup**

- **Models and datasets**

<table>
<thead>
<tr>
<th>Model</th>
<th># of Params</th>
<th>Dataset</th>
<th>Target Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNMT-8</td>
<td>191M</td>
<td>WMT16 EN-De</td>
<td>24 BLEU</td>
</tr>
<tr>
<td>GNMT-16</td>
<td>290M</td>
<td>WMT16 EN-De</td>
<td>24 BLEU</td>
</tr>
<tr>
<td>VGG-16</td>
<td>138M</td>
<td>ImageNet</td>
<td>70% top-1</td>
</tr>
<tr>
<td>AmoebaNet-18</td>
<td>318M</td>
<td>ImageNet</td>
<td>70% top-1</td>
</tr>
</tbody>
</table>
## Experimental setup

### Training approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hippie</td>
<td>The Hippie with four stages and with two stages within single nodes performs better as efficiency will be reduced greatly by the cross-node pipeline.</td>
</tr>
<tr>
<td>Gpipe+DP</td>
<td>We implement a training process that integrates Gpipe and DP, without applying recomputation and hiding communication.</td>
</tr>
<tr>
<td>DP</td>
<td>Data parallelism with intra-iteration computation communication overlap, which is one of the most efficient distributed training approaches under the PyTorch framework.</td>
</tr>
<tr>
<td>MP+DP</td>
<td>The method implemented to integrate MP and DP for training larger models.</td>
</tr>
</tbody>
</table>
Contents

• Background & Motivation
• The Hippie approach

• Evaluation
  • Experimental setup
  • Memory efficiency
  • Scalability
  • Convergence

• Conclusion
Memory efficiency

- Performance comparison using 16 GPUs
Memory efficiency

- Performance comparison using 16 GPUs
Contents

• Background & Motivation
• The Hippie approach

• Evaluation
  • Experimental setup
  • Memory efficiency
  • Scalability
• Convergence
• Conclusion
Scalability

- Multi-GPU scaling performance for GNMT-8
- Multi-GPU scaling performance for GNMT-16
Convergence

- Accuracy vs. epoch using 16 GPUs

(a) GNMT-16

(b) VGG-16
Contents

• Background & Motivation
• The Hippie approach
• Evaluation
  • Experimental setup
  • Memory efficiency
  • Scalability
  • Convergence
• Conclusion
Conclusion

• We present a distributed training framework which integrates pipelined model parallelism with data parallelism

• We introduce the *Communication Schedule*, enabling Hippie to maintain 90% scaling efficiency on a 16-GPU platform

• We introduce the *Last-stage Schedule* and *Pipeline Planner* to save 30%-60% memory consumption

• Hippie outperforms DP by up to $4.18 \times$ memory efficiency
Hippie: A Data-Paralleled Pipeline Approach to Improve Memory-Efficiency and Scalability for Large DNN Training

Xiangyu Ye, Zhiquan Lai, Shengwei Li, Lei Cai, Ding Sun, Linbo Qiao, Dongsheng Li

Thank you!