Baechi: Fast Device Placement of Machine Learning Graphs

Beomyeol Jeon†, Linda Cai*, Pallavi Srivastava⋄, Jintao Jiang‡, Xiaolan Ke†,
Yitao Meng†, Cong Xie†, Indranil Gupta†

SoCC ’20

†University of Illinois at Urbana-Champaign, *Princeton University, ⋄Microsoft, ‡UCLA
Increasing Machine Learning (ML) Model Size
Not Enough Memory

- GPUs used in AWS, Google Cloud, and Azure

<table>
<thead>
<tr>
<th>GPU</th>
<th>P4</th>
<th>M60</th>
<th>K80</th>
<th>P100</th>
<th>T4</th>
<th>V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>8 GB</td>
<td>12 GB</td>
<td>12/16 GB</td>
<td>16 GB</td>
<td>16/32 GB</td>
</tr>
</tbody>
</table>

- Even 32GB GPU insufficient for > 1.3 B parameters [1]

- ML training on memory-constrained devices
  - Smartphones, UAVs, drones, etc.

Multi-GPU Training: Model Parallelism

How to place ML operators on devices?
Why Does Device Placement Matter?

• ML Training repeats training steps of updating parameters

• *Step time*: Elapsed time for a single training step of the placed ML model

• *Bad* placement $\Rightarrow$ Step time $\uparrow$ (communication overhead $\uparrow$, no parallelism)

• *Slow* placement time $\Rightarrow$ Entire training time $\uparrow$ (placement + training)

• **Goal**: Place a ML model *fast* (low placement time) and *well* (low step time)
Prior Work

• **Expert-designed** Approach
  • E.g. Google Neural Machine Translation (GNMT) [2]
  • Require **domain knowledge** and **significant manual efforts**

• **Learning-based** Approaches
  • Reinforcement learning (RL)
  • E.g., ColocRL [3], HierarchicalRL [4], Placeto [5]
  • Require **very long** time to place ML models (2 hours ~ 3 days)
  • Require **re-training** on different ML models and varying environment

Baechi

• ML placement system that incorporates *algorithmic* approaches into TensorFlow

• Our contributions
  • Placement algorithms for *memory-constrained* environments
    • Memory-constrained Earliest Task First (*m-ETF*)
    • Memory-constrained Small Communication Time (*m-SCT*)
      • Provably *within a constant factor* of the optimal execution time*
    • Memory-constrained Topological Sort (*m-TOPO*) [strawman]
  • Optimizations
    • Co-adjust Placement, Co-placement, Operator Fusion, Sequential Communication Support

• **Place quickly:** 654–206K× faster placement time than learning-based approaches
  • Place ML models on 4 GPUs within *only 1.2* seconds

• **Place well:** *only* up to 6.2% higher step time than expert’s placements

* Conditions apply
Algorithm 1: m-ETF

• Earliest Task First (ETF) [6]
  • Schedule an operator with *earliest schedulable time* on its corresponding device *first*
  • *Infinite* memory assumed

Algorithm 1: m-ETF

- Earliest Task First (ETF) [6]
  - Schedule an operator with *earliest schedulable time* on its corresponding device *first*
  - *Infinite* memory assumed

- **Our modified version: m-ETF**
  - What if device memory limit is 5?
  - *Exclude* devices with *insufficient* memory from placement

Algorithm 2: m-SCT

- Small Communication Time (SCT) [7]
  - Find operator’s favorite child that is scheduled on the same device via ILP

Device 0

Device 1

Device 2

Memory: 6

Memory: 4

Execution time: 11

[7] Hanen and Munier. An Approximation Algorithm for Scheduling Dependent Tasks on m Processors with Small Communication Delays. ETFA ’95

* Conditions apply
Algorithm 2: m-SCT

- Small Communication Time (SCT) [7]
  - Find operator’s favorite child that is scheduled on the same device via ILP

- **Our modified version: m-SCT**
  - Determine favorite child via relaxed ILP
  - Each device memory limit is 5

Theorem 1. *m-SCT’s execution time has a constant approximation ratio with respect to the optimal execution time*. [7] Hanen and Munier. An Approximation Algorithm for Scheduling Dependent Tasks on m Processors with Small Communication Delays. ETFA ‘95

*Conditions apply*
Do the Algorithms Work for TensorFlow?

• Generated placement results were **infeasible**
• Performance was **awful**

• Challenges
  1) TensorFlow colocation constraints
  2) Excessive communication overheads
  3) Massive number of operators
  4) Different network architectures: parallel vs. sequential
Challenges #1: TensorFlow Colocation Constraints

- TensorFlow requires some operators to be *colocated*
Challenges #1: TensorFlow Colocation Constraints

• TensorFlow requires some operators to be collocated

⇒ Tried post-adjust placement
  • Fix colocation-unaware placement to satisfy the colocation constraints
    • Compute-dominant, memory-dominant, majority
  • Inconsistent performance gain
Challenges #1: TensorFlow Colocation Constraints

- TensorFlow requires some operators to be colocated

⇒ Tried post-adjust placement
  - Fix colocation-unaware placement to satisfy the colocation constraints
    - Compute-dominant, memory-dominant, majority
  - Inconsistent performance gain

⇒ Co-adjust placement
  - Consider colocations while creating schedule
  - 1st operator in a group placed ⇒ other ops in the group placed on the same device
Challenge #2: Communication Blowup

- Splitting an ML model graph
  ⇒ Communication ↑
  ⇒ Step time ↑
Challenge #2: Communication Blowup

• Splitting an ML model graph
  ⇒ Communication ↑
  ⇒ Step time ↑

⇒ Operator Co-placement
  • Operator’s output is only used by its successor
    ⇒ Place them together
  • Place respectively-matched forward and backward operators together
Challenge #3: Massive Number of Operators

• Number of operators $\uparrow \implies$ Placement time $\uparrow$

• E.g., 4-layer GNMT
  • 22,340 operators $\implies$ 7-minute placement time

$\Rightarrow$ Operator Fusion
  • Fuse operators that are *directly connected* and *in the same co-placement group*
Challenge #3: Massive Number of Operators

- Number of operators $\uparrow \implies$ Placement time $\uparrow$

- E.g., 4-layer GNMT
  - 22,340 operators $\implies$ 7-minute placement time

$\Rightarrow$ Operator Fusion
- **Fuse** operators that are *directly connected* and *in the same co-placement group*
- May introduce *cycles*
  - Checking all cycles – Expensive, Not scalable
  - **Conservative** but *scalable* heuristic
- Minimize step time
Challenge #3: Massive Number of Operators

⇒ *Forward-Operator-based Placement*
  - Place ops by *only* considering forward ops
    - Place backward ops as their corresponding forward ops on the same device
  - With sufficient memory*

• 4-layer GNMT
  - # operators: 22,340 ⇒ 706
  - Placement time: 7 minutes ⇒ 1.2 seconds

* each GPU memory ≥ model graph memory requirement
Challenge #4: Different Network Architecture

• m-SCT and m-ETF assume parallel communication

• Environment with a constrained network
  • Only sequential communication is supported
  • E.g., Indirect GPU-to-GPU communication

⇒ Sequential Communication Support
  • Introduce device communication queues
  • Support computation-communication overlap
  • Cache received data to avoid duplicate transfers
Baechi WorkFlow

tf.Graph -> Graph Generator -> Graph Optimizer -> Co-Placement Grouper -> Operator Fuser

TensorFlow Runtime

Placed tf.Graph

Execution Simulator

Global Scheduler (m-SCT/m-ETF/m-TOPO)

Device Device Device Device
How Long Does It Take to Generate Placement?

- 4 NVIDIA RTX 2080 GPUs (8GB) with shared communication
  - No NVLink (Direct GPU-to-GPU communication)

<table>
<thead>
<tr>
<th>Model</th>
<th>HierarchicalRL [34]</th>
<th>Placeto [2]</th>
<th>Baechi (m-SCT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-V3</td>
<td>11 hrs 50 mins</td>
<td>1 hr 49 mins</td>
<td>1-10 seconds</td>
</tr>
<tr>
<td>NMT (GNMT)</td>
<td>1 day 21 hrs 14 mins</td>
<td>2 days 20 hrs 40 mins</td>
<td>1.2-48 seconds</td>
</tr>
</tbody>
</table>

Inception-V3: \(654\times-42.6K\times\) Speedup

GNMT: \(3392\times-206K\times\) Speedup
How Fast Are Placed Models (Step Times)?

- Expert-designed placement
  - Inception V3 [4], GNMT [2]

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Single GPU</th>
<th>Expert</th>
<th>m-TOPO</th>
<th>m-ETF</th>
<th>m-SCT</th>
<th>m-ETF</th>
<th>m-SCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-V3</td>
<td>32</td>
<td>0.269</td>
<td>0.269</td>
<td>0.286</td>
<td>0.269</td>
<td>0.269</td>
<td>0.00% (1 GPU Expert)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.491</td>
<td>0.491</td>
<td>0.521</td>
<td>0.491</td>
<td>0.491</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNMT</td>
<td>128</td>
<td>0.251</td>
<td>0.214</td>
<td>0.265</td>
<td>0.224</td>
<td>0.212</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(length: 40)</td>
<td>256</td>
<td>0.474</td>
<td>0.376</td>
<td>0.481</td>
<td>0.354</td>
<td>0.369</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNMT</td>
<td>128</td>
<td>0.319</td>
<td>0.259</td>
<td>0.348</td>
<td>0.264</td>
<td>0.267</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(length: 50)</td>
<td>256</td>
<td>0.618</td>
<td>0.484</td>
<td>0.609</td>
<td>0.502</td>
<td>0.516</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- m-TOPO: up to 34% higher than expert
- m-ETF: -4.5% to 6.2% speedup
- m-SCT: -6.2% to 1.9% speedup

What If Memory Is Constrained?

- 30% per GPU memory (2.4 GB)

<table>
<thead>
<tr>
<th>Model</th>
<th>Single GPU</th>
<th>Expert</th>
<th>m-TOPO</th>
<th>m-ETF</th>
<th>m-SCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-V3</td>
<td>OOM</td>
<td>OOM</td>
<td>0.690 (58.6%)</td>
<td>0.312 (13.8%)</td>
<td>0.292 (7.9%)</td>
</tr>
<tr>
<td></td>
<td>OOM</td>
<td></td>
<td>0.221 (3.2%)</td>
<td>0.272 (2.6%)</td>
<td>0.230 (2.6%)</td>
</tr>
<tr>
<td>GNMT</td>
<td>OOM</td>
<td>OOM</td>
<td></td>
<td>0.212 (0.0%)</td>
<td></td>
</tr>
</tbody>
</table>

m-SCT: only up to 13.8% slower than sufficient memory
How Much Are Optimization Benefits?

• All optimizations applied (m-SCT)

<table>
<thead>
<tr>
<th>Model</th>
<th>Un-Optimized</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Num. Ops</td>
<td>Placement (seconds)</td>
</tr>
<tr>
<td>Inception-V3</td>
<td>6884</td>
<td>68.0</td>
</tr>
<tr>
<td>GNMT (length: 40)</td>
<td>18050</td>
<td>275.1</td>
</tr>
<tr>
<td>GNMT (length: 50)</td>
<td>22340</td>
<td>406.1</td>
</tr>
</tbody>
</table>

Number of Operators: **96.8%–99.8% Reduction**

Placement times: **75.6×–229.3× Speedup**

Step times: **1.1×–3.0× Speedup**
Takeaways

• Current state-of-the-art learning-based ML placement algorithms
  • Require very long placement time (2 hours ~ 3 days)
  • Require re-training the placement model on ML model and environment changes
• Baechi is a fast placement system by using algorithmic approaches
  • Placement algorithms for memory-constrained environments
    • m-TOPO (Topological Sort), m-ETF (Earlier Task First), m-SCT (Small Communication Time)
  • Optimizations
    • Co-adjust Placement, Co-placement, Operator Fusion, Sequential Communication Support
• Place fast: 654–206K× faster placement time than learning-based approaches
  • Place ML models on 4 GPUs within only 1.2 seconds
• Place well: only up to 6.2% higher step time than expert’s placements