AutoGraph: Optimizing DNN Computation Graph for Parallel GPU Kernel Execution

Yuxuan Zhao, Qi Sun, Zhuolun He, Yang Bai, Bei Yu

The Chinese University of Hong Kong
{yxzhao21,byu}@cse.cuhk.edu.hk

Feb. 07–14, 2023
Introduction
DNN Deployment Stack

DL Frameworks

High-level Computation Graph Optimization

Low-level Tensor Program Optimization

Kernel Dispatching, Kernel Submission

Runtime Optimization
DL Frameworks

High-level Computation Graph Optimization

for j in range(None):
  for k in range(None):
    for l in range(None):
      C = ...

Low-level Tensor Program Optimization

Kernel Dispatching, Kernel Submission

Runtime Optimization

DNN Deployment Stack
Prior Arts

• Equivalent Graph Substitution:
  - TASO\(^1\) takes operator definitions and specifications, then automatically generates and verifies graph substitutions.

• Parallel GPU Kernel Launch:
  - IOS\(^2\) divides the computation into different stages and uses DP to find the optimized launch schedule.
  - Nimble\(^3\) supports parallel kernel launch for the whole model and leverages the AOT scheduler to minimize scheduling overhead.

---

\(^1\)Zhihao Jia et al. (2019). “TASO: optimizing deep learning computation with automatic generation of graph substitutions”. In: *Proc. SOSP*.

\(^2\)Yaoyao Ding et al. (2021). “IOS: Inter-Operator Scheduler for CNN Acceleration”. In: *Proc. MLSys*.

• Equivalent Graph Substitution:
  - TASO\(^1\) takes operator definitions and specifications, then automatically generates and verifies graph substitutions.

• Parallel GPU Kernel Launch:
  - IOS\(^2\) divides the computation into different stages and uses DP to find the optimized launch schedule.
  - Nimble\(^3\) supports parallel kernel launch for the whole model and leverages the AOT scheduler to minimize scheduling overhead.

Can we bridge the gap between them?

---

\(^1\) Zhihao Jia et al. (2019). “TASO: optimizing deep learning computation with automatic generation of graph substitutions”. In: Proc. SOSP.


Challenges

Huge graph optimization search space

- Modern DNN models can be complex and large.
- The number of available graph substitutions are huge.
Huge graph optimization search space

- Modern DNN models can be complex and large.
- The number of available graph substitutions are huge.

Inter-operator parallelism is ignored

- Previous graph optimization methods focus on sequential kernel launch.
- Lack runtime information.

(a) Sequential kernel launch

(b) Parallel kernel launch
Details of AutoGraph
Overview of AutoGraph

- Tackle huge search space:
  - Flow-based graph partition method.
  - Dynamic programming + backtracking search.

- Tackle inter-operator parallelism:
  - Customized cost function.
  - Runtime information based on GPU Multi-Stream.
Overview of AutoGraph

- **Tackle huge search space:**
  - Flow-based graph partition method.
  - Dynamic programming + backtracking search.

- **Tackle inter-operator parallelism:**
  - Customized cost function.
  - Runtime information based on GPU Multi-Stream.
Flow-based Graph Partition

- The node capacity is defined as the number of available graph substitutions.
- The entire computation graph is recursively split into independent subgraphs by its minimum cut.
- Adjacent subgraphs are merged to form new subgraphs.
Backtracking Search via Mixed Critical Path Cost

- Backtracking search is used to optimize each subgraph.
- We use the mixed critical path cost in Equation 1 as the selection criteria.

\[
C_E = \alpha \sum_{v \in V_c} \text{cost}(v) + \sum_{v \in V} \text{cost}(v)
\]

\[
= (1 + \alpha) \sum_{v \in V_c} \text{cost}(v) + \sum_{v \in V-V_c} \text{cost}(v).
\]
A transition state in our dynamic programming + backtracking search.

- We observe that different graph partitioning sequences share the same sub-sequence.
- The problem can be solved by Equation \( 2 \).

\[
MCP[G] = \min_{p} (MCP[G - p] + BSMCP[p]).
\]
The operator nodes on different branches are assigned to different streams with proper synchronization events inserted.

CUDA Graph is used to launch the computation graph.

We sample the top-$k$ candidates for on-board verification each time.
Experimental Results
Experimental Settings

• Platform:
  - NVIDIA GeForce RTX 2080Ti GPU.
  - CUDA 11.0, cuDNN 8.0.5, and PyTorch 1.7.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name block#</th>
<th>input shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Inception-v3</td>
<td>11 [1, 3, 299, 299]</td>
</tr>
<tr>
<td></td>
<td>ResNet-50</td>
<td>16 [1, 3, 224, 224]</td>
</tr>
<tr>
<td></td>
<td>ResNeXt-50</td>
<td>16 [1, 3, 224, 224]</td>
</tr>
<tr>
<td></td>
<td>NasNet-A</td>
<td>18 [1, 3, 224, 224]</td>
</tr>
<tr>
<td></td>
<td>NasNet-Mobile</td>
<td>12 [1, 3, 224, 224]</td>
</tr>
<tr>
<td>RNN</td>
<td>RNNTC-SRU</td>
<td>10 [1 \times 10, 1024]</td>
</tr>
<tr>
<td>Transformer</td>
<td>BERT</td>
<td>8 [16 \times 64, 1024]</td>
</tr>
</tbody>
</table>
Experimental Settings

- Platform:
  - NVIDIA GeForce RTX 2080Ti GPU.
  - CUDA 11.0, cuDNN 8.0.5, and PyTorch 1.7.

- Seven modern DNNs are benchmarked:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>block#</th>
<th>input shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Inception-v3</td>
<td>11</td>
<td>[1, 3, 299, 299]</td>
</tr>
<tr>
<td></td>
<td>ResNet-50</td>
<td>16</td>
<td>[1, 3, 224, 224]</td>
</tr>
<tr>
<td></td>
<td>ResNeXt-50</td>
<td>16</td>
<td>[1, 3, 224, 224]</td>
</tr>
<tr>
<td></td>
<td>NasNet-A</td>
<td>18</td>
<td>[1, 3, 224, 224]</td>
</tr>
<tr>
<td></td>
<td>NasNet-Mobile</td>
<td>12</td>
<td>[1, 3, 224, 224]</td>
</tr>
<tr>
<td>RNN</td>
<td>RNNTC-SRU</td>
<td>10</td>
<td>[1 × 10, 1024]</td>
</tr>
<tr>
<td>Transformer</td>
<td>BERT</td>
<td>8</td>
<td>[16 × 64, 1024]</td>
</tr>
</tbody>
</table>
End-to-end Model Inference Latency

Table: Model inference latency results (ms).

<table>
<thead>
<tr>
<th>Model</th>
<th>JIT</th>
<th>TASO+JIT</th>
<th>IOS</th>
<th>Nimble</th>
<th>TASO+Nimble</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-v3</td>
<td>8.839</td>
<td>7.819</td>
<td>3.788</td>
<td>3.174</td>
<td>2.928</td>
<td>2.799</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>4.566</td>
<td>4.554</td>
<td>3.284</td>
<td>2.144</td>
<td>1.988</td>
<td>1.905</td>
</tr>
<tr>
<td>ResNeXt-50</td>
<td>7.540</td>
<td>7.369</td>
<td>3.056</td>
<td>7.708</td>
<td>5.933</td>
<td>2.892</td>
</tr>
<tr>
<td>RNNTC-SRU</td>
<td>1.496</td>
<td>1.307</td>
<td>-</td>
<td>0.486</td>
<td>0.387</td>
<td>0.387</td>
</tr>
</tbody>
</table>

- Compare with TASO, our method achieves speedup ranging from $1.04 \times$ to $3.47 \times$ on parallel kernel launch framework.
- Compare with IOS and Nimble, our method achieves speedup ranging from $1.06 \times$ to $1.26 \times$ on the benchmark models.
• “w/o. Opt.” means directly measuring the initial computation graph.
• “w/o. DP” means directly using the minimum partitioning set without our DP-based method.
The normalized throughput comparisons of different frameworks on various batch sizes for NasNet-Mobile.

- A larger batch size provides more intra-operator parallelism.
- We can still exploit inter-operator parallelism and graph optimization to further improve the inference performance.
Conclusion
• Existing graph optimization methods fail to utilize inter-operator parallelism and thus impair system capability within a parallel kernel launch framework.

• We propose AutoGraph to bridge the gap. Experimental results demonstrate that our method achieves up to $3.47 \times$ speedup over previous arts.

• Moreover, AutoGraph outperforms state-of-the-art parallel kernel launch frameworks by up to $1.26 \times$. 
THANK YOU!