Collage: Automated Integration of Deep Learning Backends

Byungsoo Jeon*1, Sunghyun Park*2, Peiyuan Liao1,4, Sheng Xu3, Tianqi Chen1,2, Zhihao Jia1

1Carnegie Mellon University, 2OctoML, 3Amazon Web Services, 4Praxis Pioneering
Deep Learning (DL) Backend

Backend
- a software library or a runtime framework that takes DL workloads as inputs and generates an optimized low-level target code

<table>
<thead>
<tr>
<th>Backend</th>
<th>XLA</th>
<th>GLOW</th>
<th>TVM</th>
<th>TC</th>
<th>nGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MKL</td>
<td>DNNL</td>
<td>OpenVINO</td>
<td>cuBLAS</td>
<td>cuDNN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>intel</td>
</tr>
</tbody>
</table>

TVM Conf 2021
Deep Learning (DL) Backend

**Backend**
- A software library or a runtime framework that takes DL workloads as inputs and generates an optimized low-level target code

<table>
<thead>
<tr>
<th>Frontend</th>
<th>XLA</th>
<th>GLOW</th>
<th>TVM</th>
<th>TC</th>
<th>nGraph</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MKL</td>
<td>DNNL</td>
<td>OpenVINO</td>
<td>cuBLAS</td>
<td>cuDNN</td>
</tr>
<tr>
<td>Hardware</td>
<td>intel</td>
<td>AMD</td>
<td>Apple</td>
<td>arm</td>
<td>XILINX</td>
</tr>
</tbody>
</table>

TVM Conf 2021
Deep Learning (DL) Backend

Backend
- a software library or a runtime framework that takes DL workloads as inputs and generates an optimized low-level target code
Observation: Diversified DL Backends

DL backends are highly diversified and evolving fast
- Each backend has its own coverage (e.g., HW, DL operators) and strength
Problem: Backend Integration

Backend Integration = Backend Register + Backend Placement

Computation Graph

Diverse Backends
- cuBLAS
- cuDNN
- TensorRT
- TVM

Optimized Backend Placement

DL System
Problem: Backend Integration

Backend Integration = **Backend Register** + Backend Placement

![Diagram]

- **Computation Graph**
- **Diverse Backends**
  - cuBLAS
  - cuDNN
  - TensorRT
  - TVM
- **Backend Register**
- **Optimized Backend Placement**
- **DL System**

TVM Conf 2021
Problem: Backend Integration

Backend Integration = Backend Register + **Backend Placement**
Existing Approach: Manual Backend Integration

- Heuristics are often sub-optimal and susceptible to be outdated
- Direct code modification to the DL framework is required

Rule-based Heuristics

If \( op \) == "conv2d"
  If \( \text{cudnn\_enabled} \)
    \( \text{lower\_to\_cudnn\_kernel} \)
  Else if ...
  Else if \( op \) == "batch\_matmul"
    If \( \text{cublas\_enabled} \)
      \( \text{lower\_to\_cublas\_kernel} \)
    ....
Our Approach: Automated Backend Integration

- It eliminates manual efforts to design heuristics and change codes
- It provides fast and stable performance across different models and hardwares

Collage

DL Workloads → Backend Specification

- cuDNN
- cuBLAS
- TensorRT

Automatic Backend Placement

Enables seamless integration

Determined → Examine

Choose the best backend

TVM Conf 2021
System Overview

Computation Graph (G)

Backend Pattern Abstraction (Sec 3)
1) Op pattern
   conv = is_op("nn.conv2d")(*, *)
   # conv + element-wise operator (e.g., ReLU)
   fused = conv.has_attr("OpPattern": K_ELEM))
   add_pattern(backend= 'cudnn', pattern= fused)

2) Op pattern rule
   # tvm_pattern_rule is a func that checks if the
   # pattern rule can be applied on the input IR
   add_pattern_rule(backend= 'tvm',
   rule= tvm_pattern_rule)

Backend Pattern Registry (Sec 3)

Backend Pattern Generators (Sec 3)

Built-in Pattern Rules

User-defined Pattern Rules

Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP

\[ C_{\text{OPT}}(G) = \min(\mathcal{M}((\text{conv1}, \text{cud})) + \mathcal{M}((\text{conv2.relu}, \text{TRT}) + \mathcal{M}((\text{conv3.add}, \text{TVM}) + \mathcal{M}((\text{dense}, \text{TVM}), \mathcal{M}((\text{conv1}, \text{TRT}) + \mathcal{M}((\text{conv3.add}, \text{TVM}) + \mathcal{M}((\text{conv2.relu}, \text{cud}) + \mathcal{M}((\text{dense}, \text{TRT}), ... \]

Op-level optimized placement

Graph Cost

Measurer (\mathcal{M})

Single Op

Op Cost

Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search

Seed

Pick best

Optimized Backend Placement

TVM Conf 2021
~ 70 LoC to integrate one backend
Overview

**Collage**

**Backend Pattern Abstraction (Sec 3)**
1. **Op pattern**
   - `conv = is_op("nn.conv2d")(* , *)`
   - `# conv + element-wise operator (e.g., ReLU)`
   - `fused = conv.has_attr("OpePattern": K_ELEM))`
   - `add_pattern(pattern= "cudnn", pattern= fused)`

2. **Op pattern rule**
   - `# tvm_pattern_rule is a func that checks if the`
   - `# pattern rule can be applied on the input IR`
   - `add_pattern_rule(backend= 'tvm', rule= tvm_pattern_rule)`

**Backend Pattern Registry (Sec 3)**
- **cuDNN**
- **Conv + ReLU**
- **TVM**
- **TensorRT**

**Op-level Placement Optimizer (Sec 4.2)** – Optimize backend placements with DP

- `$C_{opt}(G) = \min(M((conv1, cuD)) + M((conv2, relu), TRT) + M((conv3 + add), TVM) + M((dense), TVM), M((conv1), TRT) + M((conv3 + add), TVM) + M((conv2 + relu), cuD)) + M((dense), TRT), ...)$

**Graph-level Placement Optimizer (Sec 4.3)** – Fine-tune placements with evolutionary search

**Graph Cost**

**Measurer (M)**

**Single Op**

**Op Cost**

**Optimized Backend Placement**
Overview

Built-in patterns support most of popular backends (e.g., cuDNN, cuBLAS, TensorRT, TVM, MKL, etc.)
Overview

Automated Two-level Optimizer

Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP

\[ C_{OPT}(G) = \min \left( \sum \left( \mathcal{M}(\text{conv1}, \text{cuDNN}) + \mathcal{M}(\text{conv2}, \text{relu}, \text{TRT}) + \mathcal{M}(\text{conv3} + \text{add}, \text{TVM}) + \mathcal{M}(\text{dense}, \text{TRT}), \mathcal{M}(\text{conv1}, \text{TRT}) + \mathcal{M}(\text{conv3} + \text{add}, \text{TVM}) + \mathcal{M}(\text{conv2 + relu}, \text{cuDNN}) + \mathcal{M}(\text{dense}, \text{TRT}) \right) \right) \]

Op-level optimized placement

Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search

Seed

Graph

Measurer (M)

Single Op

Op Cost

Optimized Backend Placement
Overview

Collage

**Backend Pattern Abstraction (Sec 3)**

1. Op pattern
   
   ```
   conv = is_op(nn.conv2d) (*, *)
   # conv + element-wise operator (e.g., ReLU)
   fused = conv.has_attr("OpPattern": K_ELEM)
   add_pattern(backend='cudnn', pattern=fused)
   ```

2. Op pattern rule
   
   ```
   # tvm_pattern_rule is a func that checks if the
   # pattern rule can be applied on the input IR
   add_pattern_rule(backend='tvm',
   rule=tvm_pattern_rule)
   ```

**Backend Pattern Generators (Sec 3)**

- Built-in Pattern Rules
- User-defined Pattern Rules

**Op-level Placement Optimizer (Sec 4.2)** – Optimize backend placements with DP

\[
C_{OPT}(G) = \min \left( M(\text{conv1}, \text{cud}) + M(\text{conv2, relu}, \text{TRT}) + M(\text{conv3, add}, \text{TVM}) + M(\text{dense}, \text{TRT}), \right.
\]

**Graph-level Placement Optimizer (Sec 4.3)** – Fine-tune placements with evolutionary search

- Seed
- Pick best

Optimized Backend Placement
End-to-end Evaluation: NVIDIA V100, Intel Xeon

- Stable performance across different networks and hardwares
Optimized Backend Placement

- Collage leverages unique strength of each backend
- Collage maps same type of operators to different backends based on the performance landscape
- Collage employs diverse operator fusion patterns
References

Code: https://github.com/cmu-catalyst/collage