TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions


R244 Large-scale data processing and optimisation
Presentation by Martin Graf on 16/11/2022
(a) Existing DNN frameworks.
(a) Existing DNN frameworks.

(b) TASO.
(a) Existing DNN frameworks.

(b) TASO.
Manually Designed Graph Substitutions

Input Comp. Graph

Graph Subst. Generator (§2)

Graph Subst. Verifier (§3)

Verified Graph Subst.

Graph Subst. and Data Layout Joint Optimizer (§5)

Optimized Comp. Graph

(a) Existing DNN frameworks.

(b) TASO.
(b) Fusing two matrix multiplications using concatenation and split.
Enumerate potential graphs
Algorithm 1 Graph substitution generation algorithm.

1: **Input:** A set of operators \( \mathcal{P} \), and a set of input tensors \( \mathcal{I} \).
2: **Output:** Candidate graph substitutions \( \mathcal{S} \).
3: 
4: // Step 1: enumerating potential graphs.
5: \( \mathcal{D} = \{ \} \) // \( \mathcal{D} \) is a graph hash table indexed by their fingerprints.
6: \( \text{BUILD}(1, \emptyset, \mathcal{I}) \)
7: \( \text{function \hspace{0.5em} BUILD}(n, \mathcal{G}, \mathcal{I}) \)
8: \hspace{1em} if \( \mathcal{G} \) contains duplicated computation then \hspace{1em} return \hspace{1em}
9: \hspace{1em} \mathcal{D} = \mathcal{D} + (\text{FINGERPRINT}(\mathcal{G}), \mathcal{G}) \hspace{1em}
10: \hspace{1em} if \( n < \text{threshold} \) then \hspace{1em}
11: \hspace{2em} for \( \text{op} \in \mathcal{P} \) do \hspace{2em}
12: \hspace{3em} for \( i \in \mathcal{I} \) and \( i \) is a valid input to \( \text{op} \) do \hspace{3em}
13: \hspace{4em} Add operator \( \text{op} \) into graph \( \mathcal{G} \).
14: \hspace{4em} Add the output tensors of \( \text{op} \) into \( \mathcal{I} \).
15: \hspace{4em} \text{BUILD}(n + 1, \mathcal{G}, \mathcal{I}) \hspace{4em}
16: \hspace{4em} Remove operator \( \text{op} \) from \( \mathcal{G} \).
17: \hspace{4em} Remove the output tensors of \( \text{op} \) from \( \mathcal{I} \).

19: 
20: // Step 2: testing graphs with identical fingerprint.
21: \( \mathcal{S} = \{ \} \)
22: \( \text{for} \ \mathcal{G}_1, \mathcal{G}_2 \in \mathcal{D} \) with the same FINGERPRINT() do \( \text{if} \ \mathcal{G}_1 \) and \( \mathcal{G}_2 \) are equivalent for all test cases then \( \mathcal{S} = \mathcal{S} + (\mathcal{G}_1, \mathcal{G}_2) \)
24: \( \text{return} \ \mathcal{S} \)
Algorithm 1 Graph substitution generation algorithm.

1: **Input:** A set of operators \( \mathcal{P} \), and a set of input tensors \( \mathcal{I} \).
2: **Output:** Candidate graph substitutions \( \mathcal{S} \).
3: 
4: // Step 1: enumerating potential graphs.
5: \( \mathcal{D} = \{\} \) // \( \mathcal{D} \) is a graph hash table indexed by their fingerprints.
6: **Build** \((1, \emptyset, \mathcal{I})\)
7: function **Build**\((n, \mathcal{G}, \mathcal{I})\)
8: if \( \mathcal{G} \) contains duplicated computation then
9: return
10: \( \mathcal{D} = \mathcal{D} + (\text{Fingerprint}(\mathcal{G}), \mathcal{G}) \)
11: if \( n < \text{threshold} \) then
12: for \( op \in \mathcal{P} \) do
13: for \( i \in \mathcal{I} \) and \( i \) is a valid input to \( op \) do
14: Add operator \( op \) into graph \( \mathcal{G} \).
15: Add the output tensors of \( op \) into \( \mathcal{I} \).
16: **Build**\((n + 1, \mathcal{G}, \mathcal{I})\)
17: Remove operator \( op \) from \( \mathcal{G} \).
18: Remove the output tensors of \( op \) from \( \mathcal{I} \).

19: // Step 2: testing graphs with identical fingerprint.
20: \( \mathcal{S} = \{\} \)
21: for \( \mathcal{G}_1, \mathcal{G}_2 \in \mathcal{D} \) with the same Fingerprint() do
22: if \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \) are equivalent for all test cases then
23: \( \mathcal{S} = \mathcal{S} + (\mathcal{G}_1, \mathcal{G}_2) \)
24: return \( \mathcal{S} \)

---

Enumerate potential graphs

→ Fingerprinting + Hash Collisions

(Bansal, 2006)

Collect and test candidate substitutions
Table 2. Operator properties used for verification. The operators are defined in Table 1, and the properties are grouped by the operators they involve. Logical variables \(x, y, z\) and \(w\) are of type tensor, and variables \(a, c, k, p, s\) are of type parameter. The variable \(a\) is used for the axis of concatenation and split, \(c\) for the activation mode of convolution, \(k\) for the kernel shape of pooling, \(p\) for the padding mode of convolution and pooling, and \(s\) for the strides of convolution and pooling.

<table>
<thead>
<tr>
<th>Operator Property</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(yx.z.e\texttt{add}(x, y, z) = \texttt{add}(\texttt{add}(x, y), z))</td>
<td>\texttt{add} is associative \texttt{add} is commutative</td>
</tr>
<tr>
<td>(yx.z.\texttt{esum}(x, y, z) = \texttt{esum}(\texttt{esum}(x, y), z))</td>
<td>\texttt{esum} is associative</td>
</tr>
<tr>
<td>(yx.\texttt{esum}(x, y) = \texttt{esum}(\texttt{esum}(x, y), z))</td>
<td>\texttt{esum} is commutative \texttt{esum} is associative</td>
</tr>
<tr>
<td>(yx.\texttt{smul}(x, y) = \texttt{smul}(\texttt{smul}(x, y), z))</td>
<td>\texttt{smul} is associative \texttt{smul} is commutative</td>
</tr>
<tr>
<td>(yx.\texttt{smul}(x, y) = \texttt{smul}(\texttt{smul}(x, y), z))</td>
<td>\texttt{smul} is associative \texttt{smul} is commutative</td>
</tr>
<tr>
<td>(yx.\texttt{transpose}(\texttt{transpose}(x)) = x)</td>
<td>\texttt{transpose} is its own inverse</td>
</tr>
<tr>
<td>(yx.\texttt{transpose}\texttt{transpose}(x) = \texttt{transpose}\texttt{transpose}(x))</td>
<td>\texttt{transpose} is its own inverse</td>
</tr>
<tr>
<td>(yx.z.\texttt{matmul}(x, \texttt{matmul}(y, z)) = \texttt{matmul}(\texttt{matmul}(x, y), z))</td>
<td>\texttt{matmul} is associative \texttt{matmul} is linear</td>
</tr>
<tr>
<td>(yx.\texttt{smul}(x, \texttt{smul}(y, z)) = \texttt{smul}(\texttt{smul}(x, y), z))</td>
<td>\texttt{smul} is associative \texttt{smul} is linear</td>
</tr>
<tr>
<td>(yx.\texttt{smul}(x, \texttt{smul}(y, z)) = \texttt{smul}(\texttt{smul}(x, y), z))</td>
<td>\texttt{smul} is associative \texttt{smul} is linear</td>
</tr>
<tr>
<td>(yx.\texttt{conv}(\texttt{conv}(s, \texttt{pool}(x), y), z) = \texttt{conv}(\texttt{conv}(s, \texttt{pool}(x), y), z))</td>
<td>\texttt{conv} is bilinear</td>
</tr>
<tr>
<td>(yx.\texttt{conv}(\texttt{conv}(s, \texttt{pool}(x), y), z) = \texttt{conv}(\texttt{conv}(s, \texttt{pool}(x), y), z))</td>
<td>\texttt{conv} is bilinear</td>
</tr>
<tr>
<td>(yx.\texttt{conv}(\texttt{conv}(s, \texttt{pool}(x), y), z) = \texttt{conv}(\texttt{conv}(s, \texttt{pool}(x), y), z))</td>
<td>\texttt{conv} is bilinear</td>
</tr>
<tr>
<td>(yx.\texttt{relu}(\texttt{relu}(x)) = \texttt{relu}(\texttt{relu}(x)))</td>
<td>\texttt{relu} with \texttt{relu} applies \texttt{relu}</td>
</tr>
<tr>
<td>(yx.\texttt{pool}(\texttt{pool}(k, x, p, s), y) = \texttt{pool}(\texttt{pool}(k, x, p, s), y))</td>
<td>\texttt{pool} with \texttt{pool} applies \texttt{pool}</td>
</tr>
<tr>
<td>(yx.\texttt{concat}(\texttt{concat}(a, x, y), z) = \texttt{concat}(\texttt{concat}(a, x, y), z))</td>
<td>\texttt{concat} with \texttt{concat} applies \texttt{concat}</td>
</tr>
<tr>
<td>(yx.\texttt{split}(\texttt{split}(x, a, y), z) = \texttt{split}(\texttt{split}(x, a, y), z))</td>
<td>\texttt{split} with \texttt{split} applies \texttt{split}</td>
</tr>
<tr>
<td>(yx.\texttt{concat}(\texttt{concat}(x, y), z) = \texttt{concat}(\texttt{concat}(x, y), z))</td>
<td>\texttt{concat} with \texttt{concat} applies \texttt{concat}</td>
</tr>
<tr>
<td>(yx.\texttt{pool}(\texttt{pool}(k, x, p, s), y) = \texttt{pool}(\texttt{pool}(k, x, p, s), y))</td>
<td>\texttt{pool} with \texttt{pool} applies \texttt{pool}</td>
</tr>
<tr>
<td>(yx.\texttt{concat}(\texttt{concat}(x, y), z) = \texttt{concat}(\texttt{concat}(x, y), z))</td>
<td>\texttt{concat} with \texttt{concat} applies \texttt{concat}</td>
</tr>
<tr>
<td>(yx.\texttt{pool}(\texttt{pool}(k, x, p, s), y) = \texttt{pool}(\texttt{pool}(k, x, p, s), y))</td>
<td>\texttt{pool} with \texttt{pool} applies \texttt{pool}</td>
</tr>
<tr>
<td>(yx.\texttt{concat}(\texttt{concat}(x, y), z) = \texttt{concat}(\texttt{concat}(x, y), z))</td>
<td>\texttt{concat} with \texttt{concat} applies \texttt{concat}</td>
</tr>
</tbody>
</table>
Table 3. The number of remaining graph substitutions after applying the pruning techniques in order.

<table>
<thead>
<tr>
<th>Pruning Techniques</th>
<th>Remaining Substitutions</th>
<th>Reduction v.s. Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>28744</td>
<td>1×</td>
</tr>
<tr>
<td>Input tensor renaming</td>
<td>17346</td>
<td>1.7×</td>
</tr>
<tr>
<td>Common subgraph</td>
<td>743</td>
<td>39×</td>
</tr>
</tbody>
</table>
Algorithm 2 Cost-Based Backtracking Search

1: **Input**: an input graph $G_{in}$, verified substitutions $S$, a cost model $Cost(\cdot)$, and a hyper parameter $\alpha$.
2: **Output**: an optimized graph.
3: 
4: $P = \{G_{in}\}$ // $P$ is a priority queue sorted by $Cost$.
5: while $P \neq \{}$ do
6:     $G = P$.dequeue()
7:     for substitution $s \in S$ do
8:         // $LAYOUT(G, s)$ returns possible layouts applying $s$ on $G$.
9:         for layout $l \in LAYOUT(G, s)$ do
10:             // $APPLY(G, s, l)$ applies $s$ on $G$ with layout $l$.
11:             $G' = APPLY(G, s, l)$
12:             if $G'$ is valid then
13:                 if $Cost(G') < Cost(G_{opt})$ then
14:                     $G_{opt} = G'$
15:                 if $Cost(G') < \alpha \times Cost(G_{opt})$ then
16:                     $P$.enqueue($G'$)
17: return $G_{opt}$

---

Cost-based backtracking algorithm from MetaFlow (Jia, 2019) = Priority Queue + Nested for loop over all substitutions/data layouts
Figure 7. End-to-end inference performance comparison among existing DNN frameworks and TASO. The experiments were performed using a single inference sample, and all numbers were measured by averaging 1,000 runs on a NVIDIA V100 GPU. We evaluated the TASO’s performance with both the cuDNN and TVM backends. For each DNN architecture, the numbers above the TASO bars show the speedup over the best existing approach with the same backend.
Figure 10. A heat map of how often the verified substitutions are used to optimize the five DNN architectures. Only substitutions used in at least one DNN are listed. For each architecture, the number indicates how many times a substitution is used by TASO to obtain the optimized graph.
Figure 7. End-to-end inference performance comparison among existing DNN frameworks and TASO. The experiments were performed using a single inference sample, and all numbers were measured by averaging 1,000 runs on a NVIDIA V100 GPU. We evaluated the TASO’s performance with both the cuDNN and TVM backends. For each DNN architecture, the numbers above the TASO bars show the speedup over the best existing approach with the same backend.
**Figure 12.** End-to-end inference performance comparison on BERT using different strategies to optimize graph substitution and data layout.
Evaluation

Con:
• 743 graph substitutions > 43 operator properties
Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.
Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

There is a sequence of applications of user provided properties proving the correctness of the substitution.
Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

There is a sequence of applications of user provided properties proving the correctness of the substitution.

Each user provided property is equivalent to a substitution.
Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

There is a sequence of applications of user provided properties proving the correctness of the substitution.

Each user provided property is equivalent to a substitution.

All substitutions are sequences of substitutions provided by the user.
Evaluation

Con:
• 743 graph substitutions > 43 operator properties
• Why prune all redundant substitutions?
Evaluation

Con:

• 743 graph substitutions > 43 operator properties
• Why prune all redundant substitutions?
• Cost measure of joint optimizer seems oversimplified
Evaluation

Con:
• 743 graph substitutions > 43 operator properties
• Why prune all redundant substitutions?
• Cost measure of joint optimizer seems oversimplified

Pro:
• Lots of real-world considerations
Evaluation

Con:
• 743 graph substitutions > 43 operator properties
• Why prune all redundant substitutions?
• Cost measure of joint optimizer seems oversimplified

Pro:
• Lots of real-world considerations
• Many individual influential contributions
Impact and Follow Up Work

- **PET**: non-fully equivalent graph substitutions
  
  (Wang, 2021)

- **Equality Saturation**: improves joint optimization algorithm
  
  (Yang, 2021)

- **Unity**: jointly optimizes graph substitutions, data layout, and parallelism
  
  (Unger, 2022)
(Unger, 2022)
References


References


References


TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

R244 Large-scale data processing and optimisation
Presentation by Martin Graf on 16/11/2022