Oliver Rausch*, Tal Ben-Nun*, Nikoli Dryden, Andrei Ivanov, Shigang Li, Torsten Hoefler

DaCeML: A Data-Centric Optimization Framework for Machine Learning

This project received funding from the European Research Council under the European Union’s Horizon 2020 programme (Project PSAP, No. 101002047); and receives EuroHPC-JU funding under grants MAELSTROM, No. 955513 and DEEP-SEA, No. 955606, with support from the Horizon 2020 programme.
Large Scale Computation in ML

Contemporary ML Systems are Inflexible

- Researchers demand efficient, large-scale compute more than ever
- Almost all optimizations relate to the data movement
- But current compilers are highly specialized towards
  - Operators
  - Transformations
  - Models
- Tuning, or even just visualizing the output of a compiler is difficult
DaCeML

- Data-centric lowering and multi-level optimization of DNNs

Usability  Generality  Interactivity
from torch import nn
from daceml.pytorch import dace_module

@dace_module
class MyModule(nn.Module):
    def __init__(self, n_in, n_out):
        super().__init__()
        self.linear = nn.Linear(n_in, n_out)
        self.fanout = n_out

    def forward(self, x):
        return self.linear(x) / self.fanout
from torch import nn
from daceml.pytorch import dace_module

def __init__(self, n_in, n_out):
    self.linear = nn.Linear(n_in, n_out)
    self.fanout = n_out

def forward(self, x):
    return self.linear(x) / self.fanout

class MyModule(nn.Module):
    def __init__(self, n_in, n_out):
        super().__init__()
        self.linear = nn.Linear(n_in, n_out)
        self.fanout = n_out

        def forward(self, x):
            return self.linear(x) / self.fanout

Memlets: explicit data movement at all granularities

Tasklets: stateless computations

Access nodes to data containers

Library nodes with domain-specific semantics

Maps: Parametric parallelism scopes
Optimization Pipeline

Other DNN Frameworks

CPU
GPU
FPGA

Multi-Level Optimization via Progressive Lowering

+ Symbolic Shape Inference

ONNX

SDFG
Other DNN Frameworks

+ Symbolic Shape Inference

Coarse-Grained Transformations

Multi-Level Optimization via Progressive Lowering

CPU

GPU

FPGA

Algebraic Fusion
Lowering ONNX nodes

```python
@python_pure_op_implementation
def Softplus(X, Y):
    Y[:] = numpy.log(1 + numpy.exp(X))
```
```python
def forward(self, x):
    y = F.softplus(x)
    y = torch.tanh(y)
    return x * y
```

[Diagram showing the process of converting a PyTorch model to ONNX and further transformations to SDFG for optimization and execution on different devices.]
Other DNN Frameworks

Model Metadata

+ Symbolic Shape Inference

Coarse-Grained Transformations
Symbolic Automatic Differentiation
Local Data Movement Reduction
Recomputation Tuning
Multi-Level Optimization via Progressive Lowering

CPU
GPU
FPGA

Operator-Level AD

DaCeML: data-centric, Symbolic AD

Data-movement information enables automatic backward-kernel synthesis!
No other framework can perform this type of low-level pre-AD optimization.

Intermediate Memory: 0MiB

Intermediate Memory: 134.75MiB
Memlet propagation & codegen is data-layout aware

Prune layouts That don’t permit lowering to BLAS
# Operator implementation in NumPy

def Softmax(input, output):
    max = input.max(axis=axis, keepdims=True)
    exp = np.exp(input - max)
    sum = exp.sum(axis=axis, keepdims=True)
    output[:] = exp / sum
Other DNN Frameworks

Model Metadata

ONNX

Symbolic Automatic Differentiation

Coarse-Grained Transformations

Local Data Movement Reduction

Global Data Layout Optimization

Hardware Specialization

Multi-Level Optimization via Progressive Lowering

User-guided optimization, combinatorial search

CPU

GPU

FPGA

+ Symbolic Shape Inference

PyTorch

TensorFlow

mxnet
Results - Pre-AD Fusion (Mish)

- XLA manages to produce a single fused kernel
- But fails to eliminate global loads/stores introduced to stash intermediate values
Results – Softmax

Transformation recipe matches hand written kernels and generalizes to other operators (e.g. Layer Normalization)
## Results – Automatic Optimization

<table>
<thead>
<tr>
<th>Model</th>
<th>PyTorch</th>
<th>torch.jit</th>
<th>JAX</th>
<th>TF+XLA</th>
<th>DaCeML</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automatic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-50 (��)</td>
<td>14.55</td>
<td>32.04</td>
<td>9.98</td>
<td>31.94</td>
<td>14.17</td>
</tr>
<tr>
<td>Wide ResNet-50-2 (匀)</td>
<td>22.50</td>
<td>70.94</td>
<td>22.45</td>
<td>70.83</td>
<td>40.49</td>
</tr>
<tr>
<td>MobileNet V2 (��)</td>
<td>9.98</td>
<td>18.45</td>
<td>6.22</td>
<td>15.53</td>
<td>—</td>
</tr>
<tr>
<td>EfficientNet (��)</td>
<td>2.05</td>
<td>6.90</td>
<td>2.04</td>
<td>6.94</td>
<td>2.39</td>
</tr>
<tr>
<td>MLP Mixer (��)</td>
<td>1.63</td>
<td>3.65</td>
<td>1.36</td>
<td>3.66</td>
<td>1.77</td>
</tr>
<tr>
<td>FCN8s (��)</td>
<td>46.85</td>
<td>158.42</td>
<td>46.82</td>
<td>158.40</td>
<td>—</td>
</tr>
<tr>
<td>WaveNet (��)</td>
<td>23.21</td>
<td>46.39</td>
<td>18.67</td>
<td>41.49</td>
<td>—</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt; (single) (��)</td>
<td>11.05</td>
<td>31.76</td>
<td>11.05</td>
<td>31.82</td>
<td>10.93</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt; (mixed) (��)</td>
<td>2.94</td>
<td>8.18</td>
<td>2.92</td>
<td>8.20</td>
<td>3.19</td>
</tr>
<tr>
<td>DLRM (��)</td>
<td>118.07</td>
<td>126.55</td>
<td>117.38</td>
<td>126.83</td>
<td>—</td>
</tr>
</tbody>
</table>
Results – Guided Optimization

EfficientNet-B0 (MBConv)

Time (ms)

<table>
<thead>
<tr>
<th>Time</th>
<th>Forward+Backward</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New fusion opportunity

6 clicks
10 clicks
2 clicks
### Results – Guided Optimization

#### BERT-Large (encoder layer, mixed precision)

<table>
<thead>
<tr>
<th></th>
<th>Forward+Backward</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>[28 / 99]</td>
<td>8.18</td>
<td>2.94</td>
</tr>
<tr>
<td>[28 / 99]</td>
<td>8.20</td>
<td>2.92</td>
</tr>
<tr>
<td>[272 / 675]</td>
<td>8.11</td>
<td>3.19</td>
</tr>
<tr>
<td>[47 / 127]</td>
<td>10.76</td>
<td>3.80</td>
</tr>
<tr>
<td>[25 / 102]</td>
<td>7.62</td>
<td>2.74</td>
</tr>
</tbody>
</table>

Batch: 8, seqlen 512

#### EfficientNet-B0 (MBConv)

<table>
<thead>
<tr>
<th></th>
<th>Forward+Backward</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>[18 / 54]</td>
<td>6.90</td>
<td>2.05</td>
</tr>
<tr>
<td>[15 / 51]</td>
<td>6.94</td>
<td>2.04</td>
</tr>
<tr>
<td>[39 / 141]</td>
<td>7.40</td>
<td>2.39</td>
</tr>
<tr>
<td>[26 / 93]</td>
<td>6.37</td>
<td>1.54</td>
</tr>
<tr>
<td>[13 / 56]</td>
<td>5.97</td>
<td>1.40</td>
</tr>
</tbody>
</table>

Batch: 8, input: 3x224x224

EfficientNet-B0 MBConv 1
Summary

github.com/spcl/daceml