Support Convolution of CNN with Compression Sparse Matrix Multiplication Flow in TVM

Hui-Hsin Liao¹, Chao-Lin Lee¹, Jenq-Kuen Lee¹, Wei-Chih Lai², Ming-Yu Hung², Chung-Wen Huang²

Department of Computer Science,
National Tsing Hua University, Hsinchu, Taiwan ¹
MediaTek Inc. ²
Outline

• Introduction and Background
• Im2col and GEMM
• Conversion Flow in TVM
• Experiments
• Conclusions
Introduction

• Deep learning is playing an important role nowadays.
• Lots previous researches tend to reduce convolutions overhead.
  • Direct convolution
  • Im2col and GEMM (General Matrix Multiplication)
  • Winograd
  • FFT (Fast Fourier Transformation)
• Our work try to build a general flow for different backends which target to sparse models.
Background – Sparse Model

• We collect sparse models from two different sources
  • ImageNet based model: SparseZoo
  • CIFAR10 based model: Prune with distiller from Intel AILab

Example of pruning schedule

Reference: distiller, Intel AILab, SparseZoo, Song Han et al., Learning both Weights and Connections for Efficient Neural Networks, NIPS, 2015
Background – TVM

• Deep learning inference framework
• Support models with different formats
• Support different target backends
• Two level optimization
  • Relay IR : Computation graph optimization
  • TIR : Decouple algorithm from schedule
    • Algorithm: What is computed
    • Schedule: Where and When and How it’s computed

Reference: Apache TVM
Overview of Our Flow

Pruning Tools

Dense CNN Model
- prune

Sparse CNN Model
- relay pass
  - convert conv2d op to im2col_transform→dense

TVM
- relay pass
  - transform weight to compress format
  - convert dense op to sparse_dense op
- Start Inference
Im2col and GEMM

Im2col

Direct convolution

Feature map

Kernel map

Feature map

Kernel map

Output
Im2col and GEMM

Direct convolution

- **Im2col**
- **GEMM** (General Matrix Multiplication)
Im2col and GEMM – Im2col Operator

• The new shape could be inferred by parameters of convolution.

• Transferred matrix shape

\[ \text{transIn}_h = \text{channel}_{in} \times k_h \times k_w \]
\[ \text{transIn}_w = \text{batch}_{in} \times o_h \times o_w \]

• Output shape

\[ o_h = \left( \text{in}_h + \text{pad}_h - ((k_h - 1) \times \text{dilation}_h + 1))/\text{stride}_h \right] + 1 \]
\[ o_w = \left( \text{in}_w + \text{pad}_w - ((k_w - 1) \times \text{dilation}_w + 1))/\text{stride}_w \right] + 1 \]
Im2col and GEMM – Compression Format

- A matrix in which lots of the elements are zero.
- Compression Formats:

<table>
<thead>
<tr>
<th>Original Matrix</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 9 8 0 0</td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>7 0 0 6 0 0</td>
<td>0</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 5 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Block Matrix:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

- CSR Format:

<table>
<thead>
<tr>
<th>data</th>
<th>[9, 8, 7, 6, 5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>[2, 3, 0, 3, 2]</td>
</tr>
<tr>
<td>indptr</td>
<td>[0, 2, 4, 4, 5]</td>
</tr>
</tbody>
</table>

(a) CSR

- BSR Format (with block_size = (2, 2)):

<table>
<thead>
<tr>
<th>data</th>
<th>[0, 0, 7, 9, 8, 0, 6, 0, 0, 5, 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>[0, 1, 2]</td>
</tr>
<tr>
<td>indptr</td>
<td>[0, 2, 3]</td>
</tr>
</tbody>
</table>

(b) BSR
Conversion Flow in TVM

Algorithm 1: First conversion of relay graph

Input: Original relay graph
Output: Transformed relay graph
compress_list = []
while Node != NULL do
  if Node type == Var && weight belongs to conv2d then
    transform(weight)
    compress_list.append(weight.name)
    // replace with the new Var node in transformed weight shape.
  else if Node type == Call then
    if Call.op == conv2d() && weight.name in compress_list then
      new_Call = []
      new_Call.append(im2col())
      new_Call.append(dense())
      new_Call.append(reshape())
      // replace Call node with new_Call.
  end if
end while
Conversion Flow in TVM

**Algorithm 2:** Second conversion of relay graph

```
Input: Params, compress_list, compression_format
Output: New params and relay graph
weight_info = []
for weight in compress_list do
    if compression_format == "CSR" then
        data, indices, indptr = params[weight].to csr()
    end
    else if compression_format == "BSR" then
        data, indices, indptr = params[weight].to bsr()
    end
    del params[weight]
    params[weight+"data"] = data
    params[weight+"indices"] = indices
    params[weight+"indptr"] = indptr
    weight_info.append((weight, data.shape, indices.shape, indptr.shape))
end
return weight_info
```

Reference: Apache TVM
**Algorithm 2: Second conversion of relay graph**

- **Input:** Params, compress_list, compression_format
- **Output:** New params and relay graph
- **weight_info = []**
- **for** weight in compress_list **do**
  - **if** compression_format == "CSR" **then**
    - data, indices, indptr = params[weight].to_csr()
  - **else if** compression_format == "BSR" **then**
    - data, indices, indptr = params[weight].to_bsr()
  - del params[weight]
  - params[weight + "data"] = data
  - params[weight + "indices"] = indices
  - params[weight + "indptr"] = indptr
  - weight_info.append((weight, data.shape, indices.shape, indptr.shape))
- **end**
- **return** weight_info

*Reference: Apache TVM*
Experiments

• Environment
  • CPU: x86_64 Intel i7-9700K CPU
  • OS: Linux
  • LLVM: 9.0.1
  • TVM: 0.7.0

• Test Model Preparation
  • ImageNet based: SparseZoo
  • Cifar10 based: Pruning with distiller
    • Pruning for every layers of convolution weights
    • Using AGP Pruner
    • Retraining for 100 epochs with 0.006 lr.
Experiments – Accuracy Influence

- Model: Cifar-10 based

<table>
<thead>
<tr>
<th></th>
<th>ResNet20</th>
<th>Vgg16</th>
<th>ResNet56</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top-1</td>
<td>91.36</td>
<td>90.52</td>
<td>92.06</td>
</tr>
<tr>
<td>Top-5</td>
<td>99.72</td>
<td>99.42</td>
<td>99.70</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Top-1</td>
<td>91.18</td>
<td>90.44</td>
<td>91.84</td>
</tr>
<tr>
<td>Top-5</td>
<td>99.70</td>
<td>99.56</td>
<td>99.70</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Top-1</td>
<td>89.42</td>
<td>89.75</td>
<td>90.00</td>
</tr>
<tr>
<td>Top-5</td>
<td>99.59</td>
<td>99.57</td>
<td>99.65</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Top-1</td>
<td>87.54</td>
<td>89.75</td>
<td>90.00</td>
</tr>
<tr>
<td>Top-5</td>
<td>99.43</td>
<td>99.63</td>
<td>99.66</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Top-1</td>
<td>84.98</td>
<td>89.49</td>
<td>88.06</td>
</tr>
<tr>
<td>Top-5</td>
<td>99.21</td>
<td>99.61</td>
<td>99.58</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Top-1</td>
<td>77.35</td>
<td>88.52</td>
<td>83.92</td>
</tr>
<tr>
<td>Top-5</td>
<td>98.65</td>
<td>99.56</td>
<td>99.32</td>
</tr>
</tbody>
</table>

Accuracy (%)

Sparsity (%)
Experiments – Performance

- Model: ImageNet (SparseZoo)
- Speedups compared dense models (sparsity=0)
  - Base version: run with conv2d()
  - Sparse version: run with im2col() + sparse_dense()
- Speedups compared sparse models

Dense model

Sparse model (average): 87% 84% 88% 80% 85% 86% 86.5%
Conclusion

• We design a flow for sparse convolution in TVM.

• We create a new operator, im2col, in TVM to support sparse convolution.

• We design a visitor to replace target conv2d to im2col+GEMM.

• The performance of sparse models with compression scheme have at most 16.3x of speedup compared with direct convolution, both are without any TVM optimization.
Thank you for listening.

Q&A