An ML-Driven Autoconfigurator for Sparse Tensor Kernels in MLIR

Gus Smith* w/ Aart Bik, Penporn Koanantakool, and Mangpo Phothilimthana (Google)

*on internship from UW
1. Input
Naturally sparse, embedding lookups, sparse projection tensors, etc.

2. Activation
ReLU, dropout, etc.

3. Weight
Pruning, regularization, non-cuboid filters, etc.

4. Gradient
Pruning, gradually sending updates

(from Penporn Koanantakool)
sparsity takes many forms!

but, if we can exploit sparsity, it’s immensely beneficial!
How do we exploit sparsity?

With sparsity formats!
Indices: 
\begin{align*}
(0, 0) & \quad (0, 2) & \quad (0, 3) & \quad (2, 0) & \quad (2, 3)
\end{align*}

\begin{tabular}{c}
Values: \\
6 & 9 & 8 & 5 & 7
\end{tabular}

\begin{tabular}{c}
Row pointers: \\
0 & 3 & 3 & 5
\end{tabular}

\begin{tabular}{c}
Column indices: \\
0 & 2 & 3 & 0 & 3
\end{tabular}

\begin{tabular}{c}
Values: \\
6 & 9 & 8 & 5 & 7
\end{tabular}

\begin{tabular}{c}
Coarsegrained (COO)
\end{tabular}

\begin{tabular}{c}
Column pointers: \\
0 & 2 & 2 & 3 & 5
\end{tabular}

\begin{tabular}{c}
Row indices: \\
0 & 2 & 0 & 0 & 2
\end{tabular}

\begin{tabular}{c}
Values: \\
6 & 5 & 9 & 8 & 7
\end{tabular}

\begin{tabular}{c}
Compressed Sparse Row (CSR)
\end{tabular}

\begin{tabular}{c}
plus DIA, ELL, and more…
\end{tabular}

\begin{tabular}{c}
Compressed Sparse Column (CSC)
\end{tabular}
…and then along came TACO…
Dimension-wise Specification

- Each dimension can be sparse or dense
- Define traversal order: (1st dim, 2nd dim, ...)
- For a $k^{th}$-order tensor, this covers $k!2^k$ formats

Dense: All elements in that dimension are stored (e.g., dense row pointers in CSR).

Sparse: Zeros in that dimension aren’t stored.

(from Penporn Koanantakool)
Many possible formats for a matrix…many more for higher-dimensional tensors!

(a) Sparse matrix $A$  (b) Row-major coordinate storage tree  (c) Column-major coordinate storage tree

- Dense Rowmajor
  - Size: 3
  - Positions: [0, 2]
  - Indices: [0, 2]

- Dense Colmajor
  - Size: 4
  - Positions: [0, 3, 2, 3]
  - Indices: [0, 2, 3]

- CSR
  - Size: 3
  - Positions: [0, 3, 5]
  - Indices: [0, 3, 5]

- DCSR
  - Size: 4
  - Positions: [0, 3, 5]
  - Indices: [0, 3, 5]

- CSC
  - Size: 4
  - Positions: [0, 2, 3, 5]
  - Indices: [0, 2, 3]

- DCSC
  - Size: 3
  - Positions: [0, 2, 3]
  - Indices: [0, 2, 3]

- Row-slicing
  - Size: 4
  - Positions: [0, 3, 5]
  - Indices: [0, 3, 5]

- Col-slicing
  - Size: 3
  - Positions: [0, 2, 3]
  - Indices: [0, 2, 3]

- Sparse $d_1$, Dense $d_2$
  - Size: 3
  - Positions: [0, 3, 5]
  - Indices: [0, 3, 5]

- Sparse $d_1$, Sparse $d_2$
  - Size: 4
  - Positions: [0, 3, 5]
  - Indices: [0, 3, 5]

function @kernel(%a: tensor<?x?xf64, CSR>,
    %b: tensor<?x?xf64>,
    %c: tensor<?x?xf64>) -> tensor<?x?xf64> {

    ...kernel implementation...

    return %d : tensor<?x?xf64>
}
Goal: given a kernel, can we configure its sparse format?

couldn’t you just brute force it?
Goal: given any kernel, can we quickly configure its sparse format? so we could use it in a JIT!
We use a “standard” ML-for-ML approach:

1. Train a cost model
2. Use the cost model in a search procedure

A Learned Performance Model for Tensor Processing Units (Kaufman et. al.)
Learning to Optimize Halide with Tree Search and Random Programs (Adams et. al.)
Learning to Optimize Tensor Programs (Chen et. al.)
Training a Cost Model
What features should we extract from the configuration?

How should we extract features from sparse tensors?

What network should we use?
How should we train it?

Prediction
Configuration Features

What features should we extract from the configuration?
Configuration Features

- Entire sparse kernel configuration is easily featurized by packing into fixed-length vector.
- Per-dimension sparse formats, dimension ordering, plus other settings e.g. parallelization and vectorization levels.

For sparse-dense matmul kernel: ~9k configurations.
How should we extract features from sparse tensors?
Features

Manual Feature Extractor

Shape, rank, # of nonzeros

CNN

Features

Bridging the Gap between Deep Learning and Sparse Matrix Format Selection, Zhao et. al 2018
Sparse Matrix Classification on Imbalanced Datasets Using Convolutional Neural Networks, Pichel et. al. 2019
IA-SpGEMM: an input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication, Xie et. al. 2019

See Zhao et. al. 2018, Pichel et. al. 2019, etc
Variable-sized **sparse** input matrix

Now we can perform dense 2D convolution!
What network should we use? How should we train it?
CNN

- Feature vector generated from inputs
- Feature vector generated from parameterization

Density representation of tensor input

Multilayer perceptron over concatenated features

Prediction
9k parameterizations \( \times \) 3k tensor inputs = 27M points!

From sparse ResNet50 dataset (Sparse GPU Kernels for Deep Learning, Gale et. al. 2020)

Randomly sample and benchmark the runtimes of 50k (parameterization, input) points

Benchmarked on an Intel Xeon 6154 (Skylake), 72 CPUs

```python
model.compile(
    optimizer='adam',
    loss='mean_squared_error')
model.fit(...)
```
In related works, search follows similar structure (Chen et. al.)

In related works, search follows similar structure (Adams et. al.)
1. Randomly select test inputs
2. Randomly sample initial parameterizations
3. For each search round:
   a. Use the cost model to filter the set of candidates (fast)
   b. Benchmark the filtered set of candidates (slow)
   c. Record top benchmarked candidates
   d. Construct next generation of parameterizations
Sampler

Given p0 and p1, randomly forms a new parameterization from the fields of p0 and p1.

Initial parameterizations from start of iteration

Parameterizations filtered by cost model

Parameterizations filtered after benchmarking

All parameterizations

Initial parameterizations

Parameterizations filtered

Parameterizations filtered after benchmarking

Given p0 and p1, randomly forms a new parameterization from the fields of p0 and p1.
Future work

- Parameterize cost model over kernels
  - Generate features from kernels, e.g. via manual feature extraction or GNN
- Train on data from multiple kernels (SDDMM; random sparse kernels)
Thank you!