WACO: Learning workload-aware co-optimization of the format and schedule for a sparse tensor program

Jaeyeon Won, Charith Mendis, Joel Emer, Saman Amarasinghe
Sparse Tensors are Everywhere

Scientific Computing

Deep Learning

Graph Analytics
Writing Sparse Code is Hard

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>D</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

Compressing Non-zeros

**Sparse Data Representation**

### Compressed Representation
- **Rows:** 0 0 1 1 2 2
- **Columns:** 0 1 0 1 2 3
- **Values:** A B C D E F
Writing Sparse Code is Hard

Sparse Data Representation

Compressing Non-zeros

Skipping Ineffectual Computation
Writing Sparse Code is Hard

Sparse Data Representation

Skipping Ineffectual Computation

Different Loop Traversal
Writing Sparse Code is Hard

Sparse Data Representation

Skipping Ineffectual Computation

Different Loop Traversal

Compiler for Sparse Computation

- Tensor Algebra Compiler [Kjolstad et al.]
- Taichi [Hu et al.]
- SparseTIR [Ye et al.]
- Sparse CHiLL [Venkat et al.]
- Sparse Polyhedral Framework [Strout et al.]
- Sparse MLIR [Bik et al.]

...
Writing Sparse Code is Hard

Tensor Algebra Compiler (TACO)

Tensor Expression Language
Format Language
Scheduling Language

Imperative CPU, GPU Code
int32_t iA = A1_pos[0];
int32_t pA1_end = A1_pos[1];
while (iA < pA1_end) {
    int32_t i = A1_crd[iA];
    int32_t A1_segend = iA + 1;
    while (A1_segend < pA1_end &&
    A1_crd[A1_segend] == i){
        ...
    }
}
Writing Sparse Code is Hard

\[
y_i = A_{i,j} \times x_j
\]

Format of \(A_{i,j} = \text{CSR}\)

Scheduling Language

Tensor Algebra Compiler (TACO)

Input Sparse Matrix \(A_{i,j}\)

For \((\text{int32}_t\ i = 0;\ i < A_1\_\text{dimension};\ i++)\{
  \text{for} (\text{int32}_t\ jA=A_2\_\text{pos}[i];\ jA<A_2\_\text{pos}[i+1];\ jA++)\{
    \text{int32}_t\ j = A_2\_\text{crd}[jA];
    \vdots
  \}
\}

Runtime: 3ms
Writing Sparse Code is Hard

\[ y_i = A_{i,j} \times x_j \]

Format of \( A_{i,j} = \text{CSR} \)

Input Sparse Matrix \( A_{i,j} \)

Tensor Algebra Compiler (TACO)

\#pragma omp parallel for
for (int32_t i0=0; i0<(A1_dimension/32); i0++)
{
    for (int32_t i1=0; i1<32; i1++)
    {
        int32_t i = i0*32 + i1;
        ...
        .split(i,i0,i1,32)
        .reorder(i0,i1,j)
        .parallelize(i0)
    }
}

Runtime : 1ms
Writing **Fast** Sparse Code is Hard!

\[ y_i = A_{i,j} \times x_j \]

Input Sparse Matrix \( A_{i,j} \)

Tensor Algebra Compiler (TACO)

**Format**: ???

**Schedule**: ???

What would be the optimal format and schedule?
Writing Sparse Code is Hard

\[ y_i = A_{i,j} \times x_j \]

Tensor Algebra Compiler (TACO)

Format Language
Scheduling Language

Imperative CPU, GPU Code

(Matrix-Vector Multiply)
Writing Fast Tensor Program is Hard!

Optimization depends on tensor’s shape

Matrix 1

Dense Matrix

Optimal Loop Transformation
(Optimal Scheduling Language)

- .split(i, i1, i0, 256)
- .split(k, k1, k0, 256)
- .split(j, j1, j0, 16)
- .reorder(i1, k1, j1, i0, k0, j0)
- .unroll(k0, 4)
- .vectorize(i0)
- .parallelize(i1)

Matrix 2

- .split(i, i1, i0, 64)
- .reorder(i1, k, i0)
- .parallelize(i1)

Matrix 3

- .split(i, i1, i0, 64)
- .split(k, k1, k0, 16)
- .reorder(k1, i1, i0, k0)
- .parallelize(i1)
Writing Fast **Sparse** Tensor Program is Even Harder!

In sparse program, sparsity pattern now matters!
### Writing Fast Sparse Tensor Program is Even Harder!

- **Dense**
  - Rows: 0 0 1 1 2 2
  - Columns: 0 1 0 1 2 3
  - Values: A B C D E F

- **COO**
  - Rows: 0 0 1 1 2 2
  - Columns: 0 1 0 1 2 3
  - Values: A B C D E F

- **Sparse Matrix**
  - Rows: 0 1 2 3
  - Columns: 0 1 2 3
  - Values: A B C D E F

- **BCOO (1x2)**
  - Rows: 0 1 2 3
  - Columns: 0 1 0 1 2 3
  - Values: A B C D E F

- **CSR**
  - Rowpos: 0 2 4 6
  - Columns: 0 1 0 1 2 3
  - Values: A B C D E F

- **brows**: 0 1 2
- **bcols**: 0 0 1
- **bSize**: 2

---

### Format

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>A B C D E F</td>
</tr>
<tr>
<td>COO</td>
<td>A B C D E F</td>
</tr>
<tr>
<td>Sparse</td>
<td>A B C D E F</td>
</tr>
<tr>
<td>BCOO</td>
<td>A B C D E F</td>
</tr>
<tr>
<td>CSR</td>
<td>A B C D E F</td>
</tr>
</tbody>
</table>
Writing Fast Sparse Tensor Program is EvenHarder!

Sparse Matrix

Dense

Non-zero Splitting

COO

BCOO (1x2)

COO

CSR

rows 0 0 1 1 2 2
cols 0 1 0 1 2 3
vals A B C D E F

brows 0 1 2
bcols 0 0 1
vals A B C D E F

brows 0 1 2
bcols 0 0 1
vals A B C D E F

brows 0 2 4 6
bcols 0 1 0 1 2 3
vals A B C D E F

brows 0 2 4 6
bcols 0 1 0 1 2 3
vals A B C D E F
Writing Fast Sparse Tensor Program is Even Harder!

Sparse Matrix

Dense

COO

Row Splitting

CSR
Writing Fast Sparse Tensor Program is Even Harder!

Dense

COO

Sparse Matrix

CSR

BCOO (1x2)

Dense Block

vals A B C D E F

brows 0 1 2
bcols 0 0 1

bSize 2

rows 0 0 1 1 2 2
cols 0 1 0 1 2 3
vals A B C D E F

rowpos 0 2 4 6

cols 0 1 0 1 2 3
vals A B C D E F
Writing Fast Sparse Tensor Program is Even Harder!

In sparse program, sparsity pattern now matters!

Sparse format also matters!
Writing Fast Sparse Tensor Program is Even Harder!

Given Sparsity Pattern

Choice of Schedules
(Choice of Loop Transformations)

Choice of Formats
(Choice of Data Representations)

Given an input sparsity pattern, what is the best schedule and format?
Proposed Approach: WACO

WACO
(Workload-Aware Co-Optimization)

Co-Optimized Format and Schedule

WACO
Input sparse matrix

Co-Optimized Format and Schedule

split(i, i1, i0, 4)
split(k, k1, k0, 2)
split(j, j1, j0, 32)
reorder(i1, k1, j1, i0, k0, j0)
parallelize(i1, 48, 4)
Input sparse matrix

Cost Model (Performance Predictor)

Search Strategy

Repeat

Search Space

Format & Schedule

Best Format Schedule

Choose better candidate
Input sparse matrix

Cost Model
(Performance Predictor)

3ms?
10ms?

Search Space
Format & Schedule

Search Strategy
Choose better candidate

Repeat

Best
Format
Schedule
WACO : Search Space

1. Existing approach considers either format or schedule

2. Existing approach considers small search space
WACO: Search Space

1. Existing approach considers either format or schedule

![Diagram](TSOPF_RS_b2052_c1)

<table>
<thead>
<tr>
<th></th>
<th>SpMM</th>
<th>Format-only</th>
<th>Schedule-only</th>
<th>Co-optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1.11×</td>
<td>1.12×</td>
<td></td>
<td>2.02×</td>
</tr>
</tbody>
</table>

2. Existing approach considers small search space
WACO : Search Space

1. Existing approach considers either format or schedule

2. Existing approach considers small search space

<table>
<thead>
<tr>
<th>Format-only</th>
<th>Schedule-only</th>
<th>Co-optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1.11×</td>
<td>1.12×</td>
</tr>
</tbody>
</table>

PLDI’13 [Li et al.]
PPoPP’18 [Zhao et al.]
SC’20 [Sun et al.]

4 formats 4 formats 5 formats
WACO: Search Space

Choice of Formats
(Choice of Data Representations)

- Hash
- Bitmap
- BlockCSR
- COO
- CSR
- CSC
- CSB
- CSF
- ELL
- DIA

Choice of Schedules
(Choice of Loop Transformations)

- \( \text{split}(i, i_1, i_0, 256) \)
- \( \text{split}(k, k_1, k_0, 256) \)
- \( \text{split}(j, j_1, j_0, 16) \)
- \( \text{reorder}(i_1, k_1, j_1, i_0, k_0, j_0) \)
- \( \text{unroll}(k_0, 4) \)
- \( \text{vectorize}(i_0) \)
- \( \text{parallelize}(i_1) \)

- \( \text{split}(i, i_1, i_0, 64) \)
- \( \text{split}(k, k_1, k_0, 16) \)
- \( \text{reorder}(k_1, i_1, i_0, k_0) \)
- \( \text{parallelize}(i_1) \)

...
WACO: Search Space

SuperSchedule Template of $C_i = A_{i,k} \ast B_k$

- split(i,i1,i0,?)
- split(k,k1,k0,?)
- reorder(?,?,?,?)
- parallelize(?,?)

Compute Schedule

A.reorder(?,?,?,?)
A.lvlFormat(i1,?)
A.lvlFormat(i0,?)
A.lvlFormat(k1,?)
A.lvlFormat(k0,?)

Format Schedule

(Matrix-Vector Multiply)
WACO : Search Space

\[ \text{split}(i, i_1, i_0, ?) \]
\[ \text{split}(k, k_1, k_0, ?) \]
\[ \text{reorder}(?, ?, ?, ?, ?) \]
\[ \text{parallelize}(?, ?) \]

Compute Schedule

\[
\text{for } i \text{ in range}(32): \\
\quad \text{for } k \text{ in range}(32):
\]

Initial loop
WACO : Search Space

\begin{align*}
&\text{split}(i, i_1, i_0, 2) \\
&\text{split}(k, k_1, k_0, 2) \\
&\text{reorder}(i_1, k_1, i_0, k_0) \\
&\text{parallelize}(i_1, 4)
\end{align*}

Compute Schedule

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{#pragma omp schedule(dynamic, 4)} \\
\hline
\texttt{parallel-for i_1 in range(16):} \\
\quad \texttt{for k_1 in range(16):} \\
\quad \texttt{for i_0 in range(2):} \\
\quad \texttt{for k_0 in range(2):}
\hline
\end{tabular}
\end{center}

Transformed loop

Determines what loop transformations to apply.
WACO : Search Space

SuperSchedule Template of $C_i = A_{i,k} \ast B_k$

Compute Schedule

- split(i, i1, i0, ?)
- split(k, k1, k0, ?)
- reorder(?, ?, ?, ?)
- parallelize(?, ?)

Format Schedule

A. reorder(?, ?, ?, ?, ?)
A. lvlFormat(i1, ?)
A. lvlFormat(i0, ?)
A. lvlFormat(k1, ?)
A. lvlFormat(k0, ?)
Different Format Schedules made different formats.
WACO : Search Space

Different Format Schedules made different formats.
WACO : Search Space

A. reorder(\(i_1,i_0,k_1,k_0\))
A. lvlFormat(\(i_1,\text{Uncompressed}\))
A. lvlFormat(\(i_0,\text{Uncompressed}\))
A. lvlFormat(\(k_1,\text{Uncompressed}\))
A. lvlFormat(\(k_0,\text{Uncompressed}\))

Format Schedule

Different Format Schedules made different formats.
WACO : Search Space

Different Format Schedules made different formats.
WACO: Search Space

SuperSchedule Template of $C_i = A_{i,k} \ast B_k$

- .split(i,i1,i0,?)
- .split(k,k1,k0,?)
- .reorder(?,?,?,?)
- .parallelize(?,?)

Compute Schedule

Format Schedule

A. reorder(?,?,?,?)
A. lvlFormat(i1,?)
A. lvlFormat(i0,?)
A. lvlFormat(k1,?)
A. lvlFormat(k0,?)

1. Our space considers both format and schedule.
2. Our space contains $\sim 10^6$ SuperSchedules.
Input sparse matrix

Cost Model (Performance Predictor)

3 ms?

10 ms?

Search Strategy

1. Pick candidate from search space

Search Space

Choose better candidate

2. Return result

SuperSchedule
WACO : Cost Model

\[
\text{Format} + \text{Schedule} = \text{Sparsity Pattern} \rightarrow \text{Cost Model} \rightarrow \text{Predicted Runtime}
\]
WACO: Cost Model

Sparsity Pattern -> Feature Extractor

Super Schedule -> Program Embedder

Cost Model

Feature Embedding

Fully-Connected

Predicted Runtime
WACO : Cost Model (Pattern Feature Extractor)

Matrix 1

#Rows

Matrix 2

#Cols

Matrix 3

Dense World

[#Rows, #Cols]
WACO: Cost Model (Pattern Feature Extractor)

Matrix1

Matrix 2

Matrix 3

Matrix 1

Matrix 2

Matrix 3

Dense World

Sparse World

[#Rows, #Cols]

Is this enough?
WACO : Cost Model (Pattern Feature Extractor)

Human-crafted features

CNN after downsampling

Our Approach
(Submanifold Sparse CNN)

Feature List
- Number of Rows
- Number of Cols
- Number of Non-Zeros
- Average NNZ per row
- Min/Max NNZ per row
- ...

Resized Image (e.g., 128x128)

Original pattern (Arbitrary size)

Downsampling

Submanifold Sparse Convolutional Neural Network*

WACO: Cost Model (Pattern Feature Extractor)

Conventional Convolution

\[
\begin{array}{c}
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{array} \ast 
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array} = 
\begin{array}{c}
1 & 2 & 2 & 1 \\
2 & 3 & 3 & 1 \\
1 & 4 & 5 & 4 \\
1 & 3 & 3 & 2 \\
1 & 2 & 2 & 1 \\
\end{array}
\]

Submanifold Sparse Convolution

\[
\begin{array}{c}
1 & 1 \\
\end{array} \ast 
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array} = 
\begin{array}{c}
3 & 3 \\
5 \\
3 & 3 \\
\end{array}
\]

Nonzero area grows quickly 😞

Sparsity pattern is unchanged 😊

When we simply use a popular submanifold vision model,

Information does not propagate across distant non-zeros!
WACO : Cost Model (Pattern Feature Extractor)

1. Submanifold Sparse Conv
2. More Use of Stride Layers
3. Combines Multi-resolutional Patterns

Sparsity Pattern

WACONet

SConv 5x5, Stride1 → Global Pooling

SConv 3x3, Stride2 → Global Pooling

SConv 3x3, Stride2

SConv 3x3, Stride2

Global Pooling

Concat

Fully-Connected

Pattern Feature

128-dimension
Cost Model (Performance Predictor)

Search Space

1. Pick candidate from search space

2. Return result

Choose better candidate
Given a **sparsity pattern**, Search the **SuperSchedule** that minimizes the **cost**.
WACO: Search Strategy

Nearest-Neighbor Search

Given a query, search the point that minimizes a distance function.

We viewed our problem as a nearest neighbor search.
WACO : Search Strategy

**Nearest-Neighbor Search**

Given a query,
Search the point that minimizes a distance function.

**WACO Search**

Given a sparsity pattern,
Search the SuperSchedule that minimizes predicted runtime.

WACO is implemented with an existing NNS Library⁺.

Evaluation – Cost Model

Format + Schedule = Sparsity Pattern

Cost Model

Predicted Runtime
Evaluation – Cost Model

Sparsity Pattern → Feature Extractor → Feature Embedding → Fully-Connected Cost Model → Predicted Runtime

Super Schedule → Program Embedder
Four Feature Extractors

1. Hand-crafted features
2. Dense CNN after downsampling
3. Sparse CNN from a computer vision
   - MinkowskiNet
4. WACONet
   - More Stride Layers
Evaluation – Cost Model

(Lower the Better)

Train-Validation Loss

Loss

Epoch

0 10 20 30 40 50

HumanFeature train  HumanFeature val
DenseConv train  DenseConv val
MinkowskiNet train  MinkowskiNet val
WACONet train  WACONet val

Cost Model

Feature Extractor

Feature Embedder

Fully-Connected Program Embedder

55
Evaluation

• CPU: Intel Xeon E5-2680 v3
• Data: 975 Real-World Sparse Matrices
Evaluation

- CPU: Intel Xeon E5-2680 v3
- Data: 975 Real-World Sparse Matrices

<table>
<thead>
<tr>
<th>Kernels</th>
<th>Auto-tuner</th>
<th>Hand-Written</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Format-only</td>
<td>Schedule-only</td>
</tr>
<tr>
<td>SpMV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpMM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDDMM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTTKRP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

- CPU: Intel Xeon E5-2680 v3
- Data: 975 Real-World Sparse Matrices

<table>
<thead>
<tr>
<th>Kernels</th>
<th>Auto-tuner</th>
<th>Hand-Written</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Format-only</td>
<td>Schedule-only</td>
</tr>
<tr>
<td>SpMV</td>
<td>1.43x</td>
<td>2.32x</td>
</tr>
<tr>
<td>SpMM</td>
<td>1.18x</td>
<td>1.68x</td>
</tr>
<tr>
<td>SDDMM</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MTTKRP</td>
<td>1.27x</td>
<td>-</td>
</tr>
</tbody>
</table>

1. Outperforms all baselines on all kernels on average
2. Shows good result on 3D sparsity pattern (MTTKRP)
WACO : Summary

1. Search space considering both format and schedule.
   • Explore space with Nearest Neighbor Search.

2. WACONet with submanifold sparse convolution.
   • Avoid downsampling.
   • More stride layers identifies distant non-zeros.
Key takeaways

1. Auto-tuning pays the cost
   - 1000(100) runs needed in SpMV(SpMM) to amortize.

2. Load-balancing is crucial
   - Over 50% of matrices had improved performance with better load-balancing.

3. Increasing sparsity in dense block format can be helpful!
Key takeaways

1. Auto-tuning pays the cost
   - 1000(100) runs needed in SpMV(SpMM) to amortize.

2. Load-balancing is crucial
   - Over 50% of matrices had improved performance with better load-balancing.

3. Increasing sparsity in dense block format can be helpful!

4x4 Block (25% Fill) > 2x2 Block (100% Fill)
Future Direction

1. Auto-tuning pays the cost
   • 1000(100) runs needed in SpMV(SpMM) to amortize.

2. Load-balancing is crucial
   • Over 50% of matrices had improved performance with better load-balancing.

3. Increasing sparsity in dense block format can be helpful!
Thank you!