Software and Hardware for Sparse ML

Fredrik Kjolstad
Assistant Professor, Department of Computer Science
Stanford University
Background: Sparse Tensor Algebra Compilation

Tensor Algebra + UDFs

Tensor Format Specifications

CPUs/GPUs

Distributed Machines

Spatial Dataflow Accelerators

Fixed-Function Hardware

TACO

Autoscheduler

Schedules

[Chou et al. OOPSLA’18]
[Chou et al. PLDI’20]

[Henry and Hsu et al. OOPSLA’21]

[Kjolstad et al. OOPSLA’17]

[Ahrens et al. PLDI’22]

[Kjolstad et al. CGO’19]

[Senanayake et al. OOPSLA’20]

Other systems:
COMET [Mutlu et al. LCPC’20]
MLIR SparseTensor Dialect [Bik et al. TACO’22]

SPF [Zhao et al. arXiv’22]
SparseTIR [Ye et al. arXiv’22]
Overview

My view of sparsity

Why sparsity requires compilers and general hardware

Thesis

Unlike dense neural networks that can be reduced to GEMM, it will not be possible to reduce sparse neural networks to one optimized function
Sparsity as system connectivity
Two types of sparsity: learned and a priori

Dense Layer

Sparse Layer

Graph-Neural Network

a priori structure
Structure of input data

Sets

- No a priori connections
- Postulate fully connected layer
- Then may try to learn sparsity

Sequences

- Triangular matrices in transformers
- Recurrences in RNNs

Grids

- Pixel locality in CNNs

Graphs

Relational Data

<table>
<thead>
<tr>
<th>John</th>
<th>Jill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill</td>
<td>Kim</td>
</tr>
<tr>
<td>Kim</td>
<td>Mary</td>
</tr>
<tr>
<td>Mary</td>
<td>Kim</td>
</tr>
</tbody>
</table>
How sparse is graph/relational data? Often asymptotically sparse.

Assume an average degree of 150 (e.g., 150 friends)

Each matrix row then has 150 nonzeros

At 10,000 rows: \( \frac{150 \cdot 10,000}{10,000^2} = 1.5\% \text{ nonzeros} \)

At 100,000 rows: \( \frac{150 \cdot 100,000}{100,000^2} = 0.15\% \text{ nonzeros} \)

Matrix components: \( O(n^2) \)
Nonzeros: \( O(n) \)
Fraction of nonzeros: \( O(1/n) \)
Should we build graph frameworks or sparse tensor frameworks?

**Graph Framework**
- Lower level
- Can hand-implement fusion

**Sparse Tensor Framework**
- Higher-level natural notation
  - Same notation as in papers
  - Composes with dense/CNNs
  - Easy to compose multiple graphs
- Compiler can
  - Fuse computation
  - Reorder and tile
  - Port across machines
How general should tensor frameworks be?

**Kernel library**
- Fixed number of hand-optimized operations
- Fixes tensor formats

**Full tensor support**
- General tensor algebra
- User-defined functions
- Tensor reshapes and composition
- Portable across tensor formats
How to implement a sparse PyTorch in software and hardware

**Factorization**

\[ A = B \odot (CD) \]

\[ \text{factorize} \]

Matrix \( T = \text{gemm}(C,D); \)

Matrix \( A = \text{spelmul}(B,T); \)

**Compilation**

\[ A = B \odot (CD) \]

\[ \text{compile} \]

```c
int pA2 = 0;
for (int pB1 = B1_pos[0];
   pB1 < B1_pos[1]; pB1++) {
   int i = B1_crd[pB1];
   for (int pB2 = B2_pos[pB1];
        pB2 < B2_pos[pB1+1]; pB2++) {
      int j = B2_crd[pB2];
      double t = 0.0;
      for (int k = 0; k < 0; k++) {
        int pC2 = i * 0 + k;
        int pD2 = k * N + j;
        t += C[pC2] * D[pD2];
      }
   }
}
```
Factorization in **dense** tensor algebra

\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]

Transposes $\rightarrow$ GEMM $\rightarrow$ Transposes $\rightarrow$ GEMM $\rightarrow$ …

- Works pretty well for dense (at least on shared memory machines)
- The cost of transpose is modest
- The benefit of handwritten GEMM is large
Unlike dense neural networks that can be reduced to GEMM, it will not be possible to reduce sparse neural networks to one optimized function.
Factorization in **sparse** tensor algebra

\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]

- **Transposes → SpGEMM → Transposes → SpGEMM → …**
  - Requires sorting and potentially data structure conversion
  - Compilers better able to produce competitive code
  - Need to flatten data structures (e.g., COO triplets to pairs)

**No fusion across operation**

**Potential asymptotic complexity slowdown!**
Factorization destroys fusion

Sampled Dense-Dense Matrix Multiplication (SDDMM)

\[ O(IJK) \quad O(\text{NNZ}_B \cdot K) \]

Sparse matrices

Element-wise multiplication

This dot product need not be computed

Separate Operations

Fused Operation

64 inner product

10 inner product

Normalized Time

\[ \text{rma10} \quad \text{cant} \quad \text{cop20k} \quad \text{scircuit} \quad \text{mac-econ} \quad \text{pwtk} \]

\[ 2412x \quad 5186x \quad 24835x \quad 59496x \quad 73405x \quad 22400x \]
Factorization forces data movement

\[ A_{ij} = B_{ijk} c_k \]

Nodes

![compiler-generated kernel](ideal speedup)

![factorized with reshape](24x)

[1] Yadav et al. PLDI'22

Factorization prevents efficient user-defined function support

Average: 7.5x cost factorization

[Henry and Hsu et al. OOPSLA'21]
Compiler design for general sparse tensor operations

1. **Tensor Algebra**
   - + UDFs
   - + Tensor Reshaping
   - + Tensor Composition

2. **Sparse Compiler**

   - **Tensor Format Specifications**
   - **CPUs/GPUs**
   - **Distributed Machines**
   - **Spatial Dataflow Accelerators**
   - **Fixed-Function Hardware**

3. **Schedules**
   - **Autoscheduler**

- **Fusion**, **Reordering**, **Tiling**
- **Sparse spatial dataflow hardware with fusion**
Hardware design for general sparse tensor operations

Sparse tensor algebra accelerators must support:

1. **Generality**: arbitrary tensor algebra operations
2. **Data Structures**: dense and sparse data structures
3. **Fusion**: Fusion across operations
4. **Reordering**: Changing the order they process tensor dimensions
The Sparse Abstract Machine

- Abstract spatial dataflow machine architecture
- Supports all four properties (generality, data structures, fusion, and reordering)
- Also supports tiling, parallelization, vectorization, and bitvector wire protocols
- Implemented in a simulator and first prototype taped out next month
- Straightforward to compile tensor algebra to the sparse abstract machine
Inner-product sparse matrix multiplication

Hierarchical scanners load coordinates for different tensor dimensions

Repeater implements dimension broadcasting

Intersection filters coordinates

Multiplier

Scalar Reducer

Level Writer

水平 Writer

X: level i

compressed

X: level j

compressed

Xj coordinate stream

Dropped

Xi coordinate stream

Intersection

Multipliers

Scalars

Dropped

Xi coordinate stream

Xj coordinate stream
Gustafson sparse matrix multiplication

- Asymptotically less work, because intersection occurs earlier.
- Asymptotically more temporary memory, because it must reduce whole rows.
Fused SDDMM

\[ A_{ij} = B \odot (CD) \]

\[ O(\text{NNZ}_B \cdot K) \]
Conclusion and references

Unlike dense neural networks that can be reduced to GEMM, it will not be possible to reduce sparse neural networks to one optimized function.

**Compilation Approach**
- [Kjolstad et al. OOPSLA’17]
- [Kjolstad et al. MIT’20]

**Format Abstractions**
- [Chou et al. OOPSLA’18]
- [Chou et al. PLDI’20]

**Scheduling Language**
- [Kjolstad et al. CGO’19]
- [Senanayake et al. OOPSLA’20]

**Autoscheduling**
- [Ahrens et al. PLDI’22]

**Distributed Compilation**
- [Yadav et al. PLDI’22]
- [Yadav et al. SC’22]

**User-Defined Functions**
- [Henry and Hsu et al. OOPSLA’21]

**Sparse Abstract Machine**
- [Hsu et al. arXiv’22]

**Verification**
- [Kovach and Kjolstad arXiv’22]