Sparse Abstract Machine: Sparse Tensor Algebra as Dataflow Graphs

Olivia Hsu

In collaboration with Max Strange, Jaeyeon Won, Ritvik Sharma, Kunle Olukotun, Joel Emer, Mark Horowitz, and Fredrik Kjolstad

August 25th, 2022
Motivation for dataflow backends that support sparsity
Motivation for dataflow backends that support sparsity

Machine Learning
Motivation for dataflow backends that support sparsity

Machine Learning

Scientific Computing

Robotics

Simulations

Computational Biology
Motivation for dataflow backends that support sparsity
Motivation for dataflow backends that support sparsity

Machine Learning

Scientific Computing

Data Analytics

Machine Learning:
- Graph convolutions
- Activation functions
- Output: Drugs C, D lead to a side effect y_2

Scientific Computing:
- Robotics
- Simulations
- Computational Biology

Data Analytics:
- Movies
- Social Networks

NVIDIA

arm
Benefits of a compiler that targets dataflow accelerators
Benefits of a compiler that targets dataflow accelerators

Long Tail of Expressions (algorithm)
Benefits of a compiler that targets dataflow accelerators

Long Tail of Expressions (algorithm) \(\times\) Varying Compression Structures (format)
Benefits of a compiler that targets dataflow accelerators

- Long Tail of Expressions (algorithm)
- Varying Compression Structures (format)
- More Performant Backends (platform)
Benefits of a compiler that targets dataflow accelerators

- Long Tail of Expressions (algorithm)
- Varying Compression Structures (format)
- More Performant Backends (platform)
- Backend-Specific Transformations (schedules)
Benefits of a compiler that targets dataflow accelerators

- Long Tail of Expressions (algorithm)
- Varying Compression Structures (format)
- More Performant Backends (platform)
- Backend-Specific Transformations (schedules)

Users

Performance

Usability

Accelerators
Fusion over factorization in sparse applications

\[ \ldots = \ldots \times \times \times \times \times \ldots \]
Fusion over factorization in sparse applications

\[ \ldots = \ldots \times \bigcirc \times \bigcirc \times \bigcirc \times \ldots \]

\[ A = B \bigcirc (CD) \]
Fusion over factorization in sparse applications

\[ A = B \odot (CD) \]
Fusion over factorization in sparse applications

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

Unfused: \( \Theta(n^2k) \)

Fused: \( \Theta(\text{nnz}_B \cdot k) \)
Goal of the Sparse Abstract Machine (SAM)
Goal of the Sparse Abstract Machine (SAM)

Create a *logical abstraction* that can represent *sparse* applications for *dataflow* hardware
Goal of the Sparse Abstract Machine (SAM)

Create a *logical abstraction* that can represent *sparse* applications for *dataflow* hardware

Also an *intermediate representation (IR)* for automatic compilation
Goal of the Sparse Abstract Machine (SAM)

Create a *logical abstraction* that can represent *sparse* applications for *dataflow* hardware

Also an *intermediate representation (IR)* for automatic compilation

Describe the *entire* space of sparse tensor algebra expressions
Goal of the Sparse Abstract Machine (SAM)

Create a *logical abstraction* that can represent *sparse* applications for *dataflow* hardware

Also an *intermediate representation (IR)* for automatic compilation

Describe the *entire* space of sparse tensor algebra expressions

Develop and generate *efficient hardware implementations*, namely the Onyx CGRA
Abstract tensor data model (fiber trees)
Abstract tensor data model (fiber trees)

Dimension $j$

0 (0 1 0 0),

1 (2 0 3 0),

2 (0 0 0 0),

3 (0 4 0 5)

(point (3,1))
Abstract tensor data model (fiber trees)

Dimension $i$

<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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Dimensions $i$ and $j$

Values

Segments

Coordinates

Point (3,1)
Abstract tensor data model (fiber trees)

Dimensions:
- Dimension i
- Dimension j

Values:
- (0, 1, 3)
- ((1), (0, 2), (1, 3))
- ((1), (2, 3), (4, 5))

Segments and Coordinates:
- Segments: Root → i → j
- Coordinates: (0, 1, 3), ((1), (0, 2), (1, 3)), ((1), (2, 3), (4, 5))
Tensors in SAM

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))
Tensors in SAM

Arrays (Space)

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))
Tensors in SAM

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

Arrays (Space)

Segments

Coordinates

Values

0 1 3

0 1 2 1 3

1 2 3 4 5
Tensors in SAM

Streams (Time)

((0, 1, 3))

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

Arrays (Space)

Segments

Coordinates

Values

((1), (2, 3), (4, 5))
Tensors in SAM

Streams (Time)

Arrays (Space)

Segments

Coordinates

Values

D, S₀, 3, 1, 0
Tensors in SAM

Values:
1 2 3 4 5

Segments:
0 3

Coordinates:
0 1 3

Streams (Time):
((1), (0, 2), (1, 3))

Arrays (Space):
((1), (2, 3), (4, 5))

Hierarchical stop token denotes end of a segment.
Tensors in SAM

Non-control tokens:

Streams (Time):

Arrays (Space):

Hierarchical stop token denotes end of a segment.
Tensors in SAM

Non-control tokens

Hierarchical stop token denotes end of a segment

Done token denotes end of a stream

Values

Segments

Coordinates

Streams (Time)

Arrays (Space)

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))
Sparse-matrix sparse-matrix multiplication in SAM
Sparse-matrix sparse-matrix multiplication in SAM
Sparse-matrix sparse-matrix multiplication in SAM

Hierarchical Streams

Data flows through computation spatially
Sparse-matrix sparse-matrix multiplication in SAM

Data flows through computation spatially

Hierarchical sparse iteration

Hierarchical Streams
Sparse-matrix sparse-matrix multiplication in SAM

Format agnostic
Hierarchical sparse iteration

Hierarchical Streams

Data flows through computation spatially
Sparse-matrix sparse-matrix multiplication in SAM

Data flows through computation spatially

Hierarchical Streams

Format agnostic
Hierarchical sparse iteration

Stream Merging
Sparse-matrix sparse-matrix multiplication in SAM

Format agnostic
Hierarchical sparse iteration

Tensor broadcasting

Hierarchical Streams
Data flows through computation spatially

Stream Merging
Sparse-matrix sparse-matrix multiplication in SAM

Hierarchical Streams

Data flows through computation spatially

Format agnostic
Hierarchical sparse iteration

Tensor broadcasting

Stream Merging

Computation

Hierarchical Streams

Vector Reduce

Multiply

Array C Values
vals
vals

Xj coordinate stream

crd

Repeat
B: level j

ref

Array B Values
vals
vals

Xi coordinate stream

crd

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed

Segment Write
X: values compressed

Segment Write
X: level i compressed

Segment Write
X: level j compressed
Sparse-matrix sparse-matrix multiplication in SAM
Sparse-matrix sparse-matrix multiplication in SAM

Format agnostic
Hierarchical sparse iteration

Hierarchical Streams
Data flows through computation spatially

Stream Merging

Computation

Tensor Construction

Sparse outer-level accumulation

Tensor broadcasting

Dropped

Segment Scan
B: level i compressed

Segment Scan
B: level k compressed

Repeat
ref

Repeat
ref

Segment Scan
C: level i compressed

Segment Scan
C: level k compressed

Segment Scanner
C: level j compressed

Intersect
ref

ref

ref1

ref1

ref2

ref2

ref

ref

ref

ref

ref

ref

ref

ref

ref

ref

ref

ref

Segment Write
X: level i compressed

Segment Write
X: values compressed

Segment Write
X: level j compressed

Xj coordinate stream

Xi coordinate stream

Xj coordinate stream

Xi coordinate stream

Array
B Values

Array
C Values

Multiply
vals

Vector Reduce
vals

Array
vals

Array
vals

Array
vals

Xj coordinate stream

Xi coordinate stream

Computation

Tensor Construction

Format agnostic
Hierarchical sparse iteration

Hierarchical Streams
Data flows through computation spatially

Stream Merging

Computation

Tensor Construction
Inner-product algorithm SAM graph
Inner-product algorithm SAM graph
Inner-product algorithm SAM graph

Intersection at last dataflow level

Scalar reducer, less memory
System overview

Format Language
(Chou et al., 2018)

Index Notation

Concrete Index Notation
(Kjolstad et al., 2020)

Scheduling Language
(Senanayake et al., 2020)

Custard:
Front-end Compiler

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs
System overview

Format Language
(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Format Language

Index Notation

Scheduling Language

Concrete Index Notation

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Chou et al., 2018
Senanayake et al., 2020
Kjolstad et al., 2020

Concrete Index Notation

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language

(Chou et al., 2018)

Index Notation
(Senanayake et al., 2020)

Scheduling Language
(Custard: Front-end Compiler)

Concrete Index Notation
(Kjolstad et al., 2020)

Sparse Abstract Machine

Binding

Simulator

Prior Work Architectures

New RDAs and CGRAs

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK
CSR
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
Compressed Sparse
Tensors
Blocked Tensors
Linked Lists
Database
Compression Schemes
Cloud Storage

reorder
precompute
parallelize  split
map  divide
vectorize  unroll
position

Language
Compiler flow

Expression

\[ X_{ij} = B_{ik} \times C_{kj} \]

Format Language

\[ B = (\{\text{sparse}, \text{sparse}\}, \{\text{mode0}, \text{model}\}) \]

\[ C = (\{\text{sparse}, \text{sparse}\}, \{\text{model}, \text{mode0}\}) \]

Schedule

\[ \text{reorder}(i, k, j) \]
Compiler flow

Expression

$X_{ij} = B_{ik} \ast C_{kj}$

Format Language

$B=\{\text{sparse,sparse}\}, \{\text{mode0,mode1}\}$

$C=\{\text{sparse, sparse}\}, \{\text{mode1, mode0}\}$

Schedule

reorder($i, k, j$)

$\forall i \forall k \forall j X_{ij} = \sum_k (B_{ik} \ast C_{kj})$
Compiler flow

Expression

\[ X_{ij} = B_{ik} \ast C_{kj} \]

Format Language

\( B=\{\text{sparse, sparse}\}, \{\text{mode0, mode1}\} \)

\( C=\{\text{sparse, sparse}\}, \{\text{mode1, mode0}\} \)

Schedule

\( \text{reorder}(i, k, j) \)

\[ \forall_i \forall_k \forall_j X_{ij} = \Sigma_k (B_{ik} \ast C_{kj}) \]

Intersect

Repeat

\( B_i \times |j| \)

\( C_j \times |i| \)

Repeat

\( B_k \ast B_i \ast C_k \ast C_j \ast X_j \ast X_i \)
Compiler flow

Expression

\[ X_{ij} = B_{ik} \times C_{kj} \]

Format Language

\( B = (\{\text{sparse, sparse}\}, \{\text{mode0, mode1}\}) \)

\( C = (\{\text{sparse, sparse}\}, \{\text{mode1, mode0}\}) \)

Schedule

\( \text{reorder}(i, k, j) \)

\[ \forall i \forall k \forall j \text{ } X_{ij} = \sum_k (B_{ik} \times C_{kj}) \]

Segment Write \( X_i \)

Repeat \( C_{\text{root}} \times |i| \)

Segment Scan \( B_k \)

Intersect \( B_i \times \text{|j|} \)

Repeat \( C_j \)

Segment Scan \( C_k \)

Segment Write \( X_j \)

Lower to Example Implementation
Optimization extensions
Optimization extensions

• Tensor Locating
  • Coordinate skipping
Optimization extensions

• Tensor Locating
  • Coordinate skipping

• Parallelization
  • Bit vectors as streams (instead of coordinates)
  • Parallelize/Serialize
Optimization extensions

• Tensor Locating
  • Coordinate skipping

• Parallelization
  • Bit vectors as streams (instead of coordinates)
  • Parallelize/Serialize
Optimization extensions

• Tensor Locating
  • Coordinate skipping

• Parallelization
  • Bit vectors as streams (instead of coordinates)
  • Parallelize/Serialize

\[ D, S, 12, 11, 9, 3, 2, 0 \]

Assuming 4-bit elements

\[ D, S, 1000, 0101, 0000, 1011 \]
Optimization extensions

• Tensor Locating
  • Coordinate skipping
• Parallelization
  • Bit vectors as streams (instead of coordinates)
  • Parallelize/Serialize
• Splitting and Flattening

D, S, 12, 11, 9, 3, 2, 0
Assuming 4-bit elements
D, S, 1000, 0101, 0000, 1011
Optimization extensions

• Tensor Locating
  • Coordinate skipping

• Parallelization
  • Bit vectors as streams (instead of coordinates)
  • Parallelize/Serialize

• Splitting and Flattening

\[ D, S, 12, 11, 9, 3, 2, 0 \]
Assuming 4-bit elements

\[ D, S, 1000, 0101, 0000, 1011 \]
Fusion is key

\[ A = B \odot (CD) \]
Fusion is key

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

*Unfused: \( \Theta(n^2k) \)*

*Fused: \( \Theta(\text{nnz}_B \cdot k) \)*

*Increasing K dimension for Fusion*

---

**Cycles**

- **Unfused**
- **Fused coiteration**
- **Fused locating**

---

10^6

10^5

10^4

1
10
100

---

13
Dataflow matters for asymptotic behavior too
Optimizations depend on the data

![Graph showing the relationship between cycles and number of nonzeros for different data representations and optimization techniques. The graph includes data for Crd, Dense, Crd w/ Skip, Crd w/ Split, BV w/ Split, and BV.]
Optimizations depend on the data

Uniform Random Data

- Crd
- Dense
- Crd w/ Skip
- Crd w/ Split
- BV w/ Split
- BV

Data with Runs

- Crd
- Dense
- Crd w/ Skip
- Crd w/ Split
- BV w/ Split
- BV
Impact and Future Work
Impact and Future Work

• Targets both fixed-function and reconfigurable dataflow accelerators
Impact and Future Work

• Targets both fixed-function and reconfigurable dataflow accelerators
• Unified abstraction for apples-to-apples comparison of sparse techniques and designs
Impact and Future Work

• Targets both fixed-function and reconfigurable dataflow accelerators
• Unified abstraction for apples-to-apples comparison of sparse techniques and designs
• End-to-end compilation and hardware generation
Impact and Future Work

• Targets both fixed-function and reconfigurable dataflow accelerators
• Unified abstraction for apples-to-apples comparison of sparse techniques and designs
• End-to-end compilation and hardware generation
• Generating the sparse design of Onyx with SAM
Impact and Future Work

• Targets both fixed-function and reconfigurable dataflow accelerators
• Unified abstraction for apples-to-apples comparison of sparse techniques and designs
• End-to-end compilation and hardware generation

• Generating the sparse design of Onyx with SAM
• Outer-level blocking and tiling for hardware memory systems
Impact and Future Work

- Targets both fixed-function and reconfigurable dataflow accelerators
- Unified abstraction for apples-to-apples comparison of sparse techniques and designs
- End-to-end compilation and hardware generation
- Generating the sparse design of Onyx with SAM
- Outer-level blocking and tiling for hardware memory systems
- Front-end compiler and simulator improvements
Impact and Future Work

• Targets both fixed-function and reconfigurable dataflow accelerators
• Unified abstraction for apples-to-apples comparison of sparse techniques and designs
• End-to-end compilation and hardware generation

• Generating the sparse design of Onyx with SAM
• Outer-level blocking and tiling for hardware memory systems
• Front-end compiler and simulator improvements
• SAM abstraction for sparse machine learning models