Arrays 2.0: Extending The Scope Of The Array Abstraction

Saman Amarasinghe

Fredrik Kjolstad (Stanford)    Willow Ahrens (MIT)
Jaeyeon Won (MIT)              Daniel Donenfeld (MIT)
Stephen Chou (MIT)             Ryan Senanayake (MIT)
Rawn Henry (MIT)               Shoaib Kamil (Adobe)
Olivia Hsu (Stanford)          David Lugato (CEA)
Rohan Yadav (Stanford)         Charith Mendis (UIUC)
Changwan Hong (MIT)            Joel Emer (MIT)
Array Programming Is Productive

\[
\begin{align*}
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\end{align*}
\begin{align*}
\begin{bmatrix}
9 & 9 & 9 \\
9 & 9 & 9 \\
9 & 9 & 9
\end{bmatrix}
\end{align*}
= 
\begin{align*}
\begin{bmatrix}
.1 & .2 & .3 \\
.4 & .5 & .7 \\
.8 & .9 & 1
\end{bmatrix}
\end{align*}
\quad \text{normalization}
\]

\[
\begin{align*}
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\end{align*}
\begin{align*}
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1
\end{bmatrix}
\end{align*}
= 
\begin{align*}
\begin{bmatrix}
-1 & 0 & 3 \\
-4 & 0 & 6 \\
-7 & 0 & 9
\end{bmatrix}
\end{align*}
\quad \text{multiplying several columns at once}
\]

\[
\begin{align*}
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\end{align*}
\begin{align*}
\begin{bmatrix}
3 & 3 & 3 \\
6 & 6 & 6 \\
9 & 9 & 9
\end{bmatrix}
\end{align*}
= 
\begin{align*}
\begin{bmatrix}
.3 & .7 & 1. \\
.6 & .8 & 1. \\
.8 & .9 & 1.
\end{bmatrix}
\end{align*}
\quad \text{row-wise normalization}
\]

\[
\begin{align*}
\begin{bmatrix}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3
\end{bmatrix}
\end{align*}
\begin{align*}
\begin{bmatrix}
1 & 1 \\
2 & 2 \\
3 & 3
\end{bmatrix}
\end{align*}
= 
\begin{align*}
\begin{bmatrix}
1 & 2 & 3 \\
2 & 4 & 6 \\
3 & 6 & 9
\end{bmatrix}
\end{align*}
\quad \text{outer product}
\]
Array Programming Is Productive

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

/  
<table>
<thead>
<tr>
<th>9</th>
<th>9</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

=  
<table>
<thead>
<tr>
<th>.1</th>
<th>.2</th>
<th>.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>.4</td>
<td>.5</td>
<td>.7</td>
</tr>
<tr>
<td>.8</td>
<td>.9</td>
<td>1</td>
</tr>
</tbody>
</table>

normalization

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

*  
<table>
<thead>
<tr>
<th>-1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

=  
<table>
<thead>
<tr>
<th>-1</th>
<th>0</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>-7</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

multiplying several columns at once

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

/  
<table>
<thead>
<tr>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

=  
<table>
<thead>
<tr>
<th>.3</th>
<th>.7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>.6</td>
<td>.8</td>
<td>1</td>
</tr>
<tr>
<td>.8</td>
<td>.9</td>
<td>1</td>
</tr>
</tbody>
</table>

row-wise normalization

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

*  
<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

=  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

outer product

Matrix multiplication  
\[
c_{ik} = \sum_j a_{ij} b_{jk}
\]

c = np.einsum('ij,jk->ik', a, b)

Tensor multiplication  
\[
c_{ijlm} = \sum_k a_{ijk} b_{klm}
\]

c = np.einsum('ijk,klm->ijlm', a, b)
Array Programming Is Productive

Matrix multiplication

\[ c_{ik} = \sum_j a_{ij} b_{jk} \]

Tensor multiplication

\[ c_{ijlm} = \sum_k a_{ijk} b_{klm} \]

\[ c = \text{np.einsum}(\text{'ij,k->ik'}, a, b) \]

\[ c = \text{np.einsum}(\text{'ijk,klm->ijlm'}, a, b) \]
Arrays Are Fast

• Huge investments
• Cache Blocking and Tiling
• Loop unrolling
• Vectorization
• Multicore Parallelization
• Communication-avoiding algorithms
• Often at 70-90 % of peak!

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures.
Arrays Are The Oldest Abstraction...

FORTRAN had Multidimensional arrays in 1957

real :: x(14)  
real :: T(8, 13, 11)

integer, dimension(16, 14) :: A

GEMM

https://www.netlib.org/blas/#_reference_blas_version_3_11_0
... And Arrays Haven’t Changed Much Since

FORTRAN had Multidimensional arrays in 1957

```
real :: x(14)
real :: T(8, 13, 11)
integer, dimension(16, 14) :: A
```

```
GEMM

* Form C := alpha*A*B + beta*C.
* 
DO 90 J = 1,N
   IF (BETA.EQ.ZERO) THEN
      DO 50 I = 1,M
         C(I,J) = ZERO
      50 CONTINUE
   ELSE IF (BETA.NE.ONE) THEN
      DO 60 I = 1,M
         C(I,J) = BETA*C(I,J)
      60 CONTINUE
   END IF
   DO 80 L = 1,K
      TEMP = ALPHA*B(L,J)
      DO 70 I = 1,M
         C(I,J) = C(I,J) + TEMP*A(I,L)
      70 CONTINUE
   80 CONTINUE
   DO 90 J = 1,N
DO 90 CONTINUE
```

https://www.netlib.org/blas/#_reference_blas_version_3_11_0
Arrays Are

• Multi-dimensional
• Rectilinear
• Dense
• Integer grid

Of points

GEMM

* Form C := alpha*A*B + beta*C.
* DO 90 J = 1,N
  IF (BETA.EQ.ZERO) THEN
    DO 50 I = 1,M
      C(I,J) = ZERO
    50 CONTINUE
  ELSE IF (BETA.NE.ONE) THEN
    DO 60 I = 1,M
      C(I,J) = BETA*C(I,J)
    60 CONTINUE
  END IF
  DO 80 L = 1,K
    TEMP = ALPHA*B(L,J)
    DO 70 I = 1,M
      C(I,J) = C(I,J) + TEMP*A(I,L)
    70 CONTINUE
  80 CONTINUE
90 CONTINUE

https://www.netlib.org/blas/#__reference_blas_version_3_11_0
The World Is Not Dense
The World Is Not Dense

Scientific Computing

dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

A = 

LDA
The World Is Not Dense

Scientific Computing

dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

LDA

\[
A = \begin{bmatrix}
    u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\
    u_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\
    \vdots & \ddots & \ddots & \ddots & \vdots \\
    \vdots & & \ddots & \ddots & \ddots \\
    0 & \cdots & \cdots & u_{n-1,n} & u_{n,n}
\end{bmatrix}
\]

dtrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
The World Is Not Dense

Scientific Computing

dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

\[
A = \begin{bmatrix}
  u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\
  u_{2,2} & u_{2,3} & \cdots & \cdots & u_{2,n} \\
  \vdots & \vdots & \ddots & \ddots & \vdots \\
  \vdots & \vdots & \ddots & u_{n-1,n} & u_{n,n} \\
  0 & u_{1,1} & \cdots & \cdots & u_{n,n}
\end{bmatrix}
\]

dtrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)

dgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
The World Is Not Dense

Scientific Computing

dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

\[
A = \begin{bmatrix}
  u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\
  u_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  \vdots & \vdots & \vdots & \ddots & u_{n-1,n} \\
  0 & 0 & 0 & \cdots & u_{n,n}
\end{bmatrix}
\]

dtrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)

dgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

dsymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
The World Is Not Dense

Scientific Computing

\[
\begin{align*}
dgmm(&\text{TRANSA, TRANSB}, M, N, K, \text{ALPHA, A, LDA, B, LDB, BETA, C, LDC}) \\
dtrmm(&\text{SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB}) \\
dgbbmv(&\text{TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY}) \\
dsymm(&\text{SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC})
\end{align*}
\]

Networks

\[
\begin{align*}
A &= \\
A &= 1 - d \frac{N}{N} + \sum_{j} dA_{ij} r_i \\
\end{align*}
\]
The World Is Not Dense

Scientific Computing

dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

dtrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)

dsyrk(SIDE, UPLO, TRANS, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

dsymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

dgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

dgemv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

Networks

\[ r_i = \frac{1 - d}{N} + \sum_j dA_{ij} r_i \]

\[ A = \begin{bmatrix}
    u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\
    u_{2,1} & u_{2,2} & \cdots & u_{2,n} \\
    \vdots & \vdots & \ddots & \vdots \\
    u_{n-1,1} & u_{n-1,2} & \cdots & u_{n-1,n} \\
    0 & u_{n,2} & \cdots & u_{n,n}
\end{bmatrix} \]

block sparse:
The World Is Not Dense

Scientific Computing

\[
dgemm(\text{TRANSA, TRANSB}, M, N, K, \text{ALPHA, A, LDA, B, LDB, BETA, C, LDC})
\]

\[
A = \begin{bmatrix}
\begin{array}{ccccc}
  u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\
  u_{2,2} & u_{2,3} & \cdots & & \\
  \vdots & \ddots & \ddots & \ddots & \\
  \vdots & & \ddots & \ddots & \ddots \\
  0 & \cdots & \cdots & u_{n-1,n} & u_{n,n}
\end{array}
\end{bmatrix}
\]

\[
dtrmm(\text{SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB})
\]

\[
r_i = \frac{1 - d}{N} + \sum_j dA_{ij}r_i
\]

Networks

\[
dsymm(\text{SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC})
\]

\[
dgbmv(\text{TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX}, \text{BETA, Y, INCY})
\]

run-length encoding:

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

Image Processing

block sparse:
The World Is Not Dense

Scientific Computing

dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

\[ A = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ u_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & u_{n-1,n} \\ 0 & \cdots & 0 & \cdots & u_{n,n} \end{bmatrix} \]

dtrmm(SIDE, UPLO, DIAG, M, N, ALPHA, A, LDA)

dbgmmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

dsymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

Mathematical Optimization

run-length encoding:

convolution (Toeplitz):

Networks

\[ r_i = \frac{1 - d_i}{N} + \sum_j dA_{ij}r_i \]

Image Processing

\[ A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 \end{bmatrix} \]

\[ r_i = 1 - d_i + N + \sum_j dA_{ij}r_i \]

\[ A = \begin{bmatrix} 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \end{bmatrix} \]

\[ r_i = 1 - d_i + N + \sum_j dA_{ij}r_i \]

\[ A = \begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix} \]
The World Is Not Dense

Scientific Computing

dgmm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

\[ A = \begin{bmatrix}
  u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\
  u_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  u_{n-1,1} & u_{n-1,2} & \cdots & u_{n-1,n} & u_{n,n}
\end{bmatrix} \]

r_i = \frac{1 - d}{N} + \sum_j dA_{ij}r_i

Networks

dtrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)

\[ A = \begin{bmatrix}
  B_{11} & B_{12} & 0 & \cdots & 0 \\
  B_{21} & B_{22} & B_{23} & \cdots & \vdots \\
  0 & B_{32} & B_{33} & \cdots & \vdots \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & B_{43} & \cdots & B_{44} \\
  0 & 0 & 0 & \cdots & B_{54}
\end{bmatrix} \]

dgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

Mathematical Optimization

dsymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

Image Processing

run-length encoding:

\[ \begin{bmatrix}
  3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \\
  3 & 1 & 2 & 5 & 2 & 4
\end{bmatrix} \]

convolution (Toeplitz):

\[ y_i = \sum_j s_{-j}A_{ij}x_j \]

padding:

\[ \begin{bmatrix}
  0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \\
  3 & 1 & 2 & 5 & 2 & 4
\end{bmatrix} \]
Arrays Are

• Multi-dimensional
• Rectilinear
• Dense
• Integer grid

Of points
Arrays Are

• Multi-dimensional
• Rectilinear

Dense
• Integer grid

Of points
For Example, Sparse Tensors Are Everywhere

Data Analytics
- Movies
- Social Networks
- Product Reviews

Machine Learning
- Sparse Networks
- Sparse Convolutional Networks
- Graph Convolutional Network

Science and Engineering
- Robotics
- Computational Biology
- Simulations

Graph Convolutional Network
For Example, Sparse Tensors Are Everywhere

Data Analytics
- Movies
- Social Networks
- Product Reviews

Machine Learning
- Sparse Convolutional Networks
- Sparse Networks
- Graph Convolutional Network

Science and Engineering
- Robotics
- Simulations
- Computational Biology

Figure 1: Visualization of random dense and random block-sparse weight matrices, where white indicates a weight of zero. Our new kernels allow efficient usage of block-sparse weights in fully connected and convolutional layers, as illustrated in the middle figure. For convolutional layers, the kernels allow for sparsity in input and output feature dimensions; the connectivity is still dense in the spatial dimensions. The sparsity is defined at the level of blocks (right figure), with block size of at least \(8 \times 8\). At the block level, the sparsity pattern is completely configurable. Since the kernels skip computations of blocks that are zero, the computational cost is only proportional to the number of weights, not the number of input/output features.

Figure 2: Dense linear layers (left) can be replaced with layers that are sparse and wider (center) or sparse and deeper (right) while approximately retaining computational cost and memory cost. Note these costs are, in principle, proportional to the number of non-zero weights (edges). The shown networks have an equal number of edges. However, the sparse and wide network has the potential advantage of a larger information bandwidth, while the deeper network has the potential benefit of fitting nonlinear functions.

Block-sparsity unlocks various research directions (see section 6). One application we explore in experiments is the widening or deepening of neural networks, while increasing sparsity, such that the computational cost remains approximately equal as explained in figure 2. In experiments we have only scratched the surface of the applications of block-sparse linear operations; by releasing our kernels in the open, we aim to spur further advancement in model and algorithm design.

2 Capabilities

The two main components of this release are a block-sparse matrix multiplication kernel and a block-sparse convolution kernel. Both are wrapped in Tensorflow [Abadi et al., 2016] ops for easy use and the kernels are straightforward to integrate into other frameworks, such as PyTorch. Both kernels support an arbitrary block size and are optimized for 8x8, 16x16, and 32x32 block sizes. The matrix multiplication kernel supports an arbitrary block layout which is specified via a masking matrix. In addition, the feature axis is configurable. The convolution kernel supports non-contiguous input/output feature blocks of any uniform or non-uniform size specified via a configuration format (see API) though multiples of 32x32 perform best. Arbitrary dense spatial filter sizes are supported in addition to dilation, striding, padding, and edge biasing.
For Example, Sparse Tensors Are Everywhere

Data Analytics
- Movies
- Social Networks

Machine Learning
- Sparse Convolutional Networks
- Sparse Networks
- Graph Convolutional Network

Science and Engineering
- Robotics
- Simulations

Amazon

Figure 1: Visualization of random dense and random block-sparse weight matrices, where white indicates a weight of zero. Our new kernels allow efficient usage of block-sparse weights in fully connected and convolutional layers, as illustrated in the middle figure. For convolutional layers, the kernels allow for sparsity in input and output feature dimensions; the connectivity is still dense in the spatial dimensions. The sparsity is defined at the level of blocks (right figure), with block size of at least 8x8. At the block level, the sparsity pattern is completely configurable. Since the kernels skip computations of blocks that are zero, the computational cost is only proportional to the number of weights, not the number of input/output features.

Figure 2: Dense linear layers (left) can be replaced with layers that are sparse and wider (center) or sparse and deeper (right) while approximately retaining computational cost and memory cost. Note these costs are, in principle, proportional to the number of non-zero weights (edges). The shown networks have an equal number of edges. However, the sparse and wide network has the potential advantage of a larger information bandwidth, while the deeper network has the potential benefit of fitting nonlinear functions.

Block-sparsity unlocks various research directions (see section 6). One application we explore in experiments is the widening or deepening of neural networks, while increasing sparsity, such that the computational cost remains approximately equal as explained in figure 2. In experiments we have only scratched the surface of the applications of block-sparse linear operations; by releasing our kernels in the open, we aim to spur further advancement in model and algorithm design.

2 Capabilities
The two main components of this release are a block-sparse matrix multiplication kernel and a block-sparse convolution kernel. Both are wrapped in Tensorflow [Abadi et al., 2016] ops for easy use and the kernels are straightforward to integrate into other frameworks, such as PyTorch. Both kernels support an arbitrary block size and are optimized for 8x8, 16x16, and 32x32 block sizes. The matrix multiplication kernel supports an arbitrary block layout which is specified via a masking matrix. In addition, the feature axis is configurable. The convolution kernel supports non-contiguous input/output feature blocks of any uniform or non-uniform size specified via a configuration format (see API) though multiples of 32x32 perform best. Arbitrary dense spatial filter sizes are supported in addition to dilation, striding, padding, and edge biasing.
For Example, Sparse Tensors Are Everywhere

Data Analytics

Machine Learning

Science and Engineering

Movies

Sparse Convolutional Networks

Social Networks

Sparse Networks

Customers

Extremely sparse
Dense storage: 107 Exabytes
Sparse storage: 13 Gigabytes

Products

Graph Convolutional Network

Words

Amazon

Amazing

Great

Peter

Lily

Paul

Billy

Hide

Bob

Sam

Mary

Monitor

Laptop

Camera

Shoes

The Jedi

Kites

Dinosaurs
Dense Tensors Are Flexible But Can Waste Memory

<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></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dense Tensors Are Flexible But Can Waste Memory
Dense Tensors Are Flexible But Can Waste Memory
Dense Tensors Are Flexible But Can Waste Memory

\[
\text{locate}(1,2) = 1 \times 4 + 2 = 6
\]
Sparse Tensors Can Be Compressed By Adding Metadata

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

0 1 2 3
Sparse Tensors Can Be Compressed By Adding Metadata
Sparse Tensors Can Be Compressed By Adding Metadata

row(3) = ???
col(3) = ???
Sparse Tensors Can Be Compressed By Adding Metadata

Coordinate:

<table>
<thead>
<tr>
<th>rows</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cols</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

A    B    C    D    E    F

0    1    2    3    4    5

0  | A | B |
1  | C | D | E |
2  |   | F |   |
Sparse Tensors Can Be Compressed By Adding Metadata

Coordinate

<table>
<thead>
<tr>
<th>rows</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cols</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

A B C D E F

0 1 2 3 4 5

0 1 2 3

A B 2 3

C D E

F
Sparse Tensors Can Be Compressed By Adding Metadata

Coordinate Duplicates

rows: [0, 0, 1, 1, 1, 2]
cols: [0, 2, 1, 2, 3, 3]

A  B  C  D  E  F
0  1  2  3  4  5
Sparse Tensors Can Be Compressed By Adding Metadata

Compressed Sparse Rows (CSR)

pos: 0 2 5 6

cols: 0 2 1 2 3 3

A B C D E F
0 1 2 3 4 5

0 1 2 3
0 A B
1 C D E
2 F

11
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        int pB2 = i*n + j;
        int pA2 = i*n + j;
        double t = 0.0;
        for (int k = 0; k < o; k++) {
            int pB3 = pB2*o + k;
            t += B[pB3] * c[k];
        }
        A[pA2] = t;
    }
}
\[ A_{ij} = \sum_k B_{ijk} c_k \]

for (int pA = 0; pA < m*n; pA++) {
    A[pA] = 0.0;
}

for (int pB1 = B1_pos[0]; pB1 < B1_pos[1]; pB1++) {
    int i = B1_crd[pB1];
        int j = B2_crd[pB2];
        int pA2 = i*n + j;
        double t = 0.0;
        for (int pB3 = B3_pos[pB2]; pB3 < B3_pos[pB2+1]; pB3++) {
            int k = B3_crd[pB3];
            t += B[pB3] * c[k];
        }
        A[pA2] = t;
    }
}
\[ A_{ij} = \sum_{k} B_{ijk} C_k \]

for (int pA = 0; pA < m*n; pA++) {
    A[pA] = 0.0;
}

for (int pB1 = B1_pos[0]; pB1 < B1_pos[1]; pB1++) {
    int i = B1_crd[pB1];
        int j = B2_crd[pB2];
        int pA2 = i*n + j;
        double t = 0.0;
        int pB3 = B3_pos[pB2];
        int pc1 = c1_pos[0];
        while (pB3 < B3_pos[pB2+1] && pc1 < c1_pos[1]) {
            int kB = B3_crd[pB3];
            int kc = c1_crd[pc1];
            int k = min(kB, kc);
            if (kB == k && kc == k) {
                t += B[pB3] * c[pc1];
            }
            pB3 += (int)(kB == k);
            pc1 += (int)(kc == k);
        }
        A[pA2] = t;
    }
}
\[ A_{ijk} = B_{ijk} + C_{ijk} \]

Complexity Of Sparse Tensors & Code

```
int iB = 0;
int CR_pos = CR_pos[0];
while (CR_pos < CR_pos[1]) {
    int iC = CR_crd[CR_pos];
    CR_pos = CR_pos[1] + 1;
    if (iC == iB) C0_pos = C0_pos[0];
    while (C0_pos < C0_pos[1]) {
        int iC = C0_crd[C0_pos];
        C0_end = C0_pos + 1;
        if (iC == iB) {
            int B1_pos = B1_pos[iB];
            int C1_pos = C0_pos;
            while (B1_pos < B1_pos[iB + 1]) {
                int jB = B1_crd[B1_pos];
                int jC = C1_crd[C1_pos];
                int j = min(jB, jC);
                int A1_pos = (iB * A1_size) + j;
                int C1_end = C1_pos + 1;
                if (jC == j) while (C1_end < C0_end) { C1_end++; }
                if ((jB == j) && (jC == j)) {
                    int B2_pos = B2_pos[B1_pos];
                    int C2_pos = C1_pos;
                    while (B2_pos < B2_pos[B1_pos + 1]) {
                        int kB = B2_crd[B2_pos];
                        if ((kB == j) && (kB == k)) {
                            int A2_pos = (A1_pos * A2_size) + kB;
                            A2_pos = (A1_pos * A2_size) + kB;
                            if (kB == k) B2_pos++;
                            if (kC == k) C2_pos++;
                        }
                    }
                }
                if (iC == iB) C0_pos = C0_end;
            }
            iB++;
        } while (iB < B0_size);
    } while (iB < B0_size);
```
Ignoring Sparsity Is Throwing Away Performance

Sparse Matrix Vector Multiplication (SpMV)

8K x 8K double precision matrix in CSR
Ignoring Sparsity Is Throwing Away Performance

Sparse Matrix Matrix Multiplication (SpMV)

4K x 4K double precision matrix in CSR
Sparse Problems Are Everywhere

Dense Array programs

SpMM

SpMV

Density

0% 10% 50%

[Hegde, et.al., MICRO 2019]
Sparse Problems Are Everywhere

Density

0% 10% Sparse Neural Networks 50% 100%

[Dense Array programs]

SpMV

[Hegde, et.al., MICRO 2019]
Sparse Problems Are Everywhere

Sparse Neural Networks

Density

Density Log scale

Internet & Social Media
Recommendation Systems
Circuit Simulation
Computational Chemistry
Finite Element Methods
Electromagnetics Proteins
Fluid Dynamics
Problems in Statistics

Dense Array programs

SpMV
SpMM

[Hegde, et al., MICRO 2019]
• $X \times Y \times Z \times (\text{time})$
• $8K \times 8K \times 8k$
• But only 300,000 points
• Data density $0.00005859\%$
Example: Sparsity In Lidar Data

- $X \times Y \times Z \times \text{(time)}$
- $8K \times 8K \times 8k$
- But only 300,000 points
- Data density $0.00005859\%$
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \quad a = Bc \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^T c \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^T c + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]
\[ A = BCd \quad A = B^T \quad a = B^T Bc \]
\[ a = b + c \quad A = B \quad K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k \]
\[ A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} c_j \]
\[ A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^Tc \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^Tc + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]
\[ A = BCD \quad A = B^T \quad a = B^TBc \]
\[ a = b + c \quad A = B \quad K = A^TCA \]

\[ A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj} \]
\[ A_{kj} = \sum_{il} B_{ikl}C_{lj}D_{ij} \]
\[ A_{lj} = \sum_{ik} B_{ikl}C_{lj}D_{kj} \]
\[ A_{ij} = \sum_{ik} B_{ikl}c_k \]
\[ A_{ijk} = \sum_{l} B_{ikl}C_{lj} \quad A_{ik} = \sum_{l} B_{ikl}c_j \]
\[ A_{jk} = \sum_{i} B_{i}c_i \quad A_{ijl} = \sum_{k} B_{ikl}C_{kj} \]
\[ A_{jkl} = \sum_{i} B_{i}c_i \]
\[ C = \sum_{ijkl} M_{ij}P_{jk}M_{lk}P_{il} \]
\[ a = \sum_{ijklmnop} M_{ij}P_{jk}M_{kl}P_{lm}M_{nm}P_{no}M_{po}P_{ip} \]

Linear Algebra
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a = Bc \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^T c \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^T c + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]
\[ A = BCD \quad A = B^T \quad a = B^T Bc \]
\[ a = b + c \quad A = B \quad K = A^T CA \]

\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}
\]
\[
A_{ij} = \sum_{ik} B_{ikl} C_{lj} D_{kj} \quad A_{ij} = \sum_{ik} B_{ijk} c_k
\]
\[
A_{ijk} = \sum_{il} B_{ikl} C_{lj} \quad A_{ik} = \sum_{ij} B_{ijk} c_j
\]
\[
A_{jk} = \sum_{il} B_{ikl} C_{lj} \quad A_{ijl} = \sum_{ik} B_{ikl} C_{kj}
\]

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

Data analytics
(tensor factorization)
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]  
\[ a = Bc \]

\[ a = Bc + b \]  
\[ A = B + C \]  
\[ a = \alpha Bc + \beta a \]

\[ a = B^T c \]  
\[ A = \alpha B \]  
\[ a = B(c + d) \]

\[ a = B^T c + d \]  
\[ A = B + C + D \]  
\[ A = BC \]

\[ a = B \odot C \]  
\[ a = b \odot c A = 0 \]  
\[ A = B \odot (CD) \]

\[ A = BCd \]  
\[ A = B^T \]  
\[ a = B^T Bc \]

\[ a = b + c \]  
\[ A = B \]  
\[ K = A^T C A \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]

\[ A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ A_{ij} = \sum_{ik} B_{ijk} c_k \]

\[ A_{ij} = \sum_{ik} B_{ikl} C_{lj} \]
\[ A_{ik} = \sum_{ik} B_{ijk} c_j \]

\[ A_{jk} = \sum_{ik} B_{ijk} c_i \]
\[ A_{ijl} = \sum_{ik} B_{ikl} C_{kj} \]

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

Quantum Chromodynamics
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]

\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]

\[ a = B^Tc \quad A = \alpha B \quad a = B(c + d) \]

\[ a = B^Tc + d \quad A = B + C + D \quad A = BC \]

\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]

\[ A = BCd \quad A = B^T \quad a = B^T Bc \]

\[ a = b + c \quad A = B \quad K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]

\[ A_{ij} = \sum_{ik} B_{ikl} C_{lj} D_{kj} \quad A_{ij} = \sum_{ik} B_{ikl} c_k \]

\[ A_{ijk} = \sum_{i} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} C_j \]

\[ A_{jk} = \sum_{i} B_{ijk} C_i \quad A_{ijkl} = \sum_{k} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \]

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

\[ \tau = \sum_{i} \left( \sum_{j} z_j \theta_{ij} \right) \left( \sum_{k} z_k \theta_{ik} \right) \]
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[
\begin{align*}
\text{CSparse} & \quad a = Bc + a \\
\text{Eigen (SpMV)} & \quad a = Bc \\
\text{OSKI} & \quad a = \alpha Bc + \beta a \\
\text{PETSc} & \quad a = B^T c \\
& \quad a = B^T (c + d) \\
& \quad A = B \odot (CD) \\
& \quad A = BCD \\
& \quad A = B^T \quad a = B^T Bc \\
& \quad a = b + c \\
A_{ij} &= \sum_{kl} B_{ikl} C_{lj} D_{kj} \\
A_{kj} &= \sum_{il} B_{ikl} C_{lj} D_{ij} \\
A_{li} &= \sum_{ik} B_{ikl} C_{ij} D_{kj} \\
A_{ij} &= \sum_{ik} B_{ikl} C_{ij} c_k \\
A_{ijk} &= \sum_{l} B_{ikl} C_{lj} c_j \\
A_{ik} &= \sum_{j} B_{ijk} + c_j c_i \\
A_{ij} &= \sum_{k} B_{ikl} C_{kj} \\
C &= \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{li} \\
& \quad \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip}
\end{align*}
\]
Sparsity Is Currently Addressed One-Problem-At-A-Time

CSparse
\[ a = Bc + a \]

Eigen (SpMV)
\[ a = Bc \]

OSKI
OSKI has 282 specialized variants of this expression

PETSc
\[ a = B^Tc \]

A = B + C
\[ A = \alpha B \]
\[ a = B(c + d) \]

A = B ⊕ C
\[ a = b ⊕ c \quad A = 0 \]
\[ A = B ⊕ (CD) \]

A = BCd
\[ A = B^T \]
\[ a = B^T Bc \]

A = B
\[ a = b + c \]
\[ K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kij} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]

\[ A_{lij} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ A_{ij} = \sum_{k} B_{ijk} c_k \]

\[ A_{ijk} = \sum_{i} B_{ikl} C_{lj} \]
\[ A_{ik} = \sum_{k} B_{ijk} c_j \]

\[ A_{ijk} = \sum_{i} B_{ijk} c_i \]
\[ A_{ij} = \sum_{k} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{ik} P_{il} \]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \quad \Rightarrow \quad a = Bc \]

\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]

\[ a = B^T c \quad A = \alpha B \quad a = B(c + d) \]

\[ a = B^T c + d \quad A = B + C + D \quad A = BC \]

\[ A = B \odot C \quad a = b \odot c A = 0 \quad A = B \odot (CD) \]

\[ A = BCd \quad A = B^T \quad a = B^T Bc \]

\[ a = b + c \quad A = B \quad K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]

\[ A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k \]

\[ A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{k} B_{ijk} c_j \]

\[ A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \quad \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij}) \left( \sum_{k} z_k \theta_{ik} \right) \]

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

Dense Matrix

CSR    DCSR    BCSR

COO    ELLPACK    CSB

Blocked COO    CSC

DIA    Blocked DIA    DCSC
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[
\begin{align*}
  a &= Bc + a \\
  a &= Bc + b \\
  A &= B + C \\
  a &= \alpha Bc + \beta a \\
  a &= B^T c \\
  A &= \alpha B \\
  a &= B(c + d) \\
  a &= B^T c + d \\
  A &= B + C + D \\
  A &= BC \\
  A &= B \odot C \\
  a &= b \odot c A = 0 \\
  A &= B \odot (CD) \\
  A &= BCD \\
  A &= B^T \\
  a &= B^T Bc \\
  a &= b + c \\
  A &= B \\
  K &= A^T CA
\end{align*}
\]

\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \\
A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \\
A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \\
A_{ij} = \sum_{k} B_{ijk} c_k
\]

\[
A_{ijk} = \sum_{l} B_{ikl} C_{lj} \\
A_{ik} = \sum_{j} B_{ijk} c_j \\
A_{jk} = \sum_{i} B_{ijk} c_i \\
A_{ijl} = \sum_{k} B_{ikl} C_{kj}
\]

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

Dense Matrix

<table>
<thead>
<tr>
<th>CSR</th>
<th>DCSR</th>
<th>BCSR</th>
<th>Thermal Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COO</td>
<td>ELLPACK</td>
<td>CSB</td>
<td>Blocked COO</td>
</tr>
<tr>
<td>DIA</td>
<td>Blocked DIA</td>
<td>DCSC</td>
<td></td>
</tr>
</tbody>
</table>
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]
\[ a = Bc + b \]
\[ A = B + C \]
\[ a = \alpha Bc + \beta a \]
\[ a = B^T c \]
\[ A = \alpha B \]
\[ a = B(c + d) \]
\[ a = B^T c + d \]
\[ A = B + C + D \]
\[ A = BC \]
\[ A = B \odot C \]
\[ a = b \odot c \]
\[ A = B \odot (CD) \]
\[ A = BCd \]
\[ A = B^T \]
\[ a = B^T Bc \]
\[ A = B + c \]
\[ a = B \]
\[ K = A^T CA \]

\[ A_{ij} = \sum_{k} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kj} = \sum_{i} B_{ikl} C_{lj} D_{ij} \]
\[ A_{lj} = \sum_{ik} B_{ikl} C_{lj} D_{kj} \]
\[ A_{ij} = \sum_{k} B_{ijk} c_k \]
\[ A_{ik} = \sum_{j} B_{ijk} c_j \]
\[ A_{ij} = \sum_{k} B_{ijk} c_i \]
\[ A_{ij} = \sum_{k} B_{ikl} C_{kj} \]
\[ A_{ij} = \sum_{k} B_{ijkl} c_i \]

\[ \tau = \sum_{i} z_i \left( \sum_{j} z_j \theta_{ij} \right) \left( \sum_{k} z_k \theta_{ik} \right) \]

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

Dense Matrix

CSR  DCSR  BCSR  Web matrix [BG 2008]
COO  ELLPACK  CSB
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]
\[ a = Bc + b \]
\[ A = B + C \]
\[ a = \alpha Bc + \beta a \]
\[ a = B^T c \]
\[ A = \alpha B \]
\[ a = B(c + d) \]
\[ a = B^T c + d \]
\[ A = B + C + D \]
\[ A = BC \]
\[ A = B \odot C \]
\[ a = b \odot c \]
\[ A = 0 \]
\[ A = B \odot (CD) \]
\[ A = BCD \]
\[ A = B^T \]
\[ a = B^T Bc \]
\[ a = b + c \]
\[ A = B \]
\[ K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ A_{ij} = \sum_{j} B_{ij} C_{j} \]
\[ A_{ik} = \sum_{k} B_{ijk} c_k \]
\[ A_{jk} = \sum_{i} B_{ijk} c_i \]
\[ A_{ij} = \sum_{j} B_{ij} c_j \]
\[ A_{ij} = \sum_{k} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \]
\[ \tau = \sum_{i} z_i \left( \sum_{j} z_j \theta_{ij} \right) \left( \sum_{k} z_k \theta_{ik} \right) \]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]

Dense Matrix

CSR  DCSR  BCSR  Finite Elements Method, Block-Sparse NN Weights [GRK 2017]

COO  ELLPACK  CSB

Blocked COO  CSC

DIA  Blocked DIA  DCSC

Blocked COO

Blocked DIA

DCSC
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[
a = Bc + a
\]

\[
a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a
\]

\[
a = BTc \quad A = \alpha B \quad a = B(c + d)
\]

\[
a = BTc + d \quad A = B + C + D \quad A = BC
\]

\[
A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD)
\]

\[
A = BCD \quad A = BT \quad a = BT Bc
\]

\[
a = b + c \quad A = B \quad K = ATCA
\]

\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}
\]

\[
A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k
\]

\[
A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{k} B_{ijk} c_j
\]

\[
A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj}
\]

\[
C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \quad a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip}
\]

Dense Matrix

CSR   DCSR   BCSR
COO   ELLPACK   CSB   Data Analytics

Blocked COO   CSC
DIA   Blocked DIA   DCSC
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]

\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]

\[ a = B^Tc \quad A = \alpha B \quad a = B(c + d) \]

\[ a = B^Tc + d \quad A = B + C + D \quad A = BC \]

\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]

\[ A = BCd \quad A = B^T \quad a = B^T Bc \]

\[ a = b + c \quad A = B \quad K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]

\[ A_{ij} = \sum_{ik} B_{ikl} C_{lj} D_{kj} \quad A_{ij} = \sum_{ik} B_{ikl} C_{kj} \]

\[ A_{ijk} = \sum_{kl} B_{ikl} C_{lj} \quad A_{ik} = \sum_{jk} B_{ijk} c_j \]

\[ A_{ilk} = \sum_{ij} B_{ilk} c_i \quad A_{ijl} = \sum_{ik} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \]

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

\[ \tau = \sum_i z_i \left( \sum_j z_j \theta_{ij} \right) \left( \sum_k z_k \theta_{ik} \right) \]

Dense Matrix
- CSR
- DCSR
- BCSR
- COO
- ELLPACK
- CSB
- Mesh Simulations on GPUs [BG 2009]
- Blocked COO
- CSC
- DIA
- Blocked DIA
- DCSC
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \quad a = Bc \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^T c \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^T c + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = B \odot (CD) \]
\[ A = BCd \quad A = B^T \quad a = B^T Bc \]
\[ a = b + c \quad A = B \quad K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k \]
\[ A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} c_j \]
\[ A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj} \]

\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{kl} P_{il} \quad \tau = \sum_{i} z_i \left( \sum_{j} z_j \theta_{ij} \right) \left( \sum_{k} z_k \theta_{ik} \right) \]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]

Dense Matrix

\[ \text{CSR} \quad \text{DCSR} \quad \text{BCSR} \]
\[ \text{COO} \quad \text{ELLPACK} \quad \text{CSB} \]
\[ \text{Blocked COO} \quad \text{CSC} \quad \text{DIA} \quad \text{Blocked DIA} \quad \text{DCSC} \]

Convolutions, Image Processing
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \]
\[ a = Bc + b \]
\[ A = B + C \]
\[ a = \alpha Bc + \beta a \]
\[ a = B^T c \]
\[ A = \alpha B \]
\[ a = B(c + d) \]
\[ a = B^T c + d \]
\[ A = B + C + D \]
\[ A = BC \]
\[ A = B \circ C \]
\[ a = b \circ c A = 0 \]
\[ A = B \circ (CD) \]
\[ A = BCd \]
\[ A = B^T \]
\[ a = B^T Bc \]
\[ a = b + c \]
\[ A = B \]
\[ K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{ij} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{kij} = \sum_{ikl} B_{ikl} C_{lj} D_{ij} \]
\[ A_{ik} = \sum_{ij} B_{ikl} C_{lj} D_{ij} \]
\[ A_{ijl} = \sum_{k} B_{ikl} C_{lj} D_{ij} \]

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

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK  CSB
Blocked COO  CSC
DIA  Blocked DIA  DCSC  Eulerian Simulations
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[
\begin{align*}
  a &= Bc + a \\
  a &= Bc + b \\
  A &= B + C \\
  a &= \alpha Bc + \beta a \\
  a &= B^T c \\
  A &= \alpha B \\
  a &= B(c + d) \\
  a &= B^T c + d \\
  A &= B + C + D \\
  A &= BC \\
  a &= b \odot c A = 0 \\
  A &= B \odot (CD) \\
  A &= BCD \\
  A &= B^T \\
  a &= B^T Bc \\
  a &= b + c \\
  A &= B \\
  K &= A^T CA \\
  A_{ij} &= \sum_{kl} B_{ikl} C_{lj} D_{kj} \\
  A_{kl} &= \sum_{li} B_{ikl} C_{lj} D_{kj} \\
  A_{ij} &= \sum_{il} B_{ikl} C_{lj} D_{kj} \\
  A_{ij} &= \sum_{ik} B_{ikl} C_{lj} D_{kj} \\
  A_{ij} &= \sum_{ik} B_{ikl} C_{lj} D_{kj} \\
  A_{ij} &= \sum_{ik} B_{ikl} C_{lj} D_{kj} \\
  A_{ij} &= \sum_{ik} B_{ikl} C_{lj} D_{kj} \\
  A_{ij} &= \sum_{ik} B_{ikl} C_{lj} D_{kj} \\
  \tau &= \sum_i z_i \left( \sum_j z_j \theta_{ij} \right) \left( \sum_k z_k \theta_{ik} \right) \\
  C &= \sum_{ijkl} M_{ij} P_{jk} M_{kl} P_{il} \\
  a &= \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \\
  &\times
\end{align*}
\]

Dense Matrix

- CSR
- DCSR
- BCSR
- COO
- ELLPACK
- CSB
- Blocked COO
- CSC
- DIA
- Blocked DIA
- DCSC

Sparse vector
Hash Maps
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \quad a = Bc \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^T c \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^T c + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]
\[ A = BCd \quad A = B^T \quad a = B^T Bc \]
\[ a = b + c \quad A = B \quad K = A^T CA \]
\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \\
A_{ij} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k \\
A_{ijk} = \sum_{i} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} c_j \\
A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj} \\
C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \tau = \sum_{i} \left( \sum_{j} z_j \theta_{ij} \right) \left( \sum_{k} z_k \theta_{ik} \right) \\
a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \\
\]

Dense Matrix
CSR  DCSR  BCSR
COO  ELLPACK  CSB
Blocked COO  CSC
DIA  Blocked DIA  DCSC
Sparse vector  Hash Maps
Coordinates
CSF  Dense Tensors
Blocked Tensors
Sparsity Is Currently Addressed One-Problem-At-A-Time

\[ a = Bc + a \quad a = Bc \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^T c \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^T c + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c A = 0 \quad A = B \odot (CD) \]
\[ A = BCd \quad A = B^T \quad a = B^T Bc \]
\[ A = b + c \quad A = B \quad K = A^T CA \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{ij} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ A_{ij} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ A_{ij} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ A_{ij} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \]
\[ a = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{mn} P_{no} M_{po} P_{ip} \]

Sparse vector
Hash Maps

Coordinates

Dense Matrix
CSR DCSR BCSR
COO ELLPACK CSB
Blocked COO CSC
DIA Blocked DIA DCSC
Sparse vector
Hash Maps

Coordinates

Dense Tensors
Blocked Tensors

CPU
GPUs TPUs
FPGA
Sparse Tensor Hardware
Cloud Computers
Supercomputers
Sparse Tensor Compiler

The Sparse Tensor Compiler
Expression Language

\[ A = Bc + a \quad a = Bc \]
\[ A = B \odot C \quad A = B + C \quad a = aBc + a \]
\[ A = BCd \quad a = b \odot c \quad A = B \odot (CD) \]
\[ A_{ij} = \sum_{l} B_{ail} C_{lij} D_{lj} \quad A_{il} = \sum_{l} B_{a(l)lj} A_{lk} - \sum_{l} B_{a(l)lj} D_{lj} \]
\[ A_{jkl} = \sum_{l} B_{a(l)lj} C_{klj} \quad A_{lj} = \sum_{j} B_{a(l)lj} + D_{lj} \]
\[ C = \sum_{i,j,k,l} M_{ijk} P_{k} M_{l} P_{l} \tau = \sum_{i,j,k,l} z_{ij} z_{kl} (z_{ij} z_{kl}) \]
\[ a = \sum_{i,j,k,l} M_{ijk} P_{k} M_{l} P_{l} M_{m} M_{n} P_{m} P_{n} P_{p} \]
Sparse Tensor Compiler

Expression Language

\[ A = Bc + a \]
\[ A = B \odot C \]
\[ A = B + C \]
\[ A = BCd \]
\[ A = BC \]
\[ A = BC \]
\[ A = aBc \]
\[ A = BC \]
\[ A = BC \]
\[ A = aBc \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
\[ A = BC \]
Sparse Tensor Compiler

Expression Language

\[ A = Bc + a \quad a = Bc \]
\[ A = B \odot C \quad A = B + C \quad a = aBc + ba \]
\[ A = BCD \quad a = B \odot c \quad A = B \odot (CD) \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kl} \]
\[ A_{ik} = \sum_{l} B_{ikl} C_{lj} \]
\[ A_{kl} = \sum_{i} B_{ikl} C_{lj} + D_{ij} \]
\[ C = \sum_{ijkl} M_{ijkl} P_{ikl} P_{lj} \]
\[ \tau = \sum_{i} \left( \sum_{j} i \beta_{ij} \right) \left( \sum_{k} i \beta_{ik} \right) \]
\[ a = \sum_{ijkl} M_{ijkl} P_{ikl} P_{lj} P_{mn} P_{np} P_{pq} \]

Format Language

- Dense Matrix
- DCSR
- CSR
- BCSR
- COO
- CSF
- DIA
- ELLPACK
- CSB
- Blocked COO
- CSC
- DCSC
- Sparse vector
- Blocked DIA
- Dense Tensors
- Blocked Tensors

Schedule Language

- pos
- reorder
- precompute
- divide
- vectorize
- split
- parallelize

The Sparse Tensor Compiler
Sparse Tensor Compiler

Expression Language

\[ A = Bc + a \quad a = Bc \]
\[ A = B \odot C \quad A = B + C \quad a = aBC + \beta \]
\[ A = BCD \quad a = B \odot c \quad A = B \odot (CD) \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{l}D_{kj} \quad A = B^{T} \quad a = B^{T}C \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{l}D_{kj} \quad A_{ij} = \sum_{kl} B_{ikl} C_{l}D_{kj} + D_{ij} \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{l}D_{kj} \quad A_{ij} = \sum_{kl} B_{ikl} C_{l}D_{kj} + D_{ij} \]
\[ C = \sum_{ijkl} M_{ijkl} P_{k} M_{np} P_{p} \quad \tau = \sum_{ijkl} z_{ijkl} \theta_{ijkl} \sum_{ijkl} z_{ijkl} \theta_{ijkl} \]
\[ a = \sum_{ijkl} M_{ijkl} P_{k} M_{np} P_{p} \]

Format Language

Dense Matrix DCSR CSR BCSR
COO CSF DIA ELLPACK CSB
Hash Maps Blocked COO CSC
DCSC Sparse vector Blocked DIA
Dense Tensors Blocked Tensors

Schedule Language

pos reorder precompute divide vectorize split parallelize
Sparse Tensor Compiler

Expression Language

\[
A = Bc + a \\
A = B \odot C \\
A = B + C \\
A = aB \\
A = Bc + \beta a \\
A = BCd \\
a = b \odot c \\
A = B^Tc \\
A = B^T (CD) \\
A_{ij} = \sum_d B_{ia} B_{dj} \\
A_{a} = \sum_j B_{ia} c_j \\
A_{a} = \sum_d B_{ia} c_j \\
A_{ija} = \sum_d B_{ia} c_j D_{dj} \\
A_{ij} = \sum_d B_{ia} c_j + D_{ij} \\
C = \sum_{i,j,k} M_{ij} P_{jk} M_{kn} P_{nr} \\
\tau = \sum_i \left( \sum_j z_j^2 \right) \left( \sum_k^a z_k^2 \right) \\
a = \sum_{i,j,k} M_{ij} P_{jk} M_{kn} P_{nr} M_{pq} P_{qr} \\
\]

Format Language

- Dense Matrix
- DCSR
- CSR
- BCSR
- COO
- CSF
- DIA
- ELLPACK
- CSB
- Hash Maps
- Blocked COO
- CSC
- DCSC
- Sparse vector
- Blocked DIA
- Dense Tensors
- Blocked Tensors

Schedule Language

- pos
- reorder
- precompute
- vectorize
- split
- parallelize
Generated Sparse Code Performance Matches Hand-Optimized Libraries

\[ a = Bc + a \quad a = Bc \]

\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]

\[ a = B^Tc \quad A = \alpha B \quad a = B(c + d) \]

\[ a = B^Tc + d \quad A = B + C + D \quad A = BC \]

\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]

\[ A = BCd \quad A = B^T \quad a = B^TBc \]

\[ a = b + c \quad A = B \quad K = A^TCA \]

\[ A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj} \quad A_{kj} = \sum_{il} B_{ikl}C_{lj}D_{ij} \]

\[ A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj} \quad A_{ij} = \sum_{k} B_{ijk}c_k \]

\[ A_{ijk} = \sum_{l} B_{ikl}C_{lj} \quad A_{ik} = \sum_{j} B_{ijk}c_j \]

\[ A_{jk} = \sum_{i} B_{ijk}c_i \quad A_{ijl} = \sum_{k} B_{ikl}C_{kj} \]

\[ \tau = \sum_{i} z_i(\sum_{j} z_j\theta_{ij})(\sum_{k} z_k\theta_{ik}) \]

\[ C = \sum_{ijkl} M_{ij}P_{jk}M_{lk}P_{il} \]

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

Normalized time
Generated Sparse Code Performance Matches Hand-Optimized Libraries

\[ a = Bc + a \quad a = Bc \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^Tc \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^Tc + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD) \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{ij} = \sum_{klt} B_{iklt} C_{lj} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k \]
\[ A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} c_j \]
\[ A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{j} B_{ikl} C_{kj} \]
\[ \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij}) (\sum_{k} z_k \theta_{ik}) \]
\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{tl} \]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{mp} P_{po} P_{ip} \]
Generated Sparse Code Performance Matches Hand-Optimized Libraries

\[ a = Bc + a \]
\[ a = Bc + b \]
\[ A = B + C \]
\[ a = \alpha Bc + \beta a \]
\[ a = B^T c \]
\[ A = \alpha B \]
\[ a = B(c + d) \]
\[ a = B^T c + d \]
\[ A = B + C + D \]
\[ A = BC \]
\[ A = B \odot C \]
\[ a = b \odot c \]
\[ A = 0 \]
\[ A = B \odot (CD) \]
\[ A = BCd \]
\[ a = B^T Bc \]
\[ A = b + c \]
\[ A = B \]
\[ K = A^T CA \]

\[ A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \]
\[ A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij} \]
\[ A_{ij} = \sum_{kl} B_{ikl} C_{lij} D_{kj} \]
\[ A_{ij} = \sum_{il} B_{ikl} C_{lij} D_{ij} \]
\[ A_{ik} = \sum_{l} B_{ikl} C_{lj} \]
\[ A_{ik} = \sum_{l} B_{ikl} C_{lj} \]
\[ A_{ijl} = \sum_{k} B_{ikt} C_{kj} \]
\[ A_{ijk} = \sum_{l} B_{ikt} C_{lij} \]
\[ \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij}) (\sum_{k} z_k \theta_{ik}) \]
\[ C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il} \]
\[ a = \sum_{ijk\ldots} M_{ij} P_{jk} M_{kl} P_{lm} M_{mn} P_{no} M_{po} P_{ip} \]
Generated Sparse Code Performance Matches Hand-Optimized Libraries

\[ a = Bc + a \]
\[ a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a \]
\[ a = B^Tc \quad A = \alpha B \quad a = B(c + d) \]
\[ a = B^Tc + d \quad A = B + C + D \quad A = BC \]
\[ A = B \odot C \quad a = b \odot c \quad A = 0 \]
\[ A = BCD \quad A = B^T \quad a = B^T Bc \]
\[ a = b + c \quad A = B \quad K = A^TCA \]

\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{kj}
\]
\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k
\]
\[
A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} c_j
\]
\[
A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj}
\]
\[
\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij}) (\sum_{k} z_k \theta_{ik})
\]
\[
C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{il}
\]
\[ a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{nm} P_{no} M_{po} P_{ip} \]

<table>
<thead>
<tr>
<th></th>
<th>rma10</th>
<th>cop20k</th>
<th>scircuit</th>
<th>mac-econ</th>
<th>pwtk</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpMV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = Bc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDDMM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = B \odot (CD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Normalized time**

- taco
- MKL
- OSKI
- Eigen
- UBLAS
- Gmm++

2412x
24835x
59496x
73405x
22400x
Generated Sparse Code Performance Matches Hand-Optimized Libraries

\[
a = Bc + a
\]
\[
a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a
\]
\[
a = B^T c \quad A = \alpha B \quad a = B(c + d)
\]
\[
a = B^T c + d \quad A = B + C + D \quad A = BC
\]
\[
a = B \odot C \quad a = b \odot c \quad A = 0
\]
\[
a = BCd \quad A = B^T \quad a = B^T Bc
\]
\[
a = b + c \quad A = B \quad K = ATCA
\]
\[
A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}
\]
\[
A_{ij} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} C_{j}
\]
\[
A_{jk} = \sum_{i} B_{ijk} C_{i} \quad A_{ijl} = \sum_{j} B_{ikl} C_{kj}
\]
\[
A_{ij} = \sum_{i} z_i (\sum_{j} z_j \theta_{ij}) (\sum_{k} z_k \theta_{ik})
\]
\[
C = \sum_{ijkl} M_{ij} P_{jk} M_{lk} P_{li}
\]
\[
a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} M_{mn} P_{no} M_{po} P_{ip}
\]

Diagram:

- **SpMV**
  - \( a = Bc \)
  - \( A = B + C \)
  - \( A = \alpha Bc + \beta a \)
  - \( A = B^T c \)
  - \( A = B^T c + d \)
  - \( A = B + C + D \)
  - \( A = BC \)
  - \( A = b + c \)
  - \( A = B \)

- **SDDMM**
  - \( A = B \odot (CD) \)
  - \( A_{ij} = \sum_{k} B_{ijk} C_{kj} \)
  - \( A_{ik} = \sum_{j} B_{ijk} C_{j} \)
  - \( A_{jk} = \sum_{i} B_{ijk} C_{i} \)
  - \( A_{ijl} = \sum_{j} B_{ikl} C_{kj} \)

- **MTTKRP**
  - \( A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \)
  - \( A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \)

- **Normalized time**
  - \( 2412x \)
  - \( 24835x \)
  - \( 59496x \)
  - \( 73405x \)
  - \( 22400x \)
  - \( 84x \)
  - \( 319x \)
  - \( 76x \)
  - \( 20 \)
Generated Sparse Code Performance Matches Hand-Optimized Libraries

Sampled Dense-Dense Matrix Multiplication

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

- SDDMM
- taco
- Eigen
- UBLAS

Normalized time

<table>
<thead>
<tr>
<th>Library</th>
<th>Normalized time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rma10</td>
<td>2412x</td>
</tr>
<tr>
<td>cop20k</td>
<td>24835x</td>
</tr>
<tr>
<td>scircuit</td>
<td>59496x</td>
</tr>
<tr>
<td>mac-econ</td>
<td>73405x</td>
</tr>
<tr>
<td>pwtk</td>
<td>22400x</td>
</tr>
</tbody>
</table>
Generated Sparse Code Performance Matches Hand-Optimized Libraries

![Matrix Multiplication Diagram]

C

Sampled Dense-Dense Matrix Multiplication

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

SDDMM

<table>
<thead>
<tr>
<th>Library</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>rma10</td>
<td>2412x</td>
</tr>
<tr>
<td>cop20k</td>
<td>24835x</td>
</tr>
<tr>
<td>scircuit</td>
<td>59496x</td>
</tr>
<tr>
<td>mac-econ</td>
<td>73405x</td>
</tr>
<tr>
<td>pwtk</td>
<td>22400x</td>
</tr>
</tbody>
</table>

Normalized time
Generated Sparse Code Performance Matches Hand-Optimized Libraries

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

Sampled Dense-Dense Matrix Multiplication

<table>
<thead>
<tr>
<th>Library</th>
<th>Normalized Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rma10</td>
<td>2412x</td>
</tr>
<tr>
<td>cop20k</td>
<td>24835x</td>
</tr>
<tr>
<td>scircuit</td>
<td>59496x</td>
</tr>
<tr>
<td>mac-econ</td>
<td>73405x</td>
</tr>
<tr>
<td>pwtk</td>
<td>22400x</td>
</tr>
</tbody>
</table>
Generated Sparse Code Performance Matches Hand-Optimized Libraries

64 inner product

Sampled Dense-Dense Matrix Multiplication

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

<table>
<thead>
<tr>
<th>Library</th>
<th>Normalized time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rma10</td>
<td>2412x</td>
</tr>
<tr>
<td>cop20k</td>
<td>24835x</td>
</tr>
<tr>
<td>scircuit</td>
<td>59496x</td>
</tr>
<tr>
<td>mac-econ</td>
<td>73405x</td>
</tr>
<tr>
<td>pwtk</td>
<td>22400x</td>
</tr>
</tbody>
</table>

Normalized time
Generated Sparse Code Performance Matches Hand-Optimized Libraries

Sampled Dense-Dense Matrix Multiplication

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

- rma10
- cop20k
- scircuit
- mac-econ
- pwtk

<table>
<thead>
<tr>
<th>Library</th>
<th>Normalized time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rma10</td>
<td>2412x</td>
</tr>
<tr>
<td>cop20k</td>
<td>24835x</td>
</tr>
<tr>
<td>scircuit</td>
<td>59496x</td>
</tr>
<tr>
<td>mac-econ</td>
<td>73405x</td>
</tr>
<tr>
<td>pwtk</td>
<td>22400x</td>
</tr>
</tbody>
</table>
Generated Sparse Code Performance Matches Hand-Optimized Libraries

Sampled Dense-Dense Matrix Multiplication

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

<table>
<thead>
<tr>
<th>Library</th>
<th>Normalized Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rma10</td>
<td>2412x</td>
</tr>
<tr>
<td>cop20k</td>
<td>24835x</td>
</tr>
<tr>
<td>scircuit</td>
<td>59496x</td>
</tr>
<tr>
<td>mac-econ</td>
<td>73405x</td>
</tr>
<tr>
<td>pwtk</td>
<td>22400x</td>
</tr>
</tbody>
</table>

Normalized time
Generated Sparse Code Performance Matches Hand-Optimized Libraries

Sampled Dense-Dense Matrix Multiplication

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

Normalized time
Generated Sparse Code Performance Matches Hand-Optimized Libraries

- **B**
- **C**
- **D**

64 inner product
10 inner product

This dot product need not be computed

**SDDMM**

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

**Sampled Dense-Dense Matrix Multiplication**

- **rma10**
- **cop20k**
- **scircuit**
- **mac-econ**
- **pwtk**

**Normalized time**

- 2412x
- 24835x
- 59496x
- 73405x
- 22400x
**Generated Sparse Code Performance Matches Hand-Optimized Libraries**

![Diagram showing matrix multiplication](image)

**Sampled Dense-Dense Matrix Multiplication**

We will generate fused operations

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

**SDDMM**

- rma10
- cop20k
- scircuit
- mac-econ
- pwtk

Normalized time:

- rma10: 2412x
- cop20k: 24835x
- scircuit: 59496x
- mac-econ: 73405x
- pwtk: 22400x
Sparsity Beyond Zero Fill Values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Compressed Level Format

pos | 0 | 5 | 13 | 16 | 23
---|---|---|---|---|---
coord | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
vals | 1 | 1 | 1 | 5 | 5 | 6 | 6 | 6 | 6 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 1 | 1 | 1 | 8 | 8 | 8 | 2 | 2

0 | 1 | 1 | 1 | 0 | 0 | 0 | 5 | 5
1 | 6 | 6 | 6 | 6 | 1 | 1 | 1 | 1
2 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0
3 | 1 | 1 | 1 | 8 | 8 | 8 | 2 | 2
Sparsity Beyond Zero Fill Values

Compressed Level Format

pos  
0 5 13 16 23

coord  
0 1 2 6 7 0 1 2 3 4 5 6 7 0 1 2 0 1 2 3 4 5 7 7

vals  
1 1 1 5 5 6 6 6 6 1 1 1 1 3 3 3 1 1 1 8 8 8 2 2

Compressed Level Format with a Fill Value

pos  
0 5 9 17 21

coord  
3 4 5 6 7 0 1 2 3 0 1 2 3 4 5 6 7 3 4 5 6 7

vals  
0 0 0 5 5 6 6 6 6 3 3 3 0 0 0 0 0 8 8 8 2 2

Fill  
1

0 1 2 3 4 5 6 7
0 1 1 0 0 0 5 5
1 6 6 6 1 1 1 1
2 3 3 0 0 0 0 0
3 1 1 8 8 8 2 2
Sparsity Beyond Zero Fill Values

Compressed Level Format

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>5</th>
<th>13</th>
<th>16</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>vals</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Compressed Level Format with a Fill Value

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>5</th>
<th>9</th>
<th>17</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>vals</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Run Length Encoding (RLE) Level Format

- Extension of the Compressed Format
- Last value is the Fill Value
### Sparsity Beyond Zero Fill Values

**Compressed Level Format**

<table>
<thead>
<tr>
<th>pos</th>
<th>0 5 13 16 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>0 1 2 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0</td>
</tr>
<tr>
<td>vals</td>
<td>1 1 1 5 5 6 6 6 6 1 1 1 1 3 3 3 1 1 1 8 8 8 8 2 2</td>
</tr>
</tbody>
</table>

**Compressed Level Format with a Fill Value**

<table>
<thead>
<tr>
<th>pos</th>
<th>0 5 9 17 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>3 4 5 6 7 0 1 2 3 0 1 2 3 4 5 6 7 3 4 5 6 7</td>
</tr>
<tr>
<td>vals</td>
<td>0 0 0 5 5 6 6 6 6 3 3 3 0 0 0 0 0 0 8 8 8 2 2</td>
</tr>
<tr>
<td>Fill</td>
<td>1</td>
</tr>
</tbody>
</table>

**Run Length Encoding (RLE) Level Format**

<table>
<thead>
<tr>
<th>pos</th>
<th>0 3 5 7 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>0 3 6 0 4 0 3 0 3 6</td>
</tr>
<tr>
<td>vals</td>
<td>1 0 5 6 1 3 0 1 8 2</td>
</tr>
</tbody>
</table>

- Extension of the Compressed Format
- Last value is the Fill Value
# Sparsity Beyond Zero Fill Values

## Compressed Level Format

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>5</th>
<th>13</th>
<th>16</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>vals</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

## Compressed Level Format with a Fill Value

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>5</th>
<th>9</th>
<th>17</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>vals</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Fill

## Run Length Encoding (RLE) Level Format

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>vals</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

- Extension of the Compressed Format
- Last value is the Fill Value

## Unifying Sparsity and Lossless Compression
Performance Advantage In Lossless Compression

Edge Detection of MRI Image

Alpha Blending of Two Videos
Performance Advantage In Lossless Compression

Edge Detection of MRI Image

Alpha Blending of Two Videos

Size Reduction over Dense (in bytes)

Geomean Size Reduction over Dense (in bytes)
Performance Advantage In Lossless Compression

Edge Detection of MRI Image

Alpha Blending of Two Videos
Performance Advantage In Lossless Compression

Edge Detection of MRI Image

Alpha Blending of Two Videos
Example: Dot Product Of Two Vectors

$$c = \sum_{i} a_i \cdot b_i$$

\[
\begin{array}{cccccccccc}
& 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
\text{c} & 3.1 & 2.4 & 4.2 & 8.6 & 5.9 & 3.2 & 0.7 & 4.4 & 2.9 \\
& 8.6 & 1.9 & 9.4 & 5.0 & 5.4 & 1.2 & 5.2 & 3.9 & 8.0 \\
\end{array}
\]

\(a\) is a length \(n\) vector \hspace{1cm} \(b\) is a length \(n\) vector
Dense Arrays Store Every Value They Represent

\[ c = a \cdot b \]

\( a \) is dense

\( b \) is dense
Dense Arrays Store Every Value They Represent

\[ c = \begin{bmatrix} 3.1 & 2.4 & 4.2 & 8.6 & 5.9 & 3.2 & 0.7 & 4.4 & 2.9 \end{bmatrix} \]

\[ a \] is dense

\[ b \] is dense
Dense Arrays Store Every Value They Represent

\[ c = \begin{bmatrix}
3.1 & 2.4 & 4.2 & 8.6 & 5.9 & 3.2 & 0.7 & 4.4 & 2.9
\end{bmatrix} \]

\[ a \text{ is dense} \]

\[ b \text{ is dense} \]

for \( i = 1:n \)

\[ c += a[i] \times b[i] \]
Sparse Arrays Interpret Stored Data Differently

\[
c = \begin{bmatrix} 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{bmatrix}
\cdot \begin{bmatrix} 8.6 & 1.9 & 9.4 & 5.0 & 5.4 & 1.2 & 5.2 & 3.9 & 8.0 \end{bmatrix}
\]

\(a\) is sparse

\(b\) is dense
Sparse Arrays Interpret Stored Data Differently

\[
c = \begin{bmatrix} 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{bmatrix}
\]

\[
a = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & 3.1 & \cdot & \cdot & 2.4 \end{bmatrix}
\]

\[ab = \begin{bmatrix} 8.6 & 1.9 & 9.4 & 5.0 & 5.4 & 1.2 & 5.2 & 3.9 & 8.0 \end{bmatrix}\]

\[b\text{ is dense}\]

\[a\text{ is sparse}\]
Sparse Arrays Interpret Stored Data Differently

```
while p < len(a.idx):
    i = a.idx[p]
    c += a.val[p] * b[i]
    p += 1
```
Merging Multiple Sparse Requires Coordination

\[ a \text{ is sparse} \quad b \text{ is sparse} \]
Merging Multiple Sparse Requires Coordination

\[ a \text{ is sparse} \quad b \text{ is sparse} \]
Merging Multiple Sparse Requires Coordination

\[
\begin{align*}
  a.\text{idx} & : 1 & 2 & 6 & 8 \\
  a.\text{val} & : 3.1 & 2.4 & 3.2 & 4.4 \\
  b.\text{idx} & : 2 & 4 & 6 & 7 & 9 \\
  b.\text{val} & : 1.9 & 5.0 & 1.2 & 5.2 & 8.0 \\
\end{align*}
\]

\[
c = 3.1 \cdot 2.4 \cdot 0 \cdot 0 \cdot 0 \cdot 3.2 \cdot 0 \cdot 4.4 \cdot 0 \cdot 0 \cdot 0 \cdot 1.9 \cdot 5.0 \cdot 0 \cdot 1.2 \cdot 5.2 \cdot 0 \cdot 0
\]

while \( p < \text{len}(a.\text{idx}) \) && \( q < \text{len}(b.\text{idx}) \)

\[
\begin{align*}
i_a & = a.\text{idx}[p] \\
i_b & = b.\text{idx}[q] \\
i & = \text{min}(i_a, i_b) \\
\text{if } i & = i_a \&\& i = i_b \\
c & += a.\text{val}[p] \cdot b.\text{val}[q] \\
p & += i_a == i \\
q & += i_b == i
\end{align*}
\]

\( a \) is sparse \hspace{1cm} \( b \) is sparse
Some Sparse Inputs Have Structure

\[ c = a \cdot b \]

<table>
<thead>
<tr>
<th>idx</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.idx</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>val</td>
<td>3.1</td>
<td>2.4</td>
<td>3.2</td>
<td>4.4</td>
</tr>
</tbody>
</table>

\[ b.start: 4 \quad b.stop: 7 \]

| val | 5.0 | 5.4 | 1.2 | 5.2 |

\[ a \text{ is sparse} \quad b \text{ is blocked} \]
Some Sparse Inputs Have Structure

\[ a.\text{idx: } 1 \quad 2 \quad 3 \quad 4 \]
\[ a.\text{val: } 3.1 \quad 2.4 \quad 3.2 \quad 4.4 \]
\[ b.\text{start: } 4 \quad b.\text{stop: } 7 \]
\[ b.\text{val: } 5.0 \quad 5.4 \quad 1.2 \quad 5.2 \]

\[ c = \begin{bmatrix} 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{bmatrix} \]

\[ b \text{ is blocked} \]

\[ a \text{ is sparse} \]
Some Sparse Inputs Have Structure

\[ c = \begin{bmatrix} 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 5.0 & 5.4 & 1.2 & 5.2 & 0 & 0 \end{bmatrix} \]

\[ \text{while } p < \text{len}(a.\text{idx}) \&\& q < \text{len}(b.\text{idx}) \]
\[ i_a = a.\text{idx}[p] \]
\[ i_b = b.\text{idx}[q] \]
\[ i = \text{min}(i_a, i_b) \]
\[ \text{if } i == i_a \&\& i == i_b \]
\[ c += a.\text{val}[p] \times b.\text{val}[q] \]
\[ p += i_a == i \]
\[ q += i_b == i \]

\( a \) is sparse \hspace{2cm} \( b \) is blocked
Some Sparse Inputs Have Structure

\[ c = \begin{bmatrix} 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 5.0 & 5.4 & 1.2 & 5.2 & 0 & 0 \end{bmatrix} \]

- **a** is sparse
- **b** is blocked

\[
\text{while } p < \text{len}(a.\text{idx}) \&\& q < b.\text{stop} - b.\text{start} \\
\quad i_a = a.\text{idx}[p] \\
\quad i_b = b.\text{start} + q \\
\quad i = \text{min}(i_a, i_b) \\
\quad \text{if } i == i_a \&\& i == i_b \\
\quad \quad c += a.\text{val}[p] \times b.\text{val}[q] \\
\quad p += i_a == i \\
\quad q += i_b == i
\]
We Can Use Structure

\[ a \text{ is sparse} \quad b \text{ is blocked} \]
We Can Use Structure

We can use structure to efficiently perform operations on sparse and blocked data structures. For example, consider two data structures $a$ and $b$ with the following properties:

- $a$ is sparse, meaning it contains only a subset of its possible elements.
- $b$ is blocked, meaning it is stored in a contiguous block of memory.

We can use binary search on sparse structures to find elements efficiently. For instance, we can perform a binary search on $a$ to find an element.

Here is an example of how we can use structure to combine $a$ and $b$:

- $a$ has indices $\text{idx} = [1, 2, 6, 8]$ and values $\text{val} = [3.1, 2.4, 3.2, 4.4]$.
- $b$ has start index $\text{start} = 4$, stop index $\text{stop} = 7$, and values $\text{val} = [5.0, 5.4, 1.2, 5.2]$.

To combine $a$ and $b$, we can use the following operation:

$$ c = a \cdot b $$

where $c$ represents the combined structure, and the operation is performed on corresponding elements of $a$ and $b$.

This approach leverages the structure of each data type to optimize operations and reduce computational overhead.
We Can Use Structure

$$c = \begin{bmatrix} 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 5.0 & 5.4 & 1.2 & 5.2 & 0 & 0 \end{bmatrix}$$

**a** is sparse

**b** is blocked

Binary search is sparse

$$a = \begin{bmatrix} \text{idx:} & 1 & 2 & 6 & 8 \ \text{val:} & 3.1 & 2.4 & 3.2 & 4.4 \end{bmatrix}$$

$$b = \begin{bmatrix} \text{start:} & 4 \ \text{stop:} & 7 \ \text{val:} & 5.0 & 5.4 & 1.2 & 5.2 \end{bmatrix}$$
We Can Use Structure

\[ c = \begin{array}{cccccccc}
3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0
\end{array} \cdot \begin{array}{cccccccc}
0 & 0 & 0 & 5.0 & 5.4 & 1.2 & 5.2 & 0 & 0
\end{array} \]

\[ p = \text{binarysearch}(a.\text{idx}, b.\text{start}) \]

While \( p < \text{len}(a.\text{idx}) \):

\[ i = a.\text{idx}[p] \]

If \( i > b.\text{stop} \):

\[ \text{break} \]

\[ c += a.\text{val}[p] \times b.\text{val}[i - b.\text{start}] \]

\[ p += 1 \]

\( a \) is sparse

\( b \) is blocked

binary search

\( a.\text{idx} \): \[ 1 \ 2 \ 6 \ 8 \]

\( a.\text{val} \): \[ 3.1 \ 2.4 \ 3.2 \ 4.4 \]

\( b.\text{start} \): \[ 4 \]

\( b.\text{stop} \): \[ 7 \]

\( b.\text{val} \): \[ 5.0 \ 5.4 \ 1.2 \ 5.2 \]
**Algorithms For All Combinations?**

<table>
<thead>
<tr>
<th>Dense</th>
<th>Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
</tr>
</tbody>
</table>
**Algorithms For All Combinations?**

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
<td>Gather</td>
<td>?</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
<td>Merge</td>
<td>?</td>
</tr>
<tr>
<td>Block</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
### Algorithms For All Combinations?

<table>
<thead>
<tr>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
<th>Ragged</th>
<th>Run Length</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Block</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
## Algorithms For All Combinations?

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
<th>Ragged</th>
<th>Run Length</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
<td>Gather</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
<td>Merge</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

...
## Algorithms For All Combinations?

### TACO

<table>
<thead>
<tr>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
<th>Ragged</th>
<th>Run Length</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>For Loop</td>
<td>Gather</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Gather</td>
<td>Merge</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
<th>Ragged</th>
<th>Run Length</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
<td>Gather</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
<td>Merge</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

...
## Algorithms For All Combinations?

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
<th>Ragged</th>
<th>Run Length</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
<td>Gather</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
<td>Merge</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>


## Algorithms For All Combinations?

<table>
<thead>
<tr>
<th></th>
<th>Dense</th>
<th>Sparse</th>
<th>Blocked</th>
<th>Ragged</th>
<th>Run Length</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>For Loop</td>
<td>Gather</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Sparse</td>
<td>Gather</td>
<td>Merge</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

---


Looplet Language

• A general language to iterate over structured data
  • Iterating over complex structured data expressed using a language of a few primitives
    • Lookup
    • Run
    • Spike
    • Pipeline
    • Stepper
    • Jumper
    • Shift
    • Switch
**Looplet Language**

- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch
Looplet Language

- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Scalar</td>
<td>Run</td>
<td>Scalar</td>
<td>Scalar</td>
</tr>
<tr>
<td>0</td>
<td>4.2</td>
<td>0</td>
<td>8.6</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run</td>
<td></td>
<td>Run</td>
<td></td>
<td>Scalar</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>9.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Pipeline

Stepper

Spike

Run

Scalar
Looplet Language

• A general language to iterate over structured data
  • Iterating over complex structured data expressed using a language of a few primitives
    • Lookup
    • Run
    • Spike
    • Pipeline
    • Stepper
    • Jumper
    • Shift
    • Switch

• Code generation from the iteration protocols is simple and mechanical

```
Run
Pipeline
for i = 1:y.start-1
  visit(i, 0)
for i = y.start:y.stop
  visit(i, y.val[i + 1 - start])
for i = y.stop + 1:end
  visit(i, 0)
```
Looplet Language

• A general language to iterate over structured data
  • Iterating over complex structured data expressed using a language of a few primitives
    • Lookup
    • Run
    • Spike
    • Pipeline
    • Stepper
    • Jumper
    • Shift
    • Switch

• Code generation from the iteration protocols is simple and mechanical

• To coiterate, merge the individual iteration protocols
  • Use rewrite rules to simplify
Looplet Language Supports Many Types Of Structured Data

- Unifying what is currently done by multiple compilers
  - Hybrid “have-it-all” formats
  - Expanding into other types of structures

**Ragged Matrix**

- 3.5 2.5 8.6 0.4 0.8 8.9 4.0 2.3 9.8
- 2.7 0 0 0 0 0 0 0 0
- 7.0 1.8 0 0 0 0 0 0 0
- 0.9 0.6 4.1 7.3 9.0 8.9 8.9 0.9 1.6
- 5.2 4.6 4.3 5.0 9.8 3.6 2.7 0.4 0
- 5.0 0.5 0 0 0 0 0 0 0
- 7.2 2.9 0 0 0 0 0 0 0
- 0.7 3.2 2.5 2.3 4.7 8.2 8.9 8.7 3.9 7.0 8.1
- 2.0 6.8 0.9 1.1 3.7 5.0 6.5 4.0 2.6
- 0.9 5.1 5.9 7.4 0.1 5.5 0 0 0
- 7.8 9.9 4.1 1.9 1.4 3.3 3.4 8.3 4.1

**Run Length Matrix**

- 1 1 1 1 1 1 1 1 1
- 1 1 1 1 1 1 1 1 1
- 1 1 1 1 1 1 2 2 1 1
- 1 1 1 1 1 1 2 2 2 2 2 2 1
- 1 1 1 1 2 2 2 2 2 2 1
- 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
- 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
- 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
- 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
- 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2

**Symmetric Matrix**

- 0.0 9.4 6.0 9.6 6.0 5.5 5.9
- 9.4 9.3 6.0 5.1 4.4 0.3 1.9
- 6.0 6.0 9.6 8.6 2.1 8.8 0.3
- 6.0 6.0 9.6 8.6 2.1 8.8 0.3
- 9.6 5.1 8.6 9.3 4.9 4.5 4.1
- 3.3 7.6 9.1 7.4 3.3 7.6 9.1 7.4
- 6.0 4.4 2.1 4.9 7.1 7.2 3.9
- 2.1 4.0 4.9 2.7
- 5.5 0.3 8.8 4.5 7.2 0.4 4.9
- 2.3 4.7 2.0 8.9
- 5.9 1.9 0.3 4.1 3.9 4.9 2.3
- 3.9 6.6 4.2 7.9
- 6.1 6.1 7.0 3.3 2.1 2.3 3.9
- 0.7 4.1 1.4 3.7
- 4.6 6.2 2.3 7.6 4.0 4.7 6.6
- 4.1 6.3 5.0 3.2
- 3.2 3.8 7.5 9.1 4.9 2.0 4.2
- 1.4 5.0 5.8 5.1
- 3.3 0.3 7.1 7.4 2.7 8.9 7.9
- 3.7 3.2 5.1 3.4


D. Donenfeld, S. Chou, and S. Amarasinghe, “Unified Compilation for Lossless Compression and Sparse Computing”

Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But...inserting or deleting an element can be (asymptotically) slow
Dynamic Sparse Tensors

• All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  • Computing on them can be very fast
  • But…inserting or deleting an element can be (asymptotically) slow

1.63M to 255M Number of Non-Zeros Stored in the CSR Format
Dynamic Sparse Tensors

• All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  • Computing on them can be very fast
  • But...inserting or deleting an element can be (asymptotically) slow

1.63M to 255M Number of Non-Zeros Stored in the CSR Format

PageRank Computation

Insert a Single Element
Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But...inserting or deleting an element can be (asymptotically) slow
Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But...inserting or deleting an element can be (asymptotically) slow

- Many real world Applications are dynamic
Dynamic Sparse Tensors

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>J</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
</tbody>
</table>
Dynamic Sparse Tensors

• Need pointer-based, recursive data structures
Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration
Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

```
def list (e: elem nonempty n: list)
  seq = (e), n
)
def list_head (h: list)
```
Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

```
def list (
  e : elem nonempty
  n : list
  seq = (e), n
)

def list_head (
  h : list
)

def blist (
  e : elem[B] nonempty
  n : blist
  B : size in [0, 3]
  seq = (e), n
)

def blist_head (
  h : blist
)
```
Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration
Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration
Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration
Continuous Function  Spatial Database  3D Point Cloud  Computer Graphics
Programming On Continuous Data

Continuous Function  Spatial Database  3D Point Cloud  Computer Graphics
Programming On Continuous Domain Is Difficult!

1. Storing or Iterating over geometries are non-trivial
   ⇒ (Quadtree/Octree, Bounding Volume Hierarchy..)

2. **501 Lines of Code** in hand-written library (Box search query, C++)
Programming On Continuous Domain Is Difficult!

1. Core kernel(KPConv) can be expressed in a single math equation.

2. 2,330 Lines of Code in PyTorch and C.
Programming On Continuous Domain Is Difficult!

1. Core kernel (KPConv) can be expressed in a single math equation.
2. 2,330 Lines of Code in PyTorch and C.
Arrays Are

- Multi-dimensional
- Rectilinear
- Integer grid

Of points
Arrays Are

- Multi-dimensional
- Rectilinear
- Dense
- Integer grid

Of points
The Continuous Tensor Abstraction: Fresh Perspective On Tensor And Loops

A[2] // i=[0,1]
for i = 0:1

Existing Tensor Abstraction
<table>
<thead>
<tr>
<th>Continuous Tensor Abstraction</th>
<th>Existing Tensor Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-Numbered Index Access</td>
<td>For loop on continuous domain</td>
</tr>
</tbody>
</table>

```
A[2]
// i=[0,1]
for i = 0:1
```

```
A[3.1415]
// i = {x ∈ ℝ | 0 ≤ x ≤ 1}
for i = 0.0:1.0
```
Comparing To Existing Array Programming Model

Vector
(integer domain)

Existing tensor programming

```
# loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```
Comparing To Existing Array Programming Model

Pinpoint Coordinates on Continuous Domain

```
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end  # s = 44
```
Comparing To Existing Array Programming Model

**Continuous Tensor Abstraction**

Existing Tensor Abstraction

Continuous Tensor Abstraction

```java
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

```java
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i]
end # s = 44
```

Pinpoint Coordinates on Continuous Domain

Continuous Dot-Product
Comparing To Existing Array Programming Model

Existing Tensor Abstraction

Continuous Tensor Abstraction

Pinpoint Coordinates on Continuous Domain

Continuous Dot-Product

Interval Coordinates on Continuous Domain

#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44

#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i]
end # s = 44
Comparing To Existing Array Programming Model

### Existing Tensor Abstraction

```plaintext
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

### Continuous Tensor Abstraction

```
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i]
end # s = 44
```

---

### Pinpoint Coordinates on Continuous Domain

```
x[i] = 1 2 3 4
y[i] = 5 6 7 8
```

### Interval Coordinates on Continuous Domain

```
x[i] = [1, 2]
y[i] = [3, 4]
```

### Continuous Dot-Product

```
s = \int_{0}^{9.0} x_i \cdot y_i \cdot d(i)
```

- \( s = s + \int_{0}^{9.0} x_i \cdot y_i \cdot d(i) \)
Motivational Example: Radius Search In Gis

Radius Search: Get the number of points within the distance $R$. 
Motivational Example: Radius Search In GIS

```
count = 0
for dx=-1.7:1.7 # continuous
  for dy=-1.7:1.7 # continuous
    if dx*dx+dy*dy <= 1.7*1.7
      count += A[2.2+dx, 3.9+dy]
# count = 3
```
Motivational Example: Radius Search In Gis

```python
count = 0
for dx=-1.7:1.7  # continuous
    for dy=-1.7:1.7  # continuous
        if dx*dx+dy*dy <= 1.7*1.7
            count += A[2.2+dx, 3.9+dy]
# count = 3
```
Research Questions

RQ1. How can we **store infinitely many** coordinates in continuous tensor?

RQ2. How can we **iterate infinitely many** indices in continuous loop?

```python
# loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i] * d(i)
end # s = 2.4
```
All Continuous Tensors must satisfy a piecewise-constant property
All Continuous Tensors must satisfy a piecewise-constant property

\[ f(x) = \cos(x) \]
\[ g(x) = e^x \]

Piecewise-constant

Not Piecewise-constant
## Case Studies

<table>
<thead>
<tr>
<th>Applications</th>
<th>Baseline</th>
<th>Ours</th>
<th>LoC Saving</th>
<th>Perf Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius Search Query</td>
<td>501 lines</td>
<td>5 lines</td>
<td>100×</td>
<td>9.20×</td>
</tr>
<tr>
<td>Point Cloud Convolution</td>
<td>2,330 lines</td>
<td>16 lines</td>
<td>145×</td>
<td>0.23×</td>
</tr>
<tr>
<td>Trilinear Interpolation in NeRF</td>
<td>82 lines</td>
<td>9 lines</td>
<td>9×</td>
<td>1.69×</td>
</tr>
<tr>
<td>Genomic Interval Overlapping Query</td>
<td>206 lines</td>
<td>8 lines</td>
<td>26×</td>
<td>1.22×</td>
</tr>
</tbody>
</table>

**Code Fast**

**Run Fast**
Case Study: Neural Radiance Field
Case Study: Nerf

```
for t=0:T-1  # sampling on discrete timestep
    x = Ox + Dx*t  # O: ray origin, D: ray direction
    y = Oy + Dy*t
    z = Oz + Dz*t
```
Case Study: Nerf

\[ \int_0^1 \int_0^1 \text{Grid}[x + i, y + j] \cdot di \cdot dj \]

```plaintext
for t=0:T-1  # sampling on discrete timestep
  x = Ox + Dx*t  # O : ray origin, D : ray direction
  y = Oy + Dy*t
  z = Oz + Dz*t
  for i=0.0:1.0    # continuous
    for j=0.0:1.0    # continuous
      for k=0.0:1.0    # continuous
        Out[t] += Grid[x+i,y+j,z+k]*d(i)*d(j)*d(k)
```

Compute interpolation on every sampled point in ray
Case Study: NeRF

9Lines vs. 82Lines (PyTorch)

```python
for t=0:T-1  # sampling on discrete timestep
    x = Ox + Dx*t  # O : ray origin, D : ray direction
    y = Oy + Dy*t
    z = Oz + Dz*t
    for i=0.0:1.0  # continuous
        for j=0.0:1.0  # continuous
            for k=0.0:1.0  # continuous
                for c=0.27  # interpolating 28 discrete features
                    Out[t,c] += Grid[x+i, y+j, z+k, c] * d(i) * d(j) * d(k)
```

\[
\int_0^1 \int_0^1 Grid[x+i, y+j] \cdot di \cdot dj
\]
Case Study: Nerf

Single threaded CPU

9Lines vs. 82Lines (PyTorch)
Hardware For Sparsity
Most sparse data never gets used in the computation (eg: SpMSpV)
- But they all travel through most of the pipeline
- Opportunities for near-memory filtering
Hardware For Sparsity

• Most sparse data never gets used in the computation (eg: SpMSpV)
  • But they all travel through most of the pipeline
  • Opportunities for near-memory filtering

• Structured formats have a lot of exploitable patterns
  • But machines don’t understand the formats, everything is just bits
    • Nvidia Ampere understands a single sparse format
  • Opportunities to compress storage and simple near-memory operations
Hardware For Sparsity

• Most sparse data never gets used in the computation (eg: SpMSpV)
  • But they all travel through most of the pipeline
  • Opportunities for near-memory filtering

• Structured formats have a lot of exploitable patterns
  • But machines don’t understand the formats, everything is just bits
    • Nvidia Ampere understands a single sparse format
  • Opportunities to compress storage and simple near-memory operations

• Compound operations have huge benefits (eg: SDDMM)
  • Doing simple binary operations (mem → op → mem) can be asymptotically bad
  • Opportunities for doing custom compound operations
• Most sparse data never gets used in the computation (eg: SpMSpV)
  • But they all travel through most of the pipeline
  • Opportunities for near-memory filtering

• Structured formats have a lot of exploitable patterns
  • But machines don’t understand the formats, everything is just bits
    • Nvidia Ampere understands a single sparse format
  • Opportunities to compress storage and simple near-memory operations

• Compound operations have huge benefits (eg: SDDMM)
  • Doing simple binary operations (mem → op → mem) can be asymptotically bad
  • Opportunities for doing custom compound operations
• Sparse-aware hardware can have a high impact
## Hardware For Sparsity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Name</th>
<th>Authors</th>
<th>Format</th>
<th>Dataflow</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpMV</td>
<td>MergeSpMV</td>
<td>CMU</td>
<td>CSR</td>
<td>Tiled Rowmajor</td>
<td>FPGA / ASIC</td>
</tr>
<tr>
<td></td>
<td>FPGASpMV</td>
<td>Univ.Florida / Microsoft</td>
<td>CSR Variant</td>
<td>Rowmajor</td>
<td>FPGA</td>
</tr>
<tr>
<td>SpMSpM</td>
<td>SIGMA</td>
<td>Georgia Tech, Intel</td>
<td>Bitmap</td>
<td>Inner product</td>
<td>ASIC</td>
</tr>
<tr>
<td></td>
<td>OuterSPACE</td>
<td>Michigan, Arizona state</td>
<td>(CSC,CSR)</td>
<td>Outer product</td>
<td>ASIC</td>
</tr>
<tr>
<td></td>
<td>GAMMA</td>
<td>MIT</td>
<td>(CSR,CSR)</td>
<td>Rowmajor (Gustavson)</td>
<td>ASIC</td>
</tr>
<tr>
<td>SpMM</td>
<td>NVIDIA Sparse Tensor Core</td>
<td>Nvidia</td>
<td>Structured CSR</td>
<td>?</td>
<td>GPU</td>
</tr>
<tr>
<td>Sparse Conv</td>
<td>SCNN</td>
<td>Nvidia, MIT, Berkeley, Stanford</td>
<td>CSF</td>
<td>Input Stationary</td>
<td>ASIC</td>
</tr>
<tr>
<td></td>
<td>FPGASpConv</td>
<td>Zhejiang University, USC</td>
<td>Tiled CSF</td>
<td>Tiled Output Stationary</td>
<td>FPGA</td>
</tr>
<tr>
<td>Sparse Trans</td>
<td>Sanger</td>
<td>Peking University</td>
<td>Blocked CSR</td>
<td>Fused</td>
<td>ASIC</td>
</tr>
<tr>
<td>Intersection</td>
<td>AVX512 VP2INTERSECT</td>
<td>Intel</td>
<td>Bitmap</td>
<td>?</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>SSE4.2</td>
<td>Intel</td>
<td>Compressed</td>
<td>?</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>ExTensor</td>
<td>UIUC, Nvidia</td>
<td>Various Formats</td>
<td>Tiled Innerproduct</td>
<td>ASIC</td>
</tr>
<tr>
<td></td>
<td>Einsum (Compiled)</td>
<td>Stanford, MIT</td>
<td>Various Formats</td>
<td>Various Dataflows</td>
<td>ASIC</td>
</tr>
</tbody>
</table>
Sparse Array Programming in the Python Ecosystem
Sparse Array Programming in the Python Ecosystem
Sparse Array Programming in the Python Ecosystem
Sparse Array Programming in the Python Ecosystem

SciPy

NumPy

Python
Sparse Array Programming in the Python Ecosystem

Estimated User Base

6-15 million

SciPy

10-25 million

NumPy

Python
Sparse Array Programming in the Python Ecosystem

Estimated User Base

6-15 million

10-25 million

SciPy

NumPy

PyData/Sparse

Python™
Speeding up Sparse Array Programming in the Python Ecosystem
Speeding up Sparse Array Programming in the Python Ecosystem
Conclusion
Conclusion

• Array Abstraction is:
  • Simple
  • In most imperative languages
  • Highest performance
  • Familiar to most programmers
  • Used a lot (and in almost all high-performance code)
Conclusion

• Array Abstraction is:
  • Simple
  • In most imperative languages
  • Highest performance
  • Familiar to most programmers
  • Used a lot (and in almost all high-performance code)

• Expanding from a
  Multi-dimensional, Rectilinear, Dense, Integer grid of points to
  Multi-dimensional, Rectilinear, Dense, Integer grid of points
  Increases the application domain for array programming
Conclusion

• Array Abstraction is:
  • Simple
  • In most imperative languages
  • Highest performance
  • Familiar to most programmers
  • Used a lot (and in almost all high-performance code)

• Expanding from a
  Multi-dimensional, Rectilinear, Dense, Integer grid of points to
  Multi-dimensional, Rectilinear, Dense, Integer grid of points
  Increases the application domain for array programming

• Need compiler support
  • To provide the simple array abstraction while maintaining high performance
Commit Group

- Current & recent projects
  - Finch: A DSL for structured data
  - TACO: A DSL for sparse tensor algebra
  - Netblocks: A DSL Custom Network Protocols
  - SEQ: A DSL for bio informatics
  - GraphIt: A DSL for graph analysts
  - BuildIt: A Multistage programming framework in C++
  - CoLa: A DSL for data compression
  - SimIt: A DSL for sparse systems
  - MILK: A DSL for Optimizing indirect memory references
  - Cimple: A DSL for Instruction and Memory Level Parallelism
  - Codon: A Pythonic DSL framework
  - Tiramisu: A polyhedral compiler for data parallel algorithms
  - Ithemal: Performance prediction using machine learning
  - VeGen: Generating Vectorizers for vector instructions beyond SIMD
  - Vemal: Vectorization using machine learning
  - goSLP & Revec: Modernizing vectorization technology
  - OpenTuner: An extensible framework for program autotuning
Thank You

http://tensor-compiler.org/

This Work Supported By: