SPLATT
Efficient and Parallel Sparse Tensor-Matrix Multiplication

Shaden Smith¹  Niranjay Ravindran  Nicholas D. Sidiropoulos
George Karypis

University of Minnesota
¹shaden@cs.umn.edu
Tensor Introduction

Tensors are matrices extended to higher dimensions.

Example

We can model an item tagging system with a \( \text{user} \times \text{item} \times \text{tag} \) tensor.

- Very sparse!
Canonical Polyadic Decomposition (CPD)

- Extension of the singular value decomposition.
- Rank-$F$ decomposition with $F \sim 10$
- Compute $A \in \mathbb{R}^{I \times F}$, $B \in \mathbb{R}^{J \times F}$, and $C \in \mathbb{R}^{K \times F}$
Khatri-Rao Product

- Column-wise Kronecker product
- \((I \times F) \odot (J \times F) = (IJ \times F)\)

\[
\mathbf{A} \odot \mathbf{B} = [a_1 \otimes b_1, a_2 \otimes b_2, \ldots, a_n \otimes b_n]
\]
CPD with Alternating Least Squares

Computing the CPD

- We use alternating least squares.
- We operate on $X_{(1)}$, the tensor flattened to a matrix along the first dimension.

$$A = X_{(1)}(C \odot B)(C^T C \ast B^T B)^{-1}$$
MTTKRP is the bottleneck of CPD

Explicitly forming $C \odot B$ is infeasible, so we do it in place.
Related Work
Sparse Tensor-Vector Products

\[
X(i, j, k) \ast \begin{bmatrix}
B(j, f)C(k, f) \\
\end{bmatrix}
\]

Tensor Toolbox

- Popular Matlab code today for sparse tensor work
- MTTKRP uses \( \text{nnz}(X) \) space and \( 3F \cdot \text{nnz}(X) \) FLOPs
- Parallelism is difficult during “shrinking” stage
GigaTensor is a recent algorithm developed for Hadoop
- Uses $O(nnz(\mathbf{X}))$ space but $5F \cdot nnz(\mathbf{X})$ FLOPs
- Computes a column at a time
Two sparse matrix-vector multiplications per column
Requires an auxiliary sparse matrix with as many nonzeros as there are non-empty fibers
$2F(\text{nnz}(\mathbf{X}) + P)$ FLOPs, with $P$ non-empty fibers
The Surprisingly Parallel spArse Tensor Toolkit

Contributions

- Fast algorithm and data structure for MTTKRP
- Cache friendly tensor reordering
- Cache blocking for temporal locality
SPLATT—Optimized Algorithm

\[ M(i, f) = \sum_{k=1}^{K} C(k, f) \sum_{j=1}^{J} x(i, j, k) B(j, f) \]

\[ M(i,:) = \sum_{k=1}^{K} C(k,:) \ast \sum_{j=1}^{J} x(i, j, k) B(j,:) \]
We compute rows at a time instead of columns
Access patterns much better
Same complexity as DFacTo!
Only $F$ extra memory for MTTKRP
Tensor Reordering

We reorder the tensor to improve the access patterns of B and C.
Tensor Reordering – Mode Independent

\[
\begin{bmatrix}
\alpha & \beta & 0 & 0 \\
0 & \gamma & 0 & \delta
\end{bmatrix}
\]

Graph Partitioning

- We model the sparsity structure of \( \mathbf{X} \) with a tripartite graph
  - Slices are vertices, nonzeros connect slices with a triangle
- Partitioning the graph finds regions with shared indices
- We reorder the tensor to group indices in the same partition
Tensor Reordering – Mode Dependent

\[
\begin{bmatrix}
\alpha & \beta & 0 & 0 \\
0 & \gamma & 0 & \delta
\end{bmatrix}
\]

Hypergraph Partitioning

- Instead, create a new reordering for each mode of computation
- Fibers are now vertices and slices are hyperedges
- Overheads?
Sparsity is Hard

- Tiling lets us schedule nonzeros to reuse indices already in cache
- Cost: more fibers
- Tensor sparsity forces us to grow tiles
Experimental Evaluation

Experimental Evaluation
## Summary of Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>nnz</th>
<th>density</th>
</tr>
</thead>
<tbody>
<tr>
<td>NELL-2</td>
<td>15K</td>
<td>15K</td>
<td>30K</td>
<td>77M</td>
<td>1.3e-05</td>
</tr>
<tr>
<td>Netflix</td>
<td>480K</td>
<td>18K</td>
<td>2K</td>
<td>100M</td>
<td>5.4e-06</td>
</tr>
<tr>
<td>Delicious</td>
<td>532K</td>
<td>17M</td>
<td>2.5M</td>
<td>140M</td>
<td>6.1e-12</td>
</tr>
<tr>
<td>NELL-1</td>
<td>4M</td>
<td>4M</td>
<td>25M</td>
<td>144M</td>
<td>3.1e-13</td>
</tr>
</tbody>
</table>
### Effects of Tensor Reordering

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Random</th>
<th>Mode-Independent</th>
<th>Mode-Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>NELL-2</td>
<td>2.78</td>
<td>2.61 (1.06×)</td>
<td>2.60 (1.06×)</td>
</tr>
<tr>
<td>Netflix</td>
<td>6.02</td>
<td>5.26 (1.14×)</td>
<td>5.43 (1.10×)</td>
</tr>
<tr>
<td>Delicious</td>
<td>15.61</td>
<td>13.10 (1.19×)</td>
<td>12.51 (1.24×)</td>
</tr>
<tr>
<td>NELL-1</td>
<td>19.83</td>
<td>17.83 (1.11×)</td>
<td>17.55 (1.12×)</td>
</tr>
</tbody>
</table>

- **Small effect on serial performance**
- **Without cache blocking, a dense fiber can hurt cache reuse**

Shaden Smith, shaden@cs.umn.edu (U. Minnesota)
Effects of Cache Blocking

<table>
<thead>
<tr>
<th>Thds</th>
<th>SPLATT</th>
<th>tiled</th>
<th>MI+tiled</th>
<th>MD+tiled</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.14 (1.0×)</td>
<td>8.90 (0.9×)</td>
<td>8.70 (1.0×)</td>
<td>9.18 (0.9×)</td>
</tr>
<tr>
<td>2</td>
<td>4.73 (1.7×)</td>
<td>4.88 (1.7×)</td>
<td>4.37 (1.9×)</td>
<td>4.52 (1.8×)</td>
</tr>
<tr>
<td>4</td>
<td>2.54 (3.2×)</td>
<td>2.58 (3.2×)</td>
<td>2.29 (3.6×)</td>
<td>2.35 (3.5×)</td>
</tr>
<tr>
<td>8</td>
<td>1.42 (5.7×)</td>
<td>1.41 (5.8×)</td>
<td>1.26 (6.5×)</td>
<td>1.26 (6.4×)</td>
</tr>
<tr>
<td>16</td>
<td>0.90 (9.0×)</td>
<td>0.85 (9.5×)</td>
<td>0.74 (11.0×)</td>
<td>0.75 (10.8×)</td>
</tr>
</tbody>
</table>

MI and MD are mode-independent and mode-dependent reorderings, respectively.

- Cache blocking on its own is also not enough
- MI and MD are very competitive with tiling enabled
Scaling: Average Speedup vs TVec

![Graph showing the average speedup vs TVec for different tools.](image-url)
Scaling: NELL-2, Speedup vs TVec

![Graph showing speedup vs threads for different libraries: SPLATT, SPLATT+mem, GigaTensor, DFacTo, and TVec. The x-axis represents threads, and the y-axis represents speedup. Each library has a distinct line representing its performance pattern.](image-url)
Conclusions

Results
- SPLATT uses less memory than the state of the art
- Compared to DFacTo, we average $2.8 \times$ faster serially and $4.8 \times$ faster with 16 threads
- How?
  - Fast algorithm
  - Tensor reordering
  - Cache blocking

SPLATT
- Released as a C library
- cs.umn.edu/~shaden/software/