Efficient Factorization with Compressed Sparse Tensors

Shaden Smith    George Karypis

University of Minnesota
Department of Computer Science & Engineering
shaden@cs.umn.edu
Tensor Introduction

- Tensors are the generalization of matrices to $\geq 3D$
- Tensors have $m$ dimensions (or modes) and are $I_1 \times \ldots \times I_m$. 

![Diagram of a 3D tensor with dimensions labeled as users, contexts, and items.]
Canonical Polyadic Decomposition (CPD)

- We compute matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(m)}$, each with $F$ columns.

- Usually computed via *alternating least squares* (ALS).

- Matricized Tensor Times Khatri-Rao Product (MTTKRP) is the core computation of each iteration.
Uncompressed Tensors

\[
\begin{bmatrix}
 1 & 1 & 1 & 2 & 1. \\
 1 & 1 & 1 & 3 & 1. \\
 1 & 2 & 1 & 3 & 3. \\
 1 & 2 & 2 & 1 & 8. \\
 2 & 2 & 1 & 1 & 1. \\
 2 & 2 & 1 & 3 & 3. \\
 2 & 2 & 2 & 2 & 8.
\end{bmatrix}
\]
Uncompressed Tensors – MTTKRP

\[ \mathbf{A}^{(1)}(i,:) \leftarrow \mathbf{A}^{(1)}(i,:) + \mathbf{x}(i,j,k) \left[ \mathbf{A}^{(2)}(j,:) \ast \mathbf{A}^{(3)}(k,:) \right] \]
Consider three nonzeros in the fiber $\mathcal{X}(i,j,:)$ (a vector)

$$A^{(1)}(i,:)$ \leftarrow A^{(1)}(i,:)+\mathcal{X}(i,j,k_1)$ \quad $A^{(2)}(j,:)*A^{(3)}(k_1,:)$

$$A^{(1)}(i,:)$ \leftarrow A^{(1)}(i,:)+\mathcal{X}(i,j,k_2)$ \quad $A^{(2)}(j,:)*A^{(3)}(k_2,:)$

$$A^{(1)}(i,:)$ \leftarrow A^{(1)}(i,:)+\mathcal{X}(i,j,k_3)$ \quad $A^{(2)}(j,:)*A^{(3)}(k_3,:)$
Can we do better?

Consider three nonzeros in the fiber $\mathbf{x}(i, j, :)$ (a vector)

\[
\begin{align*}
\mathbf{A}^{(1)}(i, :) & \leftarrow \mathbf{A}^{(1)}(i, :) + \mathbf{x}(i, j, k_1) \\
\mathbf{A}^{(1)}(i, :) & \leftarrow \mathbf{A}^{(1)}(i, :) + \mathbf{x}(i, j, k_2) \\
\mathbf{A}^{(1)}(i, :) & \leftarrow \mathbf{A}^{(1)}(i, :) + \mathbf{x}(i, j, k_3)
\end{align*}
\]

A little factoring...

\[
\begin{align*}
\mathbf{A}^{(1)}(i, :) & \leftarrow \mathbf{A}^{(1)}(i, :) + \mathbf{A}^{(2)}(j, :) \ast \left[ \sum_{x=1}^{3} \mathbf{x}(i, j, k_x) \mathbf{A}^{(3)}(k_x, :) \right]
\end{align*}
\]
Data Structure

- Fibers are sparse vectors
- Slice $\mathcal{X}(i,:,::)$ is almost a CSR matrix...
- But, we need $m$ representations of $\mathcal{X}$
Compressed Sparse Fiber (CSF)

\[
\begin{bmatrix}
  i & j & k & l \\
  1 & 1 & 1 & 2 \\
  1 & 1 & 1 & 3 \\
  1 & 2 & 1 & 3 \\
  1 & 2 & 2 & 1 \\
  2 & 2 & 1 & 1 \\
  2 & 2 & 1 & 3 \\
  2 & 2 & 2 & 2 \\
\end{bmatrix}
\rightarrow
\begin{align*}
i & \quad 1 & \quad 2 \\
j & \quad 1 & \quad 2 & \quad 2 \\
k & \quad 1 & \quad 2 & \quad 1 & \quad 1 \\
l & \quad 2 & \quad 3 & \quad 3 & \quad 1 & \quad 1 & \quad 3 & \quad 2 \\
\end{align*}

http://cs.umn.edu/~splatt/
MTTKRP with a CSF Tensor

Objective

- We want to perform MTTKRP on each tensor mode with only one CSF representation.
- There are three types of nodes in a tree: root, internal, and leaf.
  - Each will have a tailored algorithm.
  - root and leaf are special cases of internal.
The leaf nodes determine the output location.

\[ A^{(4)}(l,:) \leftarrow A^{(4)}(l,:) + X(i,j,k,l) \left[ A^{(1)}(i,:) \ast A^{(2)}(j,:) \ast A^{(3)}(k,:) \right] \]
Hadamard products are pushed down the tree.
Hadamard products are pushed down the tree
Hadamard products are pushed down the tree
CSF-LEAF

Leaves designate write locations

![Diagram showing leaves and their corresponding write locations](http://cs.umn.edu/~splatt/)
Leaves designate write locations
The traversal continues...
The traversal continues...
The traversal continues...
The traversal continues...
The traversal continues...
\[
\sum_{x(i,j,:) \neq 0} A^{(2)}(j,:) \ast \left[ \sum_{x(i,j,k,:) \neq 0} A^{(3)}(k,:) \ast \left( \sum_{x(i,j,k,l) \neq 0} x(i,j,k,l) A^{(4)}(l,:) \right) \right]
\]
Inner products are accumulated in a buffer
Inner products are accumulated in a buffer
Hadamard products are then propagated up the CSF tree
Hadamard products are then propagated up the CSF tree.
Results are written to $A^{(1)}$
The traversal continues...
The traversal continues...
Partial results are kept in buffer
Inner products are accumulated in a buffer
Inner products are accumulated in a buffer
Results are written to $A^{(1)}$
Internal nodes use a combination of CSF-ROOT and CSF-LEAF.
Hadamard products are pushed down to the output level.
CSF-INTERNAL

- CSF-ROOT next pulls up to the output level
CSF-INTERNAL

- CSF-ROOT next pulls up to the output level
CSF-INTERNAL

CSF-ROOT next pulls up to the output level

Efficient Factorization with Compressed Sparse Tensors
CSF-ROOT next pulls up to the output level.
CSF-ROOT next pulls up to the output level
Parallelism – Tiling

- For $p$ threads, we do a $p$-way tiling of each tensor mode.
- Distributing the tiles allows us to eliminate the need for mutexes.
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>nnz</th>
</tr>
</thead>
<tbody>
<tr>
<td>NELL-2</td>
<td>12K</td>
<td>9K</td>
<td>28K</td>
<td>77M</td>
</tr>
<tr>
<td>Beer</td>
<td>33K</td>
<td>66K</td>
<td>960K</td>
<td>94M</td>
</tr>
<tr>
<td>Netflix</td>
<td>480K</td>
<td>18K</td>
<td>2K</td>
<td>100M</td>
</tr>
<tr>
<td>Delicious</td>
<td>532K</td>
<td>17M</td>
<td>3M</td>
<td>140M</td>
</tr>
<tr>
<td>NELL-1</td>
<td>3M</td>
<td>2M</td>
<td>25M</td>
<td>143M</td>
</tr>
<tr>
<td>Amazon</td>
<td>5M</td>
<td>18M</td>
<td>2M</td>
<td>1.7B</td>
</tr>
</tbody>
</table>
Storage Comparison

![Storage Comparison Chart]

- **NELL-2**
  - Tensor storage (GB): 3.7
- **Beer**
  - Tensor storage (GB): 2.3
- **Netflix**
  - Tensor storage (GB): 5.0
- **Delicious**
  - Tensor storage (GB): 8.2
- **NELL-1**
  - Tensor storage (GB): 99.3
- **Amazon**
  - Tensor storage (GB): 51.9

**Note:**
- SPLATT
- COORD
- CSF-M
- CSF-T

For more information, visit: [http://cs.umn.edu/~splatt/](http://cs.umn.edu/~splatt/)

Efficient Factorization with Compressed Sparse Tensors
Serial Comparison

![Graph showing speedup over COORD for different datasets and methods: SPLATT, CSF, and CSF+T. The x-axis represents different datasets (NELL-2, Beer, Netflix, Delicious, NELL-1, Amazon), while the y-axis represents speedup. The graph indicates varying performance across datasets, with Netflix showing a significant speedup for SPLATT and CSF+T.]
Conclusions

Compressed Sparse Fiber

- CSF uses 58% less memory than SPLATT while maintaining 81% of its performance
- CSF and related algorithms are now included in SPLATT

http://cs.umn.edu/~splatt/