Tensor-Matrix Products with a Compressed Sparse Tensor

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 $I_1 \times \ldots \times I_m$. 

![Tensor Diagram](http://cs.umn.edu/~splatt/)
Canonical Polyadic Decomposition (CPD)

- The CPD is an extension of the SVD to tensors
- We compute matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m$, each with $F$ columns and $\lambda$, a vector of weights.

$\mathbf{T} \approx \lambda_1 \mathbf{A}_1 + \cdots + \lambda_F \mathbf{A}_m$

- Usually computed via alternating least squares (ALS)
MTTKRP

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

- MTTKRP is the core computation of each iteration

\[ A_1 = X_{(1)} (A_m \odot \cdots \odot A_2) \]
Related Work
Uncompressed Tensors

- Stored as a list of coordinates
- \((i, j, k) = v\) represents one nonzero

\[
A_1(i,:) \leftarrow A_1(i,:) + \mathcal{X}(i,j,k)[A_2(j,:)*A_3(k,:)]
\]
Compressed Tensors

**SPLATT**
- SPLATT uses a hierarchical storage scheme for 3D tensors
- This allows for operation reduction and coarse-grained parallelism

http://cs.umn.edu/~splatt/
Contributions
Compressed Sparse Fiber (CSF)

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

\[
\begin{bmatrix}
  i \\
  j \\
  k \\
  l \\
\end{bmatrix}
\]

\[
\begin{array}{c}
  \text{i} \\
  \text{j} \\
  \text{k} \\
  \text{l} \\
\end{array}
\]

http://cs.umn.edu/~splatt/
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
We do a depth-first traversal on the CSF structure
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 accumulated when we reach the top
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
This time, Hadamard products are pushed \textit{down} the tree.
This time, Hadamard products are pushed *down* the tree.
This time, Hadamard products are pushed *down* the tree.
This time, Hadamard products are pushed *down* the tree
Leaves designate write locations
Leaves designate write locations
The traversal continues...
The traversal continues...
The traversal continues...
The traversal continues...
The traversal continues...
Internal nodes use a combination of CSF-ROOT and CSF-LEAF
Hadamard products are pushed down to the output level.
CSF-ROOT next pulls up to the output level
CSF-INTERNAL

CSF-ROOT next pulls up to the output level

http://cs.umn.edu/~splatt/
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
Parallelism – Tiling
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</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

http://cs.umn.edu/~splatt/
Speedup over COORD with 16 threads

- SPLATT
- CSF-M
- CSF-T

Datasets:
- NELL-2
- Beer
- Netflix
- Delicious
- NELL-1
- Amazon
Speedup over COORD with 16 threads for different datasets:

- NELL-2: SPLATT > CSF-M > CSF-T
- Beer: SPLATT > CSF-M > CSF-T
- Netflix: SPLATT > CSF-M > CSF-T
- Delicious: SPLATT > CSF-M > CSF-T
- NELL-1: SPLATT > CSF-M > CSF-T
- Amazon: SPLATT > CSF-M > CSF-T

The graph shows the speedup of SPLATT, CSF-M, and CSF-T over COORD with 16 threads for various datasets.
CSF-LEAF

The diagram illustrates the speedup over COORD with 16 threads for different datasets: NELL-2, Beer, Netflix, Delicious, NELL-1, and Amazon. The datasets are represented on the x-axis, and the speedup is shown on the y-axis. The bars are color-coded as follows:
- SPLATT (pink)
- CSF-M (blue)
- CSF-T (cyan)

The speedup is expressed as a percentage, with NELL-2 showing the highest speedup for SPLATT, followed by Netflix with a significant increase in speedup compared to COORD.
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/