A Medium-Grained Algorithm for Distributed Sparse Tensor Factorization

Shaden Smith  George Karypis

University of Minnesota
Department of Computer Science & Engineering
shaden@cs.umn.edu
Table of Contents

1 Preliminaries

2 Related Work: Coarse- and Fine-Grained Algorithms

3 A Medium-Grained Algorithm

4 Experiments

5 Conclusions
**Tensor Introduction**

- Tensors are the generalization of matrices to $\geq 3D$.
- Tensors have $m$ dimensions (or *modes*) and are $I_1 \times \ldots \times I_m$.
  - We’ll stick to $m = 3$ in this talk and call dimensions $I, J, K$.
Canonical Polyadic Decomposition (CPD)

- We compute matrices $A, B, C$, each with $F$ columns
  - $F$ is assumed to be small, on the order of 10 or 50

- Usually computed via *alternating least squares* (ALS)
- As a result, computations are *mode-centric*
Algorithm 1 CPD-ALS

1: while not converged do
2: \[ A^T = (C^T C \ast B^T B)^{-1} \left( X_1 (C \odot B) \right)^T \]
3: \[ B^T = (C^T C \ast A^T A)^{-1} \left( X_2 (C \odot A) \right)^T \]
4: \[ C^T = (B^T B \ast A^T A)^{-1} \left( X_3 (B \odot A) \right)^T \]
5: end while
### Algorithm 2 One mode of CPD-ALS

1. $\hat{A} \leftarrow X(1) (C \odot B)$ \(\triangleright \mathcal{O}(F \cdot \text{nnz}(X))\)
2. $LL^T \leftarrow \text{Cholesky}(C^T C \ast B^T B)$ \(\triangleright \mathcal{O}(F^3)\)
3. $A^T = (LL^T)^{-1} \hat{A}^T$ \(\triangleright \mathcal{O}(IF^2)\)
4. Compute $A^T A$ \(\triangleright \mathcal{O}(IF^2)\)

- Step 1 is the most expensive and the focus of this talk
Matricized Tensor Times Khatri-Rao Product (MTTKRP)

\[
\hat{A}(i,:) \leftarrow \hat{A}(i,:) + \chi(i,j,k) [B(j,:) \ast C(k,:)]
\]
\[ \hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + \mathbf{X}(i,j_1,k) [\mathbf{B}(j_1,:) \ast \mathbf{C}(k,:)] \]

\[ \hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + \mathbf{X}(i,j_2,k) [\mathbf{B}(j_2,:) \ast \mathbf{C}(k,:)] \]
Table of Contents

1 Preliminaries

2 Related Work: Coarse- and Fine-Grained Algorithms

3 A Medium-Grained Algorithm

4 Experiments

5 Conclusions
Coarse-Grained Decomposition

[Choi & Vishwanathan 2014, Shin & Kang 2014]

- Processes own complete slices of $\mathbf{X}$ and aligned factor rows
- $l/p$ rows communicated to $p-1$ processes after each update
Fine-Grained Decomposition

[Kaya & Uçar 2015]

- Most flexible: non-zeros individually assigned to processes
- Two communication steps
  1. Aggregate partial computations after MTTKRP
  2. Exchange new factor values
- Factors can be assigned to minimize communication
Finding a Fine-Grained Decomposition

Some options:
- Random assignment
- Hypergraph partitioning
- Multi-constraint hypergraph partitioning

In Practice: Hypergraph Model
- $\text{nnz}(\mathcal{X})$ vertices and $I+J+K$ hyperedges
- Tight approximation of communication and load balance
  - Distribution of factors must be considered: in practice a greedy solution works well
Table of Contents

1 Preliminaries

2 Related Work: Coarse- and Fine-Grained Algorithms

3 A Medium-Grained Algorithm

4 Experiments

5 Conclusions
Medium-Grained Decomposition

- Distribute over a grid of $p = q \times r \times s$ partitions
- $r \times s$ processes divide each $A_1, \ldots, A_q$
- Two communication steps like fine-grained
  - $O(l/p)$ rows communicated to $r \times s$ processes
Medium-Grained Decomposition

Each process owns roughly $l/p$ rows of each factor

Like before, a greedy algorithm works well
Finding a Medium-Grained Decomposition

Greedy Algorithm

1. Apply a random relabeling to modes of $\mathbf{X}$
2. Choose a decomposition dimension (algorithm in paper)
3. Compute 1D partitionings of each mode
   - Greedily chosen with load balance objective
4. Intersect!
5. Distribute factors with objective of reducing communication
# Table of Contents

1. Preliminaries

2. Related Work: Coarse- and Fine-Grained Algorithms

3. A Medium-Grained Algorithm

4. Experiments

5. Conclusions
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$I$</th>
<th>$J$</th>
<th>$K$</th>
<th>nnz</th>
</tr>
</thead>
<tbody>
<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</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>
<tr>
<td>Random1</td>
<td>20M</td>
<td>20M</td>
<td>20M</td>
<td>1.0B</td>
</tr>
<tr>
<td>Random2</td>
<td>50M</td>
<td>5M</td>
<td>5M</td>
<td>1.0B</td>
</tr>
</tbody>
</table>

[http://cs.umn.edu/~splatt/](http://cs.umn.edu/~splatt/)
## Load Balance

**Table:** Load imbalance with 64 and 128 processes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>coarse 64</th>
<th>coarse 128</th>
<th>medium 64</th>
<th>medium 128</th>
<th>fine 64</th>
<th>fine 128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>1.03</td>
<td>1.18</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Delicious</td>
<td>1.21</td>
<td>1.41</td>
<td>1.01</td>
<td>1.06</td>
<td>1.00</td>
<td>1.05</td>
</tr>
<tr>
<td>NELL</td>
<td>1.12</td>
<td>1.29</td>
<td>1.01</td>
<td>1.01</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Amazon</td>
<td>2.17</td>
<td>3.86</td>
<td>1.08</td>
<td>1.08</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Communication Volume

Average Communication Volume

Maximum Communication Volume

http://cs.umn.edu/~splatt/
Strong Scaling: Amazon

![Strong Scaling Graph]

**Graph Description:**
- **X-axis:** Number of cores
- **Y-axis:** Time per iteration
- **Legend:**
  - DFacTo
  - medium
  - ideal
  - coarse
- **Data Points:**
  - 64 cores: DFacTo
  - 128 cores: medium
  - 256 cores: ideal
  - 512 cores: coarse
  - 1024 cores: medium

**Graph Analysis:**
- The graph shows the scaling performance of different algorithms (DFacTo, medium, ideal, coarse) across varying numbers of cores.
- The ideal scaling line demonstrates the expected linear decrease in time per iteration with an increase in the number of cores.
- The coarse algorithm shows a less efficient scaling pattern compared to the ideal.
- The medium algorithm lies between the ideal and coarse, indicating a balanced performance.
- The DFacTo algorithm performs differently, highlighting its specific characteristics.

**Source:**
http://cs.umn.edu/~splatt/
Table of Contents

1 Preliminaries

2 Related Work: Coarse- and Fine-Grained Algorithms

3 A Medium-Grained Algorithm

4 Experiments

5 Conclusions
Wrapping Up...

- Medium-grained decompositions are a good middle-ground
- $1.5 \times$ to $5 \times$ faster than fine-grained decompositions with hypergraph partitioning
- DMS is $40 \times$ to $80 \times$ faster than DFactO, the fastest publicly available software

http://cs.umn.edu/~splatt/
Choosing the Shape of the Decomposition

Objective

- We need to find $q, r, s$ such that $q \times r \times s = p$
- Tensors modes are often very skewed (480k Netflix users vs 2k days)
  - We want to assign processes proportionally
  - 1D decompositions actually work well for many tensors

Algorithm

1. Start with a $1 \times 1 \times 1$ shape
2. Compute the prime factorization of $p$
3. For each prime factor $f$, starting from the largest, multiply the most imbalanced mode by $f$