Scalable Sparse Tensor Decompositions in Distributed Memory Systems

Oguz Kaya

INRIA and LIP, ENS Lyon, France

Sparse Days, June 29-July 2, 2015, St. Girons, France

Joint work with:

Bora Uçar, CNRS and LIP, ENS Lyon, France
What is a tensor?

- A vector is a 1-dimensional tensor.
- A matrix is a 2-dimensional tensor.
- A tensor $\mathcal{X} \in \mathbb{R}^{l_1 \times l_2 \times \ldots \times l_d}$ has $d$ dimensions.
Introduction and Motivation

What is a tensor?

- A vector is a 1-dimensional tensor.
- A matrix is a 2-dimensional tensor.
- A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ has $d$ dimensions.
Introduction and Motivation

What is a tensor?

- A vector is a 1-dimensional tensor.
- A matrix is a 2-dimensional tensor.
- A tensor $\mathbf{X} \in \mathbb{R}^{l_1 \times l_2 \times \cdots \times l_d}$ has $d$ dimensions.
Introduction and Motivation

What is a tensor?

- A vector is a 1-dimensional tensor.
- A matrix is a 2-dimensional tensor.
- A tensor $\mathbf{X} \in \mathbb{R}^{l_1 \times l_2 \times \ldots \times l_d}$ has $d$ dimensions.
Tensor decompositions aim to approximate a low-rank tensor.

- Used to predict missing entries, extract latent information, compression
- Applications in data mining, chemometrics, signal processing, etc.

**Goal:** To compute sparse tensor factorization in parallel
Related Work

- **GigaTensor**: Distributed memory, using map-reduce framework
- **DFacTo**: Distributed memory, SpMV formulation, A2A communication (MPI, C++)
- **SPLATT**: Shared memory, cache-efficient reorderings and tilings (OpenMP, C)

**Our contributions:**

- Two parallel algorithms using P2P communication in distributed memory, and their efficient implementations in our sparse tensor library, **HyperTensor** (MPI, C++)
- Hypergraph models of computation for reducing computational imbalance and communication requirements
Outline

1 Introduction

2 CP Decomposition

3 Coarse-Grain Parallelization

4 Fine-Grain Parallelization

5 Experiments and Results

6 Conclusion and Future Work
CANDECOMP/PARAFAC (CP) Decomposition

\[
X = \sum_{r=1}^{R} \mathbf{a}_r \mathbf{b}_r \mathbf{c}^T_r + \cdots + \sum_{r=1}^{R} \mathbf{a}_r \mathbf{b}_r \mathbf{c}^T_r = \mathbf{B} \mathbf{A} \mathbf{C}^T
\]

- \((B^T B \ast C^T C)^\dagger\) is of size \(R \times R\).
- For \(X \in \mathbb{R}^{I \times J \times K}\), \(X(1)\) is a sparse matrix of size \(I \times JK\).
- For \(B \in \mathbb{R}^{J \times R}\) and \(C \in \mathbb{R}^{K \times R}\), \((C \otimes B)\) is a dense matrix of size \(JK \times R\).
- \(M_A = X(1)(C \otimes B)\) is of size \(I \times R\) (Matricized Tensor-Times Khatri-Rao Product).
- Forming \(C \otimes B\) explicitly is not feasible; so \(M_A\) is computed implicitly.
- The update \(A = M_A(B^T B \ast C^T C)^\dagger\) is trivial.

Algorithm 1: CP-ALS for 3rd order tensors

**Input**: \(X\): tensor
- \(R\): The rank

**Output**: CP decomposition \([\lambda, \mathbf{A}, \mathbf{B}, \mathbf{C}]\)

```
repeat
    \(A \leftarrow X(1)(C \otimes B)(B^T B \ast C^T C)^\dagger\)
    Normalize columns of \(A\)
    \(B \leftarrow X(2)(C \otimes A)(A^T A \ast C^T C)^\dagger\)
    Normalize columns of \(B\)
    \(C \leftarrow X(3)(B \otimes A)(A^T A \ast B^T B)^\dagger\)
    Normalize columns of \(C\) and store the norms as \(\lambda\)
until no improvement or maximum iterations reached
```
**CANDECOMP/PARAFAC (CP) Decomposition**

\[
X = \bigoplus_{i=1}^{R} a_i \otimes b_i \otimes c_i + \cdots + \bigoplus_{i=1}^{R} a_R \otimes b_R \otimes c_R
\]

- \((B^TB \ast C^TC)^\dagger\) is of size \(R \times R\).
- For \(X \in \mathbb{R}^{I \times J \times K}\), \(X_{(1)}\) is a sparse matrix of size \(I \times JK\).
- For \(B \in \mathbb{R}^{J \times R}\) and \(C \in \mathbb{R}^{K \times R}\), \((C \odot B)\) is a dense matrix of size \(JK \times R\).
- \(M_A = X_{(1)}(C \odot B)\) is of size \(I \times R\) (Matricized Tensor-Times Khatri-Rao Product).

\[\Rightarrow\] Forming \(C \odot B\) explicitly is not feasible; so \(M_A\) is computed implicitly.

- The update \(A = M_A(B^TB \ast C^TC)^\dagger\) is trivial.

**Algorithm 2: CP-ALS for 3rd order tensors**

**Input**: \(X\): tensor
\(R\): The rank

**Output**: CP decomposition \([\lambda, A, B, C]\)

repeat
  \(A \leftarrow X_{(1)}(C \odot B)(B^TB \ast C^TC)^\dagger\)
  Normalize columns of \(A\)
  \(B \leftarrow X_{(2)}(C \odot A)(A^TA \ast C^TC)^\dagger\)
  Normalize columns of \(B\)
  \(C \leftarrow X_{(3)}(B \odot A)(A^TA \ast B^TB)^\dagger\)
  Normalize columns of \(C\) and store the norms as \(\lambda\)
until no improvement or maximum iterations reached
CANDECOMP/PARAFAC (CP) Decomposition

\[
\mathbf{X} = \sum_{i=1}^{R} a_i \otimes b_i \otimes c_i + \cdots + \sum_{i=1}^{R} a_i \otimes b_i \otimes c_i
\]

- \((\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\) is of size \(R \times R\).
- For \(\mathbf{X} \in \mathbb{R}^{I \times J \times K}\), \(\mathbf{X}_{(1)}\) is a sparse matrix of size \(I \times JK\).
- For \(\mathbf{B} \in \mathbb{R}^{J \times R}\) and \(\mathbf{C} \in \mathbb{R}^{K \times R}\), \((\mathbf{C} \odot \mathbf{B})\) is a dense matrix of size \(JK \times R\).
- \(\mathbf{M}_A = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})\) is of size \(I \times R\) (Matricized Tensor-Times Khatri-Rao Product).
- Forming \(\mathbf{C} \odot \mathbf{B}\) explicitly is not feasible; so \(\mathbf{M}_A\) is computed implicitly.
- The update \(\mathbf{A} = \mathbf{M}_A(\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\) is trivial.

**Algorithm 3: CP-ALS for 3rd order tensors**

**Input**: \(\mathbf{X}\): tensor
\(R\): The rank

**Output**: CP decomposition \([\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\)

```
repeat
    \(\mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\)
    Normalize columns of \(\mathbf{A}\)
    \(\mathbf{B} \leftarrow \mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})(\mathbf{A}^T \mathbf{A} \ast \mathbf{C}^T \mathbf{C})^\dagger\)
    Normalize columns of \(\mathbf{B}\)
    \(\mathbf{C} \leftarrow \mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{A}^T \mathbf{A} \ast \mathbf{B}^T \mathbf{B})^\dagger\)
    Normalize columns of \(\mathbf{C}\) and store the norms as \(\lambda\)
until no improvement or maximum iterations reached
```
CANDECOMP/PARAFAC (CP) Decomposition

\[ \mathbf{X} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r \]

- \((\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\) is of size \(R \times R\).
- For \(\mathbf{X} \in \mathbb{R}^{I \times J \times K}\), \(\mathbf{X}^{(1)}\) is a sparse matrix of size \(I \times JK\).
- For \(\mathbf{B} \in \mathbb{R}^{J \times R}\) and \(\mathbf{C} \in \mathbb{R}^{K \times R}\), \((\mathbf{C} \odot \mathbf{B})\) is a dense matrix of size \(JK \times R\).
- \(\mathbf{M}_A = \mathbf{X}^{(1)}(\mathbf{C} \odot \mathbf{B})\) is of size \(I \times R\) (Matricized Tensor-Times Khatri-Rao Product).
- Forming \(\mathbf{C} \odot \mathbf{B}\) explicitly is not feasible; so \(\mathbf{M}_A\) is computed implicitly.
- The update \(\mathbf{A} = \mathbf{M}_A(\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\) is trivial.

**Algorithm 4: CP-ALS for 3rd order tensors**

**Input**: \(\mathbf{X}\): tensor
- \(R\): The rank

**Output**: CP decomposition \([\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\)

```
repeat
\(A \leftarrow \mathbf{X}^{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\)
Normalize columns of \(\mathbf{A}\)
\(B \leftarrow \mathbf{X}^{(2)}(\mathbf{C} \odot \mathbf{A})(\mathbf{A}^T \mathbf{A} \ast \mathbf{C}^T \mathbf{C})^\dagger\)
Normalize columns of \(\mathbf{B}\)
\(C \leftarrow \mathbf{X}^{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{A}^T \mathbf{A} \ast \mathbf{B}^T \mathbf{B})^\dagger\)
Normalize columns of \(\mathbf{C}\) and store the norms as \(\lambda\)
until no improvement or maximum iterations reached
```
**CANDECOMP/PARAFAC (CP) Decomposition**

- \((B^T B \ast C^T C)^\dagger\) is of size \(R \times R\).
- For \(X \in \mathbb{R}^{I \times J \times K}\), \(X^{(1)}\) is a sparse matrix of size \(I \times JK\).
- For \(B \in \mathbb{R}^{J \times R}\) and \(C \in \mathbb{R}^{K \times R}\), \((C \odot B)\) is a dense matrix of size \(JK \times R\).
- \(M_A = X^{(1)}(C \odot B)\) is of size \(I \times R\) (Matricized Tensor-Times Khatri-Rao Product).
  - \(\Rightarrow\) Forming \(C \odot B\) explicitly is not feasible; so \(M_A\) is computed implicitly.
- The update \(A = M_A(B^T B \ast C^T C)^\dagger\) is trivial.

**Algorithm 5:** CP-ALS for 3rd order tensors

**Input:** \(X\): tensor
\(R\): The rank

**Output:** CP decomposition \([\lambda, A, B, C]\)

**repeat**
- \(A \leftarrow X^{(1)}(C \odot B)(B^T B \ast C^T C)^\dagger\)
  - Normalize columns of \(A\)
- \(B \leftarrow X^{(2)}(C \odot A)(A^T A \ast C^T C)^\dagger\)
  - Normalize columns of \(B\)
- \(C \leftarrow X^{(3)}(B \odot A)(A^T A \ast B^T B)^\dagger\)
  - Normalize columns of \(C\) and store the norms as \(\lambda\)

**until no improvement or maximum iterations reached**
CANDECOMP/PARAFAC (CP) Decomposition

\[ \mathbf{X} = \sum_{r=1}^{R} \left( \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r \right) \]

- \((\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\) is of size \(R \times R\).
- For \(\mathbf{X} \in \mathbb{R}^{I \times J \times K}\), \(\mathbf{X}_{(1)}\) is a sparse matrix of size \(I \times JK\).
- For \(\mathbf{B} \in \mathbb{R}^{J \times R}\) and \(\mathbf{C} \in \mathbb{R}^{K \times R}\), \((\mathbf{C} \circ \mathbf{B})\) is a dense matrix of size \(JK \times R\).
- \(\mathbf{M}_A = \mathbf{X}_{(1)}(\mathbf{C} \circ \mathbf{B})\) is of size \(I \times R\) (Matricized Tensor-Times Khatri-Rao Product).
  \[ \Rightarrow \text{Forming } \mathbf{C} \circ \mathbf{B} \text{ explicitly is not feasible; so } \mathbf{M}_A \text{ is computed implicitly.} \]
- The update \(\mathbf{A} = \mathbf{M}_A(\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger\) is trivial.

**Algorithm 6: CP-ALS for 3rd order tensors**

**Input**: \(\mathbf{X}\): tensor

**R**: The rank

**Output**: CP decomposition \([\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\)

**repeat**

\[ \mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \circ \mathbf{B})(\mathbf{B}^T \mathbf{B} \ast \mathbf{C}^T \mathbf{C})^\dagger \]

Normalize columns of \(\mathbf{A}\)

\[ \mathbf{B} \leftarrow \mathbf{X}_{(2)}(\mathbf{C} \circ \mathbf{A})(\mathbf{A}^T \mathbf{A} \ast \mathbf{C}^T \mathbf{C})^\dagger \]

Normalize columns of \(\mathbf{B}\)

\[ \mathbf{C} \leftarrow \mathbf{X}_{(3)}(\mathbf{B} \circ \mathbf{A})(\mathbf{A}^T \mathbf{A} \ast \mathbf{B}^T \mathbf{B})^\dagger \]

Normalize columns of \(\mathbf{C}\) and store the norms as \(\lambda\)

**until no improvement or maximum iterations reached**
Matricized Tensor-Times Khatri-Rao Product (MTTKRP)

- \( M_A = X_{(1)}(C \odot B), M_A \in \mathbb{R}^{I \times R} \)
- It's very much like SpMV!!
- In SpMV, for each nonzero \( x_{i,j} \), we perform \( a(i) = a(i) + x_{i,j} \ast b(j) \).
- In MTTKRP, \( x_{i,j,k} \) multiplies vectors \( B(j,:) \) and \( C(k,:) \) and updates \( M_A(i,:) \).
- How to parallelize?

Algorithm 7: MTTKRP for the 3rd order tensors

\[ M_A = X_{(1)}(B \odot C) \]

**Input**: \( X \): tensor
- \( B, C \): Factor matrices in all modes except the first
- \( I_A \): Number of rows of the factor \( A \)
- \( R \): Rank of the factors

**Output**: \( M_A \)

Initialize \( M_A \) to zeros of size \( I_A \times R \)

foreach \( x_{i,j,k} \in X \) do

1. \[ M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \ast C(k,:)] \]
Matricized Tensor-Times Khatri-Rao Product (MTTKRP)

- $M_A = X(1)(C \odot B)$, $M_A \in \mathbb{R}^{I \times R}$
- It’s very much like SpMV!!
- In SpMV, for each nonzero $x_{i,j}$, we perform $a(i) = a(i) + x_{i,j} * b(j)$.
- In MTTKRP, $x_{i,j,k}$ multiplies vectors $B(j,:)$ and $C(k,:)$ and updates $M_A(i,:)$.
- How to parallelize?

**Algorithm 8: MTTKRP for the 3rd order tensors**

**Input:** $X$: tensor  
$B, C$: Factor matrices in all modes except the first  
$I_A$: Number of rows of the factor $A$  
$R$: Rank of the factors

**Output:** $M_A = X(1)(B \odot C)$

Initialize $M_A$ to zeros of size $I_A \times R$

foreach $x_{i,j,k} \in X$ do

1. $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) * C(k,:)]$
Matricized Tensor-Times Khatri-Rao Product (MTTKRP)

- $M_A = X(1)(C \odot B), \quad M_A \in \mathbb{R}^{I \times R}$
- It's very much like SpMV!!
- In SpMV, for each nonzero $x_{i,j}$, we perform $a(i) = a(i) + x_{i,j} \times b(j)$.
- In MTTKRP, $x_{i,j,k}$ multiplies vectors $B(j,:) \text{ and } C(k,:) \text{ and updates } M_A(i,:)$. 
- How to parallelize?

**Algorithm 9: MTTKRP for the 3rd order tensors**

**Input:**
- $X$: tensor
- $B, C$: Factor matrices in all modes except the first
- $I_A$: Number of rows of the factor $A$
- $R$: Rank of the factors

**Output:** $M_A = X(1)(B \odot C)$

Initialize $M_A$ to zeros of size $I_A \times R$

**foreach** $x_{i,j,k} \in X$ **do**

1. $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \times C(k,:)]$
Matricized Tensor-Times Khatri-Rao Product (MTTKRP)

- $M_A = X(1)(C \odot B)$, $M_A \in \mathbb{R}^{I \times R}$
- It’s very much like SpMV!!
- In SpMV, for each nonzero $x_{i,j}$, we perform $a(i) = a(i) + x_{i,j} \cdot b(j)$.
- In MTTKRP, $x_{i,j,k}$ multiplies vectors $B(j,:)$ and $C(k,:)$ and updates $M_A(i,:)$.
- How to parallelize?

**Algorithm 10**: MTTKRP for the 3rd order tensors

**Input**
- $X$: tensor
- $B, C$: Factor matrices in all modes except the first
- $I_A$: Number of rows of the factor $A$
- $R$: Rank of the factors

**Output**: $M_A = X(1)(B \odot C)$

Initialize $M_A$ to zeros of size $I_A \times R$

foreach $x_{i,j,k} \in X$ do

1. $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \cdot C(k,:)]$
Matricized Tensor-Times Khatri-Rao Product (MTTKRP)

- \( M_A = X(1)(C \odot B), \ M_A \in \mathbb{R}^{I \times R} \)
- It's very much like SpMV!!
- In SpMV, for each nonzero \( x_{i,j} \), we perform \( a(i) = a(i) + x_{i,j} \times b(j) \).
- In MTTKRP, \( x_{i,j,k} \) multiplies vectors \( B(j,:) \) and \( C(k,:) \) and updates \( M_A(i,:) \).
- How to parallelize?

**Algorithm 11**: MTTKRP for the 3rd order tensors

<table>
<thead>
<tr>
<th>Input</th>
<th>( \mathcal{X} ): tensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B, C ): Factor matrices in all modes except the first</td>
<td></td>
</tr>
<tr>
<td>( I_A ): Number of rows of the factor ( A )</td>
<td></td>
</tr>
<tr>
<td>( R ): Rank of the factors</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>( M_A = X(1)(B \odot C) )</th>
</tr>
</thead>
</table>

Initialize \( M_A \) to zeros of size \( I_A \times R \)

for each \( x_{i,j,k} \in \mathcal{X} \)
do

1. \( M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \times C(k,:)] \)
Coarse-Grain MTTKRP within CP-ALS

- Computing any row $M_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$.
  $\Rightarrow$ Owner of $M_A(i,:)$ is responsible for computing $A(i,:) = M_A(i,:)(B^TB \ast C^TC)^\dagger$.
- $B^TB$ and $C^TC$ are available to all processes.
- Atomic tasks are partitioned to processes; process $p$ owns the set of rows $M_A(I_p,:)$. Each process $p$ has all the nonzeros in the tensor slice $X(I_p,:)$. For $x_{i,j,k} \in X(I_p,:)$, $B(j,:)$ and $C(k,:)$ are available to process $p$.
- Make sure at the end all the premises of the next iteration satisfy.
- How to decide on $I_p$?

Algorithm 12: Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p_p$ within CP-ALS

```
Input : $I_p$, indices where $\mu_A(i) = p$
- $X(I_p,:)$, tensor slices
- $B^TB$ and $C^TC$, $R \times R$ matrices
- $B(J_p,:), C(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $X(I_p,:)$

On exit : $A(I_p,:)$ is computed, its rows are sent to processes needing them; $A^TA$ is available

Initialize $M_A(I_p,:)$ to all zeros of size $|I_p| \times R$

foreach $i \in I_p$ do
  foreach $x_{i,j,k} \in X(I_p,:)$ do
    $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \ast C(k,:)]$
  endforeach

2 $A(I_p,:) \leftarrow M_A(I_p,:)(B^TB \ast C^TC)^\dagger$

foreach $i \in I_p$ do
  Send $A(i,:)$ to all processes having nonzeros in $X(I_p,:)$

4 Locally compute $A(I_p,:)^TA(I_p,:)$ and all-reduce the results to form $A^TA$
```
Coarse-Grain MTTKRP within CP-ALS

- Computing any row $M_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$.
  $\Rightarrow$ Owner of $M_A(i,:)$ is responsible for computing $A(i,:) = M_A(i,:)(B^TB \ast C^TC)^\dagger$.

  $B^TB$ and $C^TC$ are available to all processes.

- Atomic tasks are partitioned to processes; process $p$ owns the set of rows $M_A(I_p,:)$. Each process $p$ has all the nonzeros in the tensor slice $X(I_p,:,:)$.

  For $x_{i,j,k} \in X(I_p,:,:)$, $B(j,:)$ and $C(k,:)$ are available to process $p$.

- Make sure at the end all the premises of the next iteration satisfy.

- How to decide on $I_p$?

**Algorithm 13:** Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p_p$ within CP-ALS

**Input:** $I_p$, indices where $\mu_A(i) = p$

- $X_{I_p,:}$: tensor slices
- $B^T B$ and $C^T C$, $R \times R$ matrices
- $B(J_p,:), C(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $X_{I_p,:}$

**On exit:** $A(I_p,:)$ is computed, its rows are sent to processes needing them; $A^TA$ is available

Initialize $M_A(I_p,:)$ to all zeros of size $|I_p| \times R$

**foreach** $i \in I_p$ do

1. **foreach** $x_{i,j,k} \in X_{I_p,:}$ do

   $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \ast C(k,:)]$

2. $A(I_p,:) \leftarrow M_A(I_p,:)(B^T B \ast C^T C)^\dagger$

**foreach** $i \in I_p$ do

3. Send $A(i,:)$ to all processes having nonzeros in $X_{I_p,:}$

4. Locally compute $A(I_p,:)^T A(I_p,:)$ and all-reduce the results to form $A^TA$
Coarse-Grain MTTKRP within CP-ALS

- Computing any row $M_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$.
  $\Rightarrow$ Owner of $M_A(i,:)$ is responsible for computing $A(i,:) = M_A(i,:)(B^TB \star C^TC)^\dagger$.

- $B^TB$ and $C^TC$ are available to all processes.

- Atomic tasks are partitioned to processes; process $p$ owns the set of rows $M_A(I_p,:)$.

- Each process $p$ has all the nonzeros in the tensor slice $X(I_p,:,:)$.

- For $x_{i,j,k} \in X(I_p,:,:)$, $B(j,:)$ and $C(k,:)$ are available to process $p$.

- Make sure at the end all the premises of the next iteration satisfy.

- How to decide on $I_p$?

**Algorithm 14**: Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p$ within CP-ALS

**Input**: $I_p$, indices where $\mu_A(i) = p$
- $X_{I_p,:,:}$, tensor slices
- $B^TB$ and $C^TC$, $R \times R$ matrices
- $B(J_p,:), C(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $X_{I_p,:,:}$

**On exit**: $A(I_p,:)$ is computed, its rows are sent to processes needing them; $A^TA$ is available

**Initialize** $M_A(I_p,:)$ to all zeros of size $|I_p| \times R$

**foreach** $i \in I_p$ do
  **foreach** $x_{i,j,k} \in X_{I_p,:,:}$ do
    $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \star C(k,:)]$
  endfor
  $A(I_p,:) \leftarrow M_A(I_p,:)(B^TB \star C^TC)^\dagger$

**foreach** $i \in I_p$ do
  Send $A(i,:)$ to all processes having nonzeros in $X_{I_p,:,:}$
  Locally compute $A(I_p,:)^TA(I_p,:)$ and all-reduce the results to form $A^TA$
Computing any row $M_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$.

$\Rightarrow$ Owner of $M_A(i,:)$ is responsible for computing $A(i,:) = M_A(i,:)(B^TB \ast C^TC)^\dagger$.

$B^TB$ and $C^TC$ are available to all processes.

Atomic tasks are partitioned to processes; process $p$ owns the set of rows $M_A(I_p,:)$.

Each process $p$ has all the nonzeros in the tensor slice $X(I_p,:,:)$.

For $x_{i,j,k} \in X(I_p,:,:)$, $B(j,:)$ and $C(k,:)$ are available to process $p$.

Make sure at the end all the premises of the next iteration satisfy.

How to decide on $I_p$?

---

**Algorithm 15**: Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p$ within CP-ALS

**Input**: $I_p$, indices where $\mu_A(i) = p$
- $X(I_p,:,:)$: tensor slices
- $B^TB$ and $C^TC$, $R \times R$ matrices
- $B(J_p,:), C(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $X(I_p,:,:)$

**On exit**: $A(I_p,:)$ is computed, its rows are sent to processes needing them; $A^TA$ is available

Initialize $M_A(I_p,:)$ to all zeros of size $|I_p| \times R$

foreach $i \in I_p$ do
  foreach $x_{i,j,k} \in X(I_p,:,:)$ do
    $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}B(j,:) \ast C(k,:)]$
  end
  $A(I_p,:) \leftarrow M_A(I_p,:)(B^TB \ast C^TC)^\dagger$
  foreach $i \in I_p$ do
    Send $A(i,:)$ to all processes having nonzeros in $X(I_p,:,:)$
    Locally compute $A(I_p,:)^TA(I_p,:)$ and all-reduce the results to form $A^TA$
Coarse-Grain MTTKRP within CP-ALS

- Computing any row $M_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$.
  $\Rightarrow$ Owner of $M_A(i,:)$ is responsible for computing $A(i,:) = M_A(i,:) (B^T B \ast C^T C)^\dagger$.
- $B^T B$ and $C^T C$ are available to all processes.
- Atomic tasks are partitioned to processes; process $p$ owns the set of rows $M_A(I_p,:)$. Each process $p$ has all the nonzeros in the tensor slice $X(I_p,:)$.
- For $x_{i,j,k} \in X(I_p,:)$, $B(j,:)$ and $C(k,:)$ are available to process $p$.
- Make sure at the end all the premises of the next iteration satisfy.
- How to decide on $I_p$?

**Algorithm 16:** Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p_p$ within CP-ALS

**Input:** $I_p$, indices where $\mu_A(i) = p$

- $X_{I_p,:}$: tensor slices
- $B^T B$ and $C^T C$, $R \times R$ matrices
- $B(J_p,:), C(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $X_{I_p,:}$

**On exit:** $A(I_p,:)$ is computed, its rows are sent to processes needing them; $A^T A$ is available

Initialize $M_A(I_p,:)$ to all zeros of size $|I_p| \times R$

foreach $i \in I_p$ do
  foreach $x_{i,j,k} \in X_{I_p,:}$ do
    1 $M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \ast C(k,:)]$
  2 $A(I_p,:) \leftarrow M_A(I_p,:) (B^T B \ast C^T C)^\dagger$
  foreach $i \in I_p$ do
  3 Send $A(i,:)$ to all processes having nonzeros in $X_{I_p,:}$.
  4 Locally compute $A(I_p,:)^T A(I_p,:)$ and all-reduce the results to form $A^T A$
Coarse-Grain MTTKRP within CP-ALS

- Computing any row $M_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$.
  $\Rightarrow$ Owner of $M_A(i,:)$ is responsible for computing $A(i,:)=M_A(i,:)(B^TB \times C^TC)^\dagger$.
- $B^TB$ and $C^TC$ are available to all processes.
- Atomic tasks are partitioned to processes; process $p$ owns the set of rows $M_A(I_p,:)$.
- Each process $p$ has all the nonzeros in the tensor slice $X(I_p,:,:)$. 
- For $x_{i,j,k} \in X(I_p,:,:), B(j,:) \text{ and } C(k,:)$ are available to process $p$.
- Make sure at the end all the premises of the next iteration satisfy.
- How to decide on $I_p$?

Algorithm 17: Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p_p$ within CP-ALS

Input: $I_p$, indices where $\mu_A(i) = p$
- $X(I_p,:,:)$: tensor slices
- $B$ and $C^T$, $R \times R$ matrices
- $B(J_p,:), C(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $X(I_p,:,:)$

On exit: $A(I_p,:)$ is computed, its rows are sent to processes needing them; $A^TA$ is available

Initialize $M_A(I_p,:)$ to all zeros of size $|I_p| \times R$

foreach $i \in I_p$ do
  foreach $x_{i,j,k} \in X(I_p,:,:)$ do
    $1. M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \times C(k,:)]$
  2. $A(I_p,:) \leftarrow M_A(I_p,:)(B^TB \times C^TC)^\dagger$
  foreach $i \in I_p$ do
    3. Send $A(i,:)$ to all processes having nonzeros in $X(I_p,:,:)$
  4. Locally compute $A(I_p,:)^TA(I_p,:)$ and all-reduce the results to form $A^TA$
Computing any row $\mathbf{M}_A(i,:)$ is an atomic task owned by the process $\mu_A(i)$. 
$\Rightarrow$ Owner of $\mathbf{M}_A(i,:)$ is responsible for computing $\mathbf{A}(i,:) = \mathbf{M}_A(i,:)(\mathbf{B}^T\mathbf{B} \ast \mathbf{C}^T\mathbf{C})^\dagger$.

- $\mathbf{B}^T\mathbf{B}$ and $\mathbf{C}^T\mathbf{C}$ are available to all processes.
- Atomic tasks are partitioned to processes; process $p$ owns the set of rows $\mathbf{M}_A(I_p,:)$. 
- Each process $p$ has all the nonzeros in the tensor slice $\mathcal{X}(I_p,:)$. 
- For $x_{i,j,k} \in \mathcal{X}(I_p,:), \mathbf{B}(j,:)$ and $\mathbf{C}(k,:)$ are available to process $p$.
- Make sure at the end all the premises of the next iteration satisfy.

How to decide on $I_p$?

Algorithm 18: Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor $p_p$ within CP-ALS

Input: $I_p$, indices where $\mu_A(i) = p$ 
- $\mathcal{X}(I_p,:)$: tensor slices 
- $\mathbf{B}(J_p,:)$, $\mathbf{C}(K_p,:)$, Rows of the factor matrices, where $J_p$ and $K_p$ correspond to the unique second and third mode indices in $\mathcal{X}_{I_p,:}$ 

On exit: $\mathbf{A}(I_p,:)$ is computed, its rows are sent to processes needing them; $\mathbf{A}^T\mathbf{A}$ is available 

Initialize $\mathbf{M}_A(I_p,:)$ to all zeros of size $|I_p| \times R$

foreach $i \in I_p$ do

foreach $x_{i,j,k} \in \mathcal{X}_{I_p,:}$ do

1 $\mathbf{M}_A(i,:) \leftarrow \mathbf{M}_A(i,:) + x_{i,j,k}[\mathbf{B}(j,:) \ast \mathbf{C}(k,:)]$

2 $\mathbf{A}(I_p,:) \leftarrow \mathbf{M}_A(I_p,:) (\mathbf{B}^T\mathbf{B} \ast \mathbf{C}^T\mathbf{C})^\dagger$

foreach $i \in I_p$ do

3 Send $\mathbf{A}(i,:)$ to all processes having nonzeros in $\mathcal{X}_{I_p,:}$

4 Locally compute $\mathbf{A}(I_p,:)^T\mathbf{A}(I_p,:)$ and all-reduce the results to form $\mathbf{A}^T\mathbf{A}$
Coarse-Grain MTTKRP within CP-ALS

Computing any row \( M_A(i,:) \) is an atomic task owned by the process \( \mu_A(i) \).

\[ \Rightarrow \text{Owner of } M_A(i,:) \text{ is responsible for computing } A(i :) = M_A(i,:)(B^T B \ast C^T C)^\dagger. \]

- \( B^T B \) and \( C^T C \) are available to all processes.
- Atomic tasks are partitioned to processes; process \( p \) owns the set of rows \( M_A(I_p,:) \).
- Each process \( p \) has all the nonzeros in the tensor slice \( X(I_p,:) \).
- For \( x_{i,j,k} \in X(I_p,:) \), \( B(j,:) \) and \( C(k,:) \) are available to process \( p \).
- Make sure at the end all the premises of the next iteration satisfy.
- How to decide on \( I_p \)?

\begin{algorithm}
\textbf{Algorithm 19:} Coarse-grain MTTKRP for the first mode of 3rd order tensors at processor \( p \) within CP-ALS
\begin{itemize}
\item \textbf{Input}: \( I_p \), indices where \( \mu_A(i) = p \)
\item \( X_{I_p,:}, \) tensor slices
\item \( B^T B \) and \( C^T C \), \( R \times R \) matrices
\item \( B(J_p,:), C(K_p,:) \), Rows of the factor matrices, where \( J_p \) and \( K_p \) correspond to the unique second and third mode indices in \( X_{I_p,:} \)
\item \textbf{On exit}: \( A(I_p,:) \) is computed, its rows are sent to processes needing them; \( A^T A \) is available
\end{itemize}
\begin{enumerate}
\item Initialize \( M_A(I_p,:) \) to all zeros of size \( |I_p| \times R \)
\item \textbf{foreach} \( i \in I_p \) \textbf{do}
\item \textbf{foreach} \( x_{i,j,k} \in X_{I_p,:} \) \textbf{do}
\item \[ M_A(i,:) \leftarrow M_A(i,:) + x_{i,j,k}[B(j,:) \ast C(k,:)] \]
\item \textbf{foreach} \( i \in I_p \) \textbf{do}
\item \textbf{Send} \( A(i,:) \) to all processes having nonzeros in \( X_{I_p,:} \).
\item \textbf{Locally compute} \( A(I_p,:)^T A(I_p,:) \) and all-reduce the results to form \( A^T A \)
\end{enumerate}
\end{algorithm}
Hypergraph Partitioning

- Hypergraph $H(V, E)$ is a set of vertices $V$ and hyperedges $E$ connecting the vertices.
- Hypergraph partitioning aims to partition the vertices equitably so that the cutsize is minimized.
  - $\lambda(h)$: The number of parts the hyperedge $h$ connects.
  - \[ \text{cutsize} = \sum_{h \in E} [\lambda(h) - 1] \]
Hypergraph Partitioning

- Hypergraph $H(V, E)$ is a set of vertices $V$ and hyperedges $E$ connecting the vertices.
- Hypergraph partitioning aims to partition the vertices equitably so that the cutsize is minimized.
  - $\lambda(h)$: The number of parts the hyperedge $h$ connects.
  - cutsize = $\sum_{h \in E} [\lambda(h) - 1]$
Hypergraph Model for Coarse-Grain MTTKRP

- For each atomic task $M_A(i,:)$, create a vertex with weight $|X(i,:, :)|;$ do the same for $M_B(j,:)$, and $M_C(k,:)$.
- For each vertex, add a hyperedge to model dependency.
- Nonzeros imply data dependencies; connect vertices to corresponding hyperedges.
- Balancing vertex weights $\Rightarrow$ balancing the computation.
- Minimizing the cutsize $\Rightarrow$ minimizing the total communication volume.
Example: Hypergraph Model for Coarse-Grain MTTKRP

- Example: \( \mathbf{X} \in \mathbb{R}^{1 \times 2 \times 2} \), 
  \( \mathbf{X} = \{(1, 1, 2), (1, 2, 1)\} \)
- Add dependencies for a-type vertices...
Example: Hypergraph Model for Coarse-Grain MTTKRP

- Example: $\mathbf{X} \in \mathbb{R}^{1 \times 2 \times 2}$, $\mathbf{X} = \{(1, 1, 2), (1, 2, 1)\}$
- Add dependencies for a-type vertices...
Example: $\mathbf{X} \in \mathbb{R}^{1 \times 2 \times 2}$, 
$\mathbf{X} = \{(1, 1, 2), (1, 2, 1)\}$

Add dependencies for a-type vertices...
Example: Hypergraph Model for Coarse-Grain MTTKRP

- Example: $\mathbf{X} \in \mathbb{R}^{1 \times 2 \times 2}$, $\mathbf{X} = \{(1, 1, 2), (1, 2, 1)\}$
- Add dependencies for b-type vertices...
Example: Hypergraph Model for Coarse-Grain MTTKRP

- Example: $\mathcal{X} \in \mathbb{R}^{1 \times 2 \times 2}$, $\mathcal{X} = \{(1, 1, 2), (1, 2, 1)\}$
- Add dependencies for c-type vertices...
Example: Hypergraph Model for Coarse-Grain MTTKRP

- Example: $\mathcal{X} \in \mathbb{R}^{1 \times 2 \times 2}$, $\mathcal{X} = \{(1, 1, 2), (1, 2, 1)\}$
- Consider the following partition:
  - Load balancing for each vertex type corresponds to balancing to compute $A$, $B$, and $C$.
  - For a vertex with hyperedge $n$, $\lambda(n) - 1$ is the exact number of sends needed.
Outline

1. Introduction
2. CP Decomposition
3. Coarse-Grain Parallelization
4. Fine-Grain Parallelization
5. Experiments and Results
6. Conclusion and Future Work
Tensor nonzeros are partitioned; \( \mu_X(i,j,k) = p \) if \( x_{i,j,k} \in X_p \).

Performing the multiplication \( x_{i,j,k}[B(j,:) \ast C(k,:)] \) and generating 'local sum' is an atomic task.

Process \( p \) owns the set of rows \( A(I_p,:) \).

Assumed availability of \( B^TB \) and \( C^TC \), and needed rows of \( B \) and \( C \).

Partial results \( M_A(F_p \setminus I_p,:) \) are sent/received (fold)

Rows of \( A(I_p,:) \) and \( A(F_p \setminus I_p,:) \) are sent/received to/from other processes. (expand)
Hypergraph Model for Fine-Grain MTTKRP

- For $M_A(i,:)$, $M_B(j,:)$, and $M_C(k,:)$, create vertices $a_i$, $b_j$, and $c_k$ with unit weight, and their hyperedges.
- For each nonzero $x_{i,j,k}$, create a vertex $x_{i,j,k}$ of unit weight, and connect to the hyperedges of $a_i$, $b_j$, and $c_k$.
- Perform multi-constraint hypergraph partitioning on $a$, $b$, $c$, and $x$-type vertices.
  - Balancing $x$-type vertices $\Rightarrow$ Balancing computational workload.
  - Balancing $a$, $b$, $c$-type vertices $\Rightarrow$ Balancing memory usage.
- Minimize the cutsize $\Rightarrow$ Minimize the total communication volume.
- **Advantage:** Much more flexibility to partition the tasks of finer granularity
- **Disadvantage:** Two communication phases needed
Example: \( \mathcal{X} \in \mathbb{R}^{3 \times 3 \times 3} \),
\( \mathcal{X} = \{(1, 2, 3), (2, 3, 1), (3, 1, 2)\} \)

Consider the following partition:

- Load balancing \( x \)-type vectices correspond to balancing the computational load.
- Load balancing \( a, b, \) and \( c \)-type vectices correspond to balancing the memory requirements.
- For a vertex with hyperedge \( n \), \( \lambda(n) - 1 \) is the exact number of sends needed.
Outline

1. Introduction
2. CP Decomposition
3. Coarse-Grain Parallelization
4. Fine-Grain Parallelization
5. Experiments and Results
6. Conclusion and Future Work
Dataset

Our dataset consists of two 3-dimensional, and two 4-dimensional tensors.

- **Netflix**: user×review×time tensor formed from Netflix user movie reviews.
- **NELL-B**: entity×relation×entity tensor formed from the NELL knowledge database entries (e.g., 'Chopin plays piano').
- **Delicious and Flickr**: user×resource×tag×time tensors formed from the web-crawl of Delicious.com and Flickr.com during 2006 and 2007, by Görlitz et al.

<table>
<thead>
<tr>
<th>Tensor</th>
<th>$l_1$</th>
<th>$l_2$</th>
<th>$l_3$</th>
<th>$l_4$</th>
<th>#nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>480K</td>
<td>17K</td>
<td>2K</td>
<td>-</td>
<td>100M</td>
</tr>
<tr>
<td>NELL-B</td>
<td>3.2M</td>
<td>301</td>
<td>638K</td>
<td>-</td>
<td>78M</td>
</tr>
<tr>
<td>Delicious</td>
<td>1K</td>
<td>530K</td>
<td>17M</td>
<td>2.4M</td>
<td>140M</td>
</tr>
<tr>
<td>Flickr</td>
<td>713</td>
<td>319K</td>
<td>28M</td>
<td>1.6M</td>
<td>112M</td>
</tr>
</tbody>
</table>
Results - Runtime (Netflix, NELL-B)

- Fine-grain kernel with hypergraph partitioning achieves 194x/81x speedup on Netflix/NELL-B.
- The best of other methods yields to 69x/39x on Netflix/NELL-B.
- Hypergraph partitioning for the coarse-grain kernel provides only a slight edge over block partitioning.
Results - Speedup (Flickr, Delicious)

- 129x/123x speedup using fine-grain kernel using hypergraph partitioning on Flickr/Delicious
- 65x/55x speedup for the best of others
- No scalability beyond 1024 MPI ranks.
Communication Analysis

- Fine-grain hypergraph partitioning
  ⇒ \(20\times\) less communication
  ⇒ \(5\times\) more speedup
- Reduction in the communication volume is not impressive for the coarse-grain kernel.
- Communication latency is the major bottleneck to be addressed.

Table: Statistics for the computation and communication requirements in one CP-ALS iteration for 512-way partitionings of the Netflix tensor

<table>
<thead>
<tr>
<th>Mode</th>
<th>Comp. load</th>
<th>Comm. volume</th>
<th>Num. msg.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>ht-finegrain-hp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>196672</td>
<td>196251</td>
<td>21079</td>
</tr>
<tr>
<td>2</td>
<td>196672</td>
<td>196251</td>
<td>18028</td>
</tr>
<tr>
<td>3</td>
<td>196672</td>
<td>196251</td>
<td>3545</td>
</tr>
<tr>
<td>ht-finegrain-random</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>197507</td>
<td>196251</td>
<td>272326</td>
</tr>
<tr>
<td>2</td>
<td>197507</td>
<td>196251</td>
<td>29282</td>
</tr>
<tr>
<td>3</td>
<td>197507</td>
<td>196251</td>
<td>7766</td>
</tr>
<tr>
<td>ht-coarsegrain-hp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>364181</td>
<td>196251</td>
<td>302001</td>
</tr>
<tr>
<td>2</td>
<td>349123</td>
<td>196251</td>
<td>59523</td>
</tr>
<tr>
<td>3</td>
<td>737570</td>
<td>196251</td>
<td>23524</td>
</tr>
<tr>
<td>ht-coarsegrain-block</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>198602</td>
<td>196251</td>
<td>239337</td>
</tr>
<tr>
<td>2</td>
<td>367966</td>
<td>196251</td>
<td>33889</td>
</tr>
<tr>
<td>3</td>
<td>737570</td>
<td>196251</td>
<td>24659</td>
</tr>
</tbody>
</table>
We have provided first parallel MTTKRP with P2P communication, as well as hypergraph models of computation.

We achieved significantly better speedup than the state-of-the-art.

For further speedup we plan to investigate:

- Shared-memory parallelism
- Methods to control communication latency

We have utilized the following computational resources:

- **(C/G)runch**: Code development, testing, and initial performance tests
- **PSMN**: Initial experiments and MPI optimizations
- **IDRIS Ada**: Final experiments
Thank you for your time.

Questions/Comments?