High Performance Parallel Tucker Decomposition of Sparse Tensors

Oguz Kaya

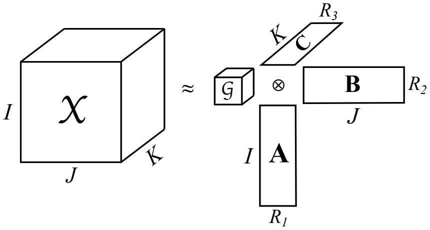
INRIA and LIP, ENS Lyon, France

SIAM PP'16, April 14, 2016, Paris, France

Joint work with:

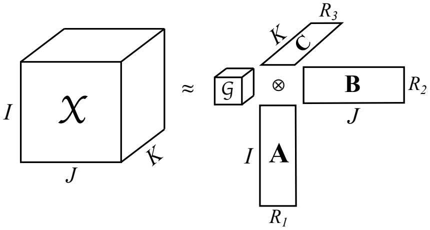
Bora Uçar, CNRS and LIP, ENS Lyon, France

Tucker Tensor Decomposition



- Tucker decomposition
  - provides a rank-$(R_1, \ldots, R_N)$ approximation of a tensor.
  - consists of a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and N matrices having $R_1, \ldots, R_N$ columns.
- We are interested in the case when $\mathcal{X}$ is **big**, **sparse**, and is of **low rank**.
  - Example: Google web queries, Netflix movie ratings, Amazon product reviews, etc.

Tucker Tensor Decomposition



- Tucker decomposition
  - provides a rank-$(R_1, \ldots, R_N)$ approximation of a tensor.
  - consists of a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and N matrices having $R_1, \ldots, R_N$ columns.
- We are interested in the case when $\mathcal{X}$ is **big**, **sparse**, and is of **low rank**.
  - Example: Google web queries, Netflix movie ratings, Amazon product reviews, etc.

Tucker Tensor Decomposition

- Related Work:
  - Matlab Tensor Toolbox by Kolda et al.
  - Efficient and scalable computations with sparse tensors (Baskaran et al., '12)
  - Parallel Tensor Compression for Large-Scale Scientific Data (Austin et al., '15)
  - Haten2: Billion-scale tensor decompositions (Jung et al., '15)
- Applications (in data mining):
  - CubeSVD: A Novel Approach to Personalized Web Search (Sun et al., '05)
  - Tag Recommendations Based on Tensor Dimensionality Reduction (Symeonidis et al., '08)
  - Extended feature combination model for recommendations in location-based mobile services (Sattari et al. '15)
- Goal: To compute sparse Tucker decomposition in parallel (shared/distributed memory).

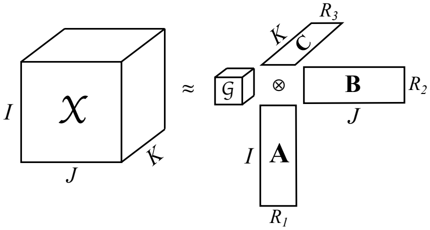
Tucker Tensor Decomposition

- Related Work:
  - Matlab Tensor Toolbox by Kolda et al.
  - Efficient and scalable computations with sparse tensors (Baskaran et al., '12)
  - Parallel Tensor Compression for Large-Scale Scientific Data (Austin et al., '15)
  - Haten2: Billion-scale tensor decompositions (Jung et al., '15)
- Applications (in data mining):
  - CubeSVD: A Novel Approach to Personalized Web Search (Sun et al., '05)
  - Tag Recommendations Based on Tensor Dimensionality Reduction (Symeonidis et al., '08)
  - Extended feature combination model for recommendations in location-based mobile services (Sattari et al. '15)
- Goal: To compute sparse Tucker decomposition in parallel (shared/distributed memory).

Tucker Tensor Decomposition

- Related Work:
  - Matlab Tensor Toolbox by Kolda et al.
  - Efficient and scalable computations with sparse tensors (Baskaran et al., '12)
  - Parallel Tensor Compression for Large-Scale Scientific Data (Austin et al., '15)
  - Haten2: Billion-scale tensor decompositions (Jung et al., '15)
- Applications (in data mining):
  - CubeSVD: A Novel Approach to Personalized Web Search (Sun et al., '05)
  - Tag Recommendations Based on Tensor Dimensionality Reduction (Symeonidis et al., '08)
  - Extended feature combination model for recommendations in location-based mobile services (Sattari et al. '15)
- Goal: To compute sparse Tucker decomposition in parallel (shared/distributed memory).

Higher Order Orthogonal Iteration (HOOI) Algorithm



Algorithm: HOOI for 3rd order tensors

repeat

- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$
- $\mathbf{A} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R_1)$ // R_1 *leading left singular vectors*
- $\hat{\mathbf{B}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_3 \mathbf{C}]_{(2)}$
- $\mathbf{B} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{B}}, R_2)$
- $\hat{\mathbf{C}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B}]_{(2)}$
- $\mathbf{C} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{C}}, R_3)$

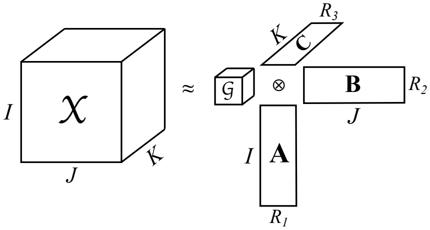
until *no more improvement or maximum iterations reached*

$\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$

return $[\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}]$

- We discuss the case where $R_1 = R_2 = \cdots = R_N = R$ and $N = 3$.
- $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are dense.
- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \in \mathbb{R}^{I \times R^2}$ is called tensor-times-matrix multiply (TTM).
- $\hat{\mathbf{A}} \in \mathbb{R}^{I \times R^{N-1}}$, $\hat{\mathbf{B}} \in \mathbb{R}^{J \times R^{N-1}}$, and $\hat{\mathbf{C}} \in \mathbb{R}^{K \times R^{N-1}}$ are dense. (R^2 columns for $N = 3$)

Higher Order Orthogonal Iteration (HOOI) Algorithm



Algorithm: HOOI for 3rd order tensors

repeat

- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$
- $\mathbf{A} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R_1)$ $//R_1$ *leading left singular vectors*
- $\hat{\mathbf{B}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_3 \mathbf{C}]_{(2)}$
- $\mathbf{B} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{B}}, R_2)$
- $\hat{\mathbf{C}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B}]_{(2)}$
- $\mathbf{C} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{C}}, R_3)$

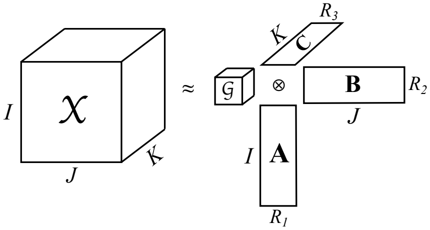
until *no more improvement or maximum iterations reached*

$\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$

return $[\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}]$

- We discuss the case where $R_1 = R_2 = \cdots = R_N = R$ and $N = 3$.
- $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are dense.
- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \in \mathbb{R}^{I \times R^2}$ is called tensor-times-matrix multiply (TTM).
- $\hat{\mathbf{A}} \in \mathbb{R}^{I \times R^{N-1}}$, $\hat{\mathbf{B}} \in \mathbb{R}^{J \times R^{N-1}}$, and $\hat{\mathbf{C}} \in \mathbb{R}^{K \times R^{N-1}}$ are dense. (R^2 columns for $N = 3$)

Higher Order Orthogonal Iteration (HOOI) Algorithm



Algorithm: HOOI for 3rd order tensors

repeat

- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$
- $\mathbf{A} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R_1)$ // R_1 *leading left singular vectors*
- $\hat{\mathbf{B}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_3 \mathbf{C}]_{(2)}$
- $\mathbf{B} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{B}}, R_2)$
- $\hat{\mathbf{C}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B}]_{(2)}$
- $\mathbf{C} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{C}}, R_3)$

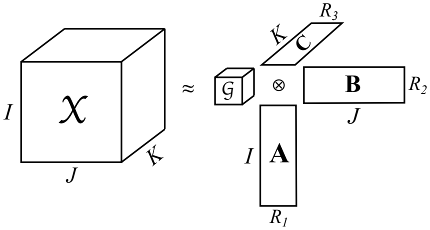
until *no more improvement or maximum iterations reached*

$\mathcal{G} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$

return $[\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]$

- We discuss the case where $R_1 = R_2 = \cdots = R_N = R$ and $N = 3$.
- $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are dense.
- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \in \mathbb{R}^{I \times R^2}$ is called tensor-times-matrix multiply (TTM).
- $\hat{\mathbf{A}} \in \mathbb{R}^{I \times R^{N-1}}$, $\hat{\mathbf{B}} \in \mathbb{R}^{J \times R^{N-1}}$, and $\hat{\mathbf{C}} \in \mathbb{R}^{K \times R^{N-1}}$ are dense. (R^2 columns for $N = 3$)

Higher Order Orthogonal Iteration (HOOI) Algorithm



Algorithm: HOOI for 3rd order tensors

repeat

$\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$

$\mathbf{A} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R_1)$ // R_1 *leading left singular vectors*

$\hat{\mathbf{B}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_3 \mathbf{C}]_{(2)}$

$\mathbf{B} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{B}}, R_2)$

$\hat{\mathbf{C}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B}]_{(2)}$

$\mathbf{C} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{C}}, R_3)$

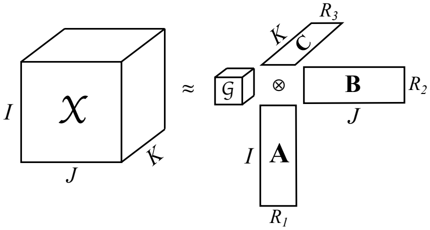
until *no more improvement or maximum iterations reached*

$\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$

return $[\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}]$

- We discuss the case where $R_1 = R_2 = \cdots = R_N = R$ and $N = 3$.
- $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are dense.
- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \in \mathbb{R}^{I \times R^2}$ is called tensor-times-matrix multiply (TTM).
- $\hat{\mathbf{A}} \in \mathbb{R}^{I \times R^{N-1}}$, $\hat{\mathbf{B}} \in \mathbb{R}^{J \times R^{N-1}}$, and $\hat{\mathbf{C}} \in \mathbb{R}^{K \times R^{N-1}}$ are dense. (R^2 columns for $N = 3$)

Higher Order Orthogonal Iteration (HOOI) Algorithm



Algorithm: HOOI for 3rd order tensors

repeat
- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$
- $\mathbf{A} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R_1)$ // R_1 *leading left singular vectors*
- $\hat{\mathbf{B}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_3 \mathbf{C}]_{(2)}$
- $\mathbf{B} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{B}}, R_2)$
- $\hat{\mathbf{C}} \leftarrow [\boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B}]_{(2)}$
- $\mathbf{C} \leftarrow \mathrm{TRSVD}(\hat{\mathbf{C}}, R_3)$

until *no more improvement or maximum iterations reached*

$\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$

return $[\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}]$

- We discuss the case where $R_1 = R_2 = \cdots = R_N = R$ and $N = 3$.
- $\mathbf{A} \in \mathbb{R}^{I \times R}$,$\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are dense.
- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \in \mathbb{R}^{I \times R^2}$ is called tensor-times-matrix multiply (TTM).
- $\hat{\mathbf{A}} \in \mathbb{R}^{I \times R^{N-1}}$,$\hat{\mathbf{B}} \in \mathbb{R}^{J \times R^{N-1}}$, and $\hat{\mathbf{C}} \in \mathbb{R}^{K \times R^{N-1}}$ are dense. (R^2 columns for $N = 3$)

Tensor-Times-Matrix Multiply

- $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$, $\hat{\mathbf{A}} \in \mathbb{R}^{I \times R^2}$
- $\mathbf{B}(j,:) \otimes \mathbf{C}(k,:) \in \mathbb{R}^{R^2}$ is a Kronecker product.
- For each nonzero $x_{i,j,k}$;
  $\hat{\mathbf{A}}(i,:)$ receives the update $x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$.

Algorithm: $\hat{\mathbf{A}} \leftarrow [\boldsymbol{\mathcal{X}} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)}$

1 $\hat{\mathbf{A}} \leftarrow zeros(I, R^2)$
foreach $x_{i,j,k} \in \boldsymbol{\mathcal{X}}$ **do**
2 $\quad \hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$

Outline

(Bad) Fine-Grain Parallel TTM within Tucker-ALS

- $\mathbf{A}$ and $\hat{\mathbf{A}}$ are rowwise distributed
  - Process p owns and computes $\mathbf{A}(I_p,:)$ and $\hat{\mathbf{A}}(I_p,:)$.
- Tensor nonzeros are partitioned (arbitrarily)
  - Process p owns the subset of nonzeros $\boldsymbol{\mathcal{X}}_p$
  - Performing $x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$ and generating a partial result for $\hat{\mathbf{A}}(i,:)$ is a fine-grain task
  - We use post-communication scheme at each iteration:
    $\rightarrow$ at the beginning, rows of **A**, **B**, and **C** are available.
    $\rightarrow$ at the end, only **A** is updated and communicated.
- Partial row results of $\hat{\mathbf{A}}$ are sent/received (**fold**).
- Rows of $\mathbf{A}(I_p,:)$ are sent/received (**expand**).

Algorithm: Computing **A** in fine-grain HOOI at process p

foreach $x_{i,j,k} \in \boldsymbol{\mathcal{X}}_p$ **do**
    $\hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$
Send/Receive and sum up "partial" rows of $\hat{\mathbf{A}}$
$\mathbf{A}(I_p,:) \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R)$
Send/Receive rows of **A**

(Bad) Fine-Grain Parallel TTM within Tucker-ALS

- Number of rows sent/received in fold/expand are equal.
  - Each communication unit of **expand** has size R.
  - Each communication unit of **fold** has size R^{N-1}.
- We want to avoid *assembling* $\hat{\mathbf{A}}$ in **fold** communication.
- We need to compute TRSVD($\hat{\mathbf{A}}$, R).

Algorithm: Computing **A** in fine-grain HOOI at process p

foreach $x_{i,j,k} \in \boldsymbol{\mathcal{X}}_p$ **do**
  $\hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$
Send/Receive and sum up "partial" rows of $\hat{\mathbf{A}}$
$\mathbf{A}(I_p,:) \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R)$
Send/Receive rows of **A**

(Bad) Fine-Grain Parallel TTM within Tucker-ALS

- Number of rows sent/received in fold/expand are equal.
  - Each communication unit of **expand** has size R.
  - Each communication unit of **fold** has size R^{N-1}.
- We want to avoid *assembling* $\hat{\mathbf{A}}$ in **fold** communication.
- We need to compute $\mathrm{TRSVD}(\hat{\mathbf{A}}, R)$.

Algorithm: Computing $\mathbf{A}$ in fine-grain HOOI at process p

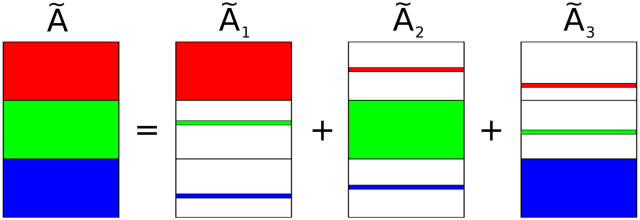
foreach $x_{i,j,k} \in \boldsymbol{\mathcal{X}}_p$ **do**
  $\hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$

~~Send/Receive and sum up "partial" rows of $\hat{\mathbf{A}}$~~

$\mathbf{A}(I_p,:) \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R)$

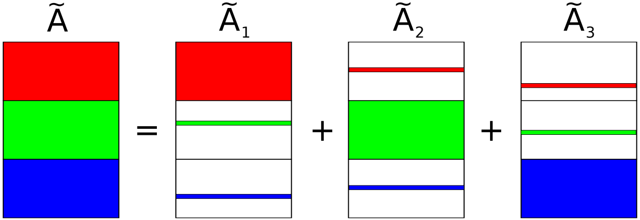
Send/Receive rows of $\mathbf{A}$

Computing TRSVD



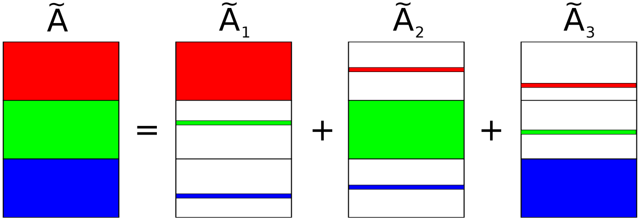
- Gram matrix $\hat{\mathbf{A}}\hat{\mathbf{A}}^T$?
- Iterative solvers?
  - Need to perform $\hat{\mathbf{A}}x$ and $\hat{\mathbf{A}}^T x$ efficiently.

Computing TRSVD



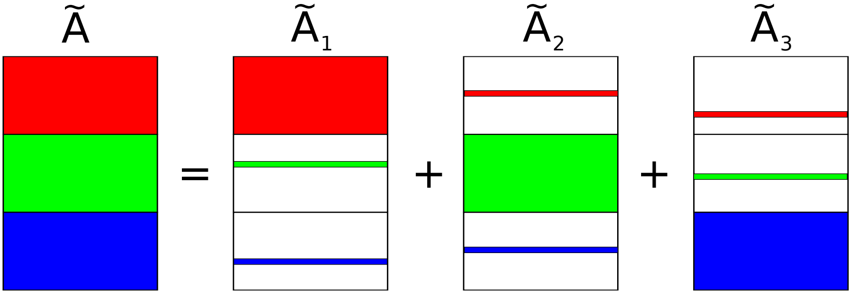
- Gram matrix $\hat{\mathbf{A}}\hat{\mathbf{A}}^T$?
- Iterative solvers?
  - Need to perform $\hat{\mathbf{A}}x$ and $\hat{\mathbf{A}}^T x$ efficiently.

Computing TRSVD

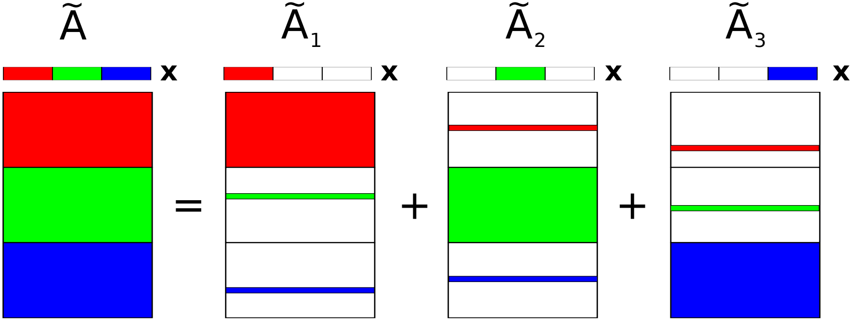


- Gram matrix $\hat{\mathbf{A}}\hat{\mathbf{A}}^T$?
- Iterative solvers?
  - Need to perform $\hat{\mathbf{A}}x$ and $\hat{\mathbf{A}}^T x$ efficiently.

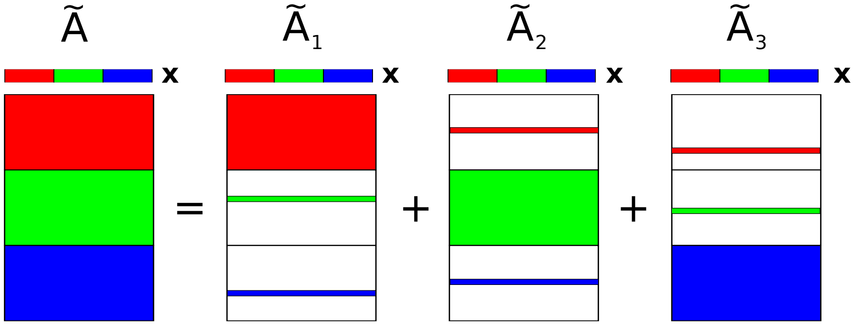
Computing $y \leftarrow \hat{\mathbf{A}}x$



Computing $y \leftarrow \hat{\mathbf{A}}x$

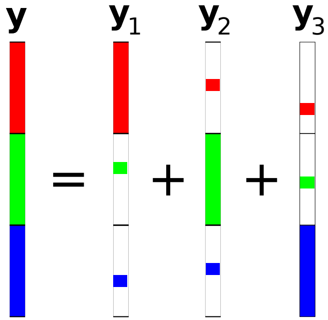


Computing $y \leftarrow \hat{\mathbf{A}}x$



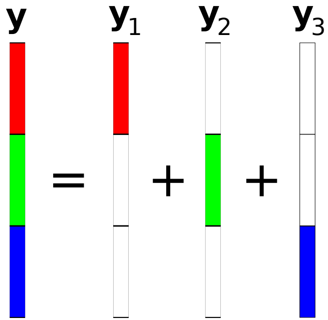
- For each unit of communication, we perform extra work in MxV.

Computing $y \leftarrow \hat{\mathbf{A}}x$



- Instead of communicating R^{N-1} entries, we communicate 1! (per SVD iteration)

Computing $y \leftarrow \hat{\mathbf{A}}x$



- $y \leftarrow \hat{\mathbf{A}}^T x$ works in reverse with the same communication cost.
- Row distribution of $\mathbf{y}$ and left-singular vectors are the same as $\hat{\mathbf{A}}$
  - $\mathbf{A}$ gets the same row distribution as $\hat{\mathbf{A}}$.

(Good) Fine-Grain Parallel TTM within Tucker-ALS

Algorithm: Computing **A** in fine-grain HOOI at process p

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

  $\hat{\mathbf{A}}(i,:) \leftarrow \hat{\mathbf{A}}(i,:) + x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$

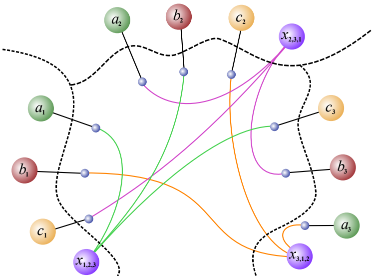
$\mathbf{A}(I_p,:) \leftarrow \mathrm{TRSVD}(\hat{\mathbf{A}}, R, \text{MxV}(\dots), \text{MTxV}(\dots))$

Send/Receive rows of **A**

Hypergraph Model for Parallel HOOI

- Multi-constraint hypergraph partitioning
  - We balance computation and memory costs.
- By minimizing the **cutsize** of the hypergraph, we minimize:
  - the total communication volume of MtV/MTxV,
  - the total extra MxV/MTxV work,
  - and the total volume of communication for TTM.
- Ideally, should minimize the **maximum**, not **total**

$$\mathcal{X} = \{(1,2,3),(2,3,1),(3,1,2)\}$$



Outline

Experimental Setup

- HyperTensor
  - Hybrid OpenMP/MPI code in C++
  - Dependencies to BLAS, LAPACK, and C++11 STL
  - SLEPc/PETSc for distributed memory TRSVD computations
- IBM BlueGene/Q Machine
  - 16GB memory and 16 cores (at 1.6GHz) per node
  - Experiments using up to 4096 cores (256 nodes)
- R_i is set to 5/10 for 4/3-dimensional tensors.

Tensor sizes

Tensor	I_1	I_2	I_3	I_4	#nonzeros
Netflix	480K	17K	2K	-	100M
NELL	3.2M	301	638K	-	78M
Delicious	1K	530K	17M	2.4M	140M
Flickr	713	319K	28M	1.6M	112M

Results - Flickr/Delicious

Per iteration runtime of the parallel HOOI (in seconds)

#nodes×#cores	Delicious				Flickr			
	fine-hp	fine-rd	coarse-hp	coarse-bl	fine-hp	fine-rd	coarse-hp	coarse-bl
1 × 16	-	-	-	-	-	-	-	-
2 × 16	-	-	-	-	-	-	-	-
4 × 16	-	-	-	-	-	-	-	-
8 × 16	164.9	-	235.3	400.5	206.2	-	287.5	308.5
16 × 16	85.2	162.0	197.5	302.4	115.6	221.8	210.5	230.1
32 × 16	47.6	96.2	155.6	206.5	64.6	124.5	166.3	190.1
64 × 16	27.2	57.8	98.9	159.6	36.8	69.9	124.1	129.0
128 × 16	18.2	34.7	80.8	96.4	22.6	42.9	87.9	102.3
256 × 16	12.2	22.1	65.1	77.1	20.0	29.2	73.8	86.3

- Coarse-grain kernel is slow due to load imbalance and communication.
- On Delicious, fine-hp is **1.8x/5.4x/6.4x** faster than fine-rd/coarse-hp/coarse-bl.
- On Flickr, fine-hp is **1.5x/3.7x/4.3x** faster than fine-rd/coarse-hp/coarse-bl.
- All instances achieve scalability to 4096 cores.

Results - NELL/Netflix

Per iteration runtime of the parallel HOOI (in seconds)

#nodes×#cores	NELL				Netflix			
	fine-hp	fine-rd	coarse-hp	coarse-bl	fine-hp	fine-rd	coarse-hp	coarse-bl
1×16	222.1	222.1	240.1	240.1	-	-	-	-
2×16	151.6	137.6	198.5	164.4	-	-	-	-
4×16	87.7	75.9	180.6	131.4	33.7	39.2	46.0	42.8
8×16	67.8	46.9	172.5	109.7	18.6	26.1	30.6	33.4
16×16	54.9	28.3	112.4	94.1	10.3	18.3	32.2	27.8
32×16	43.9	17.2	73.8	68.2	5.7	13.9	26.2	26.7
64×16	35.4	11.9	67.1	54.5	3.9	10.9	26.2	21.7
128×16	26.7	8.4	50.3	48.5	2.9	8.7	19.8	18.7
256×16	14.8	7.7	48.1	44.9	3.8	8.3	14.7	16.1

- On Netflix, fine-hp is **2.8x/5x/5.5x** faster than fine-rd/coarse-hp/coarse-bl.
- On NELL, fine-rd is faster than fine-hp (**5x less** total comm. but **2x more** max comm.)

Outline

Conclusion

- We provide
  - **the first** high performance shared/distributed memory **parallel algorithm**/implementation for the **Tucker decomposition of sparse tensors**
  - **hypergraph partitioning models** of these computations for better scalability.
- We achieve scalability **up to 4096 cores** even with random partitioning.
- We enable Tucker-based tensor analysis of very big sparse data.

Conclusion

- We provide
  - **the first** high performance shared/distributed memory **parallel algorithm**/implementation for the **Tucker decomposition of sparse tensors**
  - **hypergraph partitioning models** of these computations for better scalability.
- We achieve scalability **up to 4096 cores** even with random partitioning.
- We enable Tucker-based tensor analysis of very big sparse data.

Conclusion

- We provide
  - **the first** high performance shared/distributed memory **parallel algorithm**/implementation for the **Tucker decomposition of sparse tensors**
  - **hypergraph partitioning models** of these computations for better scalability.
- We achieve scalability **up to 4096 cores** even with random partitioning.
- We enable Tucker-based tensor analysis of very big sparse data.

References

Contact

oguz.kaya@ens-lyon.fr
www.oguzkaya.com

bora.ucar@ens-lyon.fr
perso.ens-lyon.fr/bora.ucar