Sparse Tensor Factorization: Algorithms, Data Structures, and Challenges

George Karypis and Shaden Smith

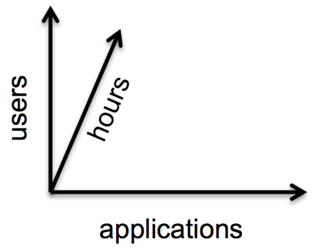
University of Minnesota
Department of Computer Science & Engineering

Outline

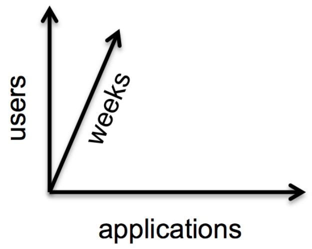
Table of Contents

Tensors

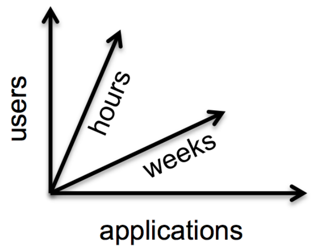
- Tensors are the generalization of matrices to higher dimensions.
- Allow us to represent and analyze multi-dimensional data (*multi-way data analysis*).



Application usage in the course of a day.



Application usage in the course of the weeks.



Application usage in the course of a day across the weeks.

Notation

- The number of dimensions (or *modes*) is m.
- The size of a 3-mode tensor will be $I \times J \times K$.

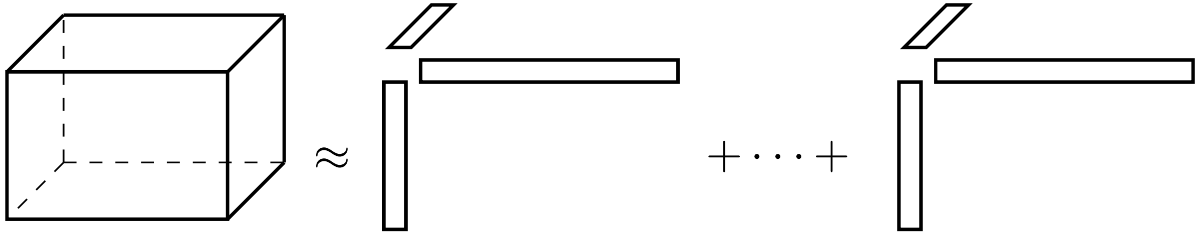
Tensor storage – coordinate form

Each non-zero is a tuple of *indices* and a *value*.

1	1	1	2	1.0
1	1	1	3	2.0
1	2	1	1	3.0
1	2	1	3	4.0
1	2	2	1	5.0
2	2	2	1	6.0
2	2	2	2	7.0
2	2	2	3	8.0

Canonical polyadic decomposition (CPD)

- The CPD models a tensor as a summation of rank-1 tensors.
  - A rank-1 tensor is the outer product of m vectors.



$$x_{ijk} \approx \sum_{f=1}^{F} a_{if} b_{jf} c_{kf} \text{ for } i = 1, \dots, I, j = 1, \dots, J, k = 1, \dots, K$$

Notation

- $\mathbf{A}, \mathbf{B}, \mathbf{C}$, each with F columns, will be used to denote the factor matrices for a 3-mode tensor.
- $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(m)}$ will be used for $m > 3$.

Applications

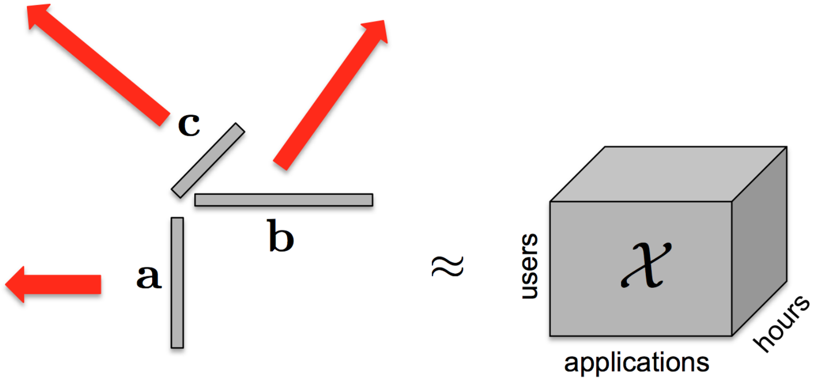
Tensor factorization has emerged as a popular tool in several data-intensive fields:

- Context-aware recommender systems: top-N recommendation and rating prediction.
- Precision healthcare: electronic health record analysis.
- Cybersecurity: intrusion detection.

Interpretation of decompositions

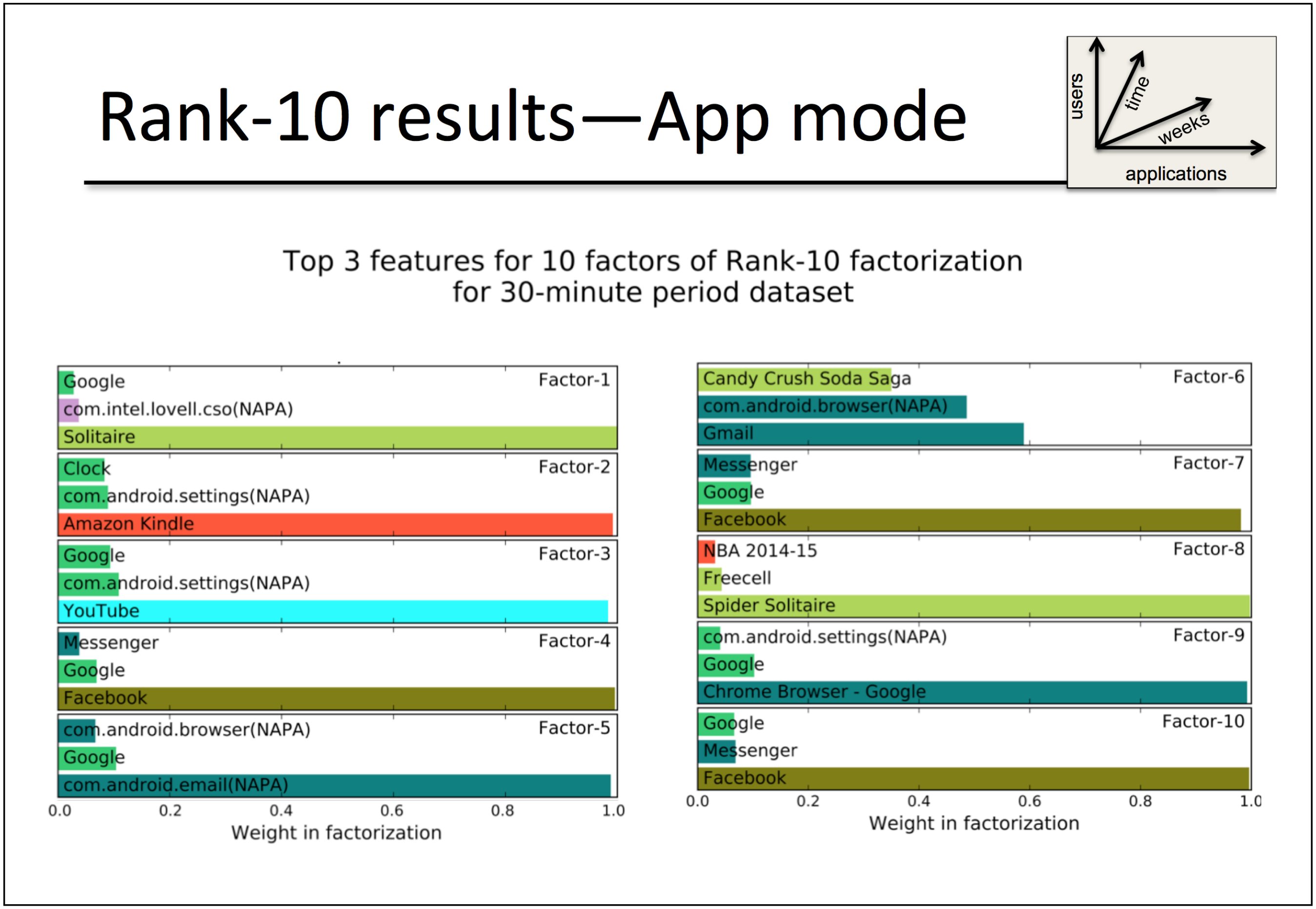
The values indicate the relative amount of usage during the different times of the day.
The hours during which the tablets are used the more, the higher their corresponding values will be.

The values indicate the relative amount of usage of the different applications.
The more time users spend on an application, the higher its value will be.



The values indicate the relative amount of usage of the different users.
The more time a user spends on his/her tablet, the higher his/her value will be.

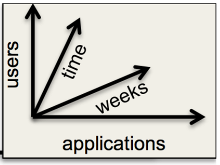
Example–1

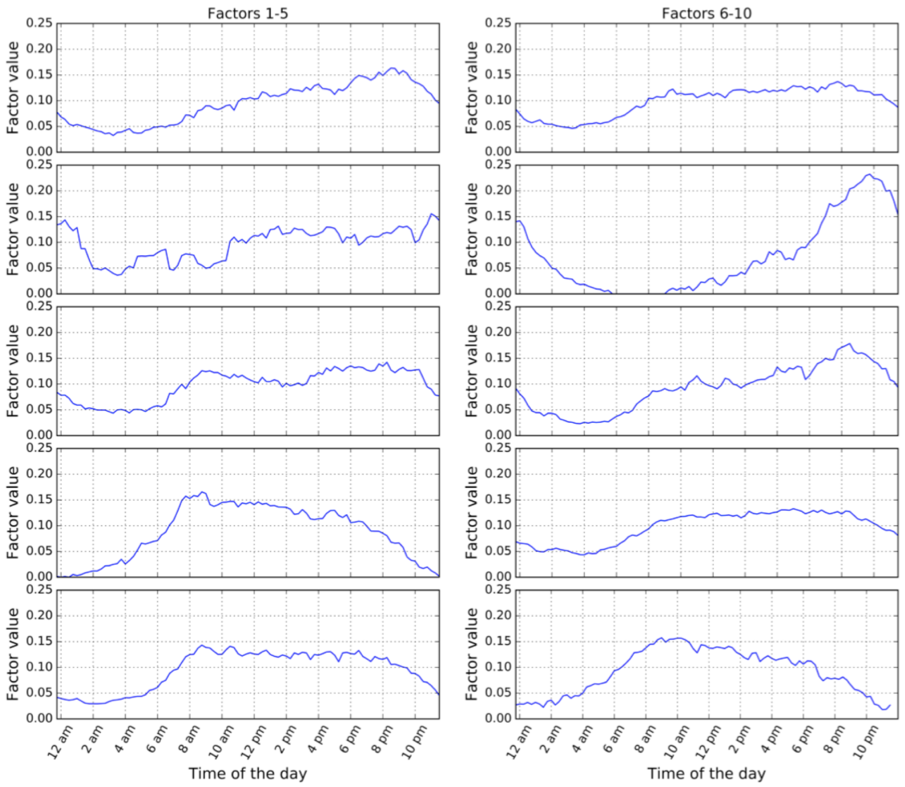


Rank-10 results—App mode

Example–2

Rank-10 results—Time mode





Solitaire

Kindle

YouTube

Facebook

Email

Gmail,
Browser
Candy Crush

Facebook

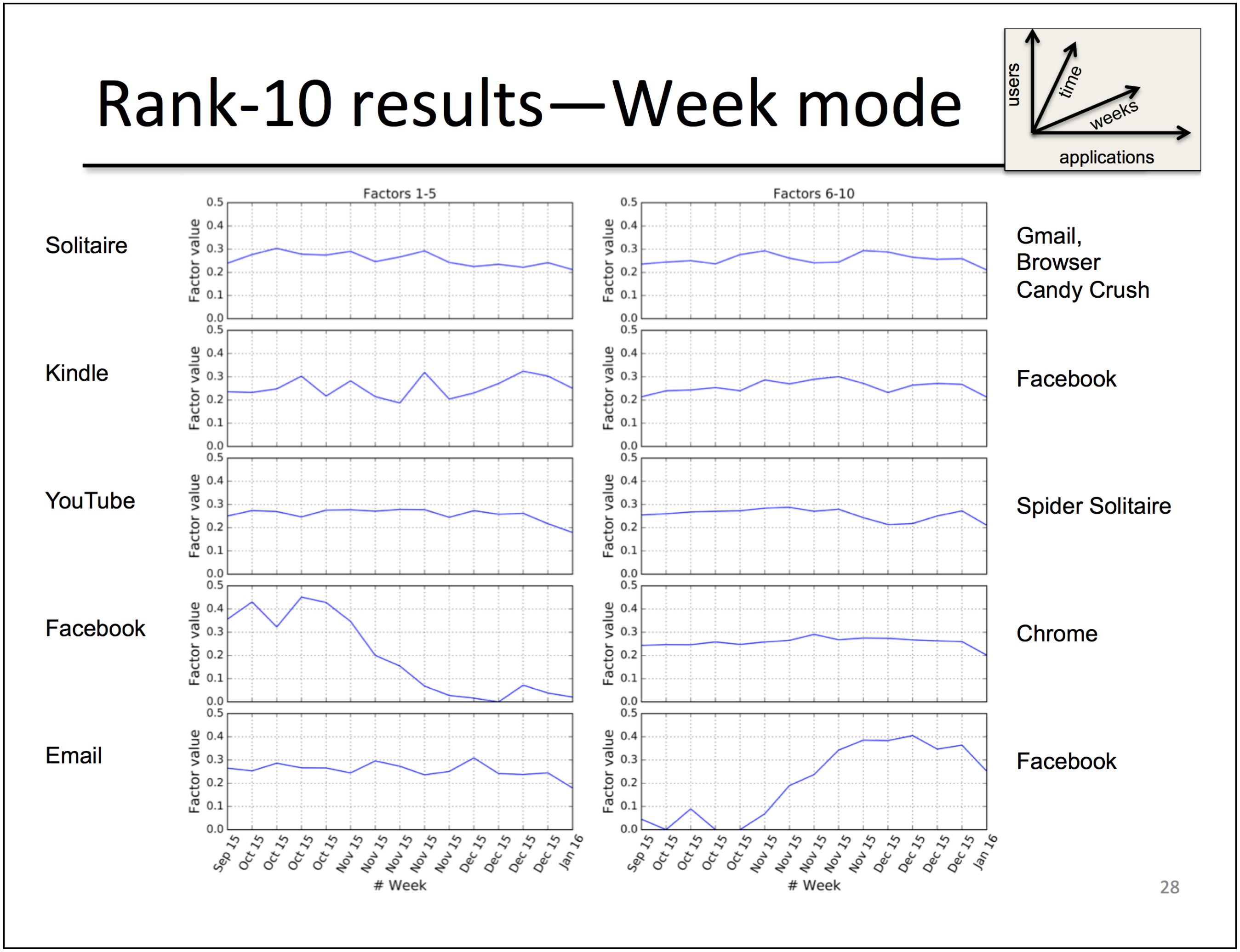
Spider Solitaire

Chrome

Facebook

27

Example–3



Rank-10 results—Week mode

Datasets used

Dataset	I	J	K	nnz
NELL-2	12K	9K	28K	77M
Beer	33K	66K	960K	94M
Netflix	480K	18K	2K	100M
Delicious	532K	17M	3M	140M
NELL-1	3M	2M	25M	143M
Yahoo	1M	625k	133	210M
Random-1	20M	20M	20M	1.0B
Random-2	50M	5M	5M	1.0B
Amazon	5M	18M	2M	1.7B

- NELL tensors are made of *noun-verb-noun* triplets.
- Beer and Amazon are *user-item-word* product reviews.
- Netflix and Yahoo are *user-item-time* product ratings.
- Delicious is made of *user-item-tag* triplets.
- Random tensors are synthetic, uniformly distributed triplets.

CPD – loss functions

The objective is a combination of the factorization quality (the *loss*) and regularization terms (to prevent overfitting).

$$\underset{\mathbf{A},\mathbf{B},\mathbf{C}}{\text{minimize}} \quad \underbrace{\mathcal{L}(\boldsymbol{\mathcal{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C})}_{\text{Loss}} + \underbrace{\lambda \left(||\mathbf{A}||_F^2 + ||\mathbf{B}||_F^2 + ||\mathbf{C}||_F^2 \right)}_{\text{Regularization}}$$

CPD – loss functions

The objective is a combination of the factorization quality (the *loss*) and regularization terms (to prevent overfitting).

$$\underset{\mathbf{A},\mathbf{B},\mathbf{C}}{\text{minimize}} \qquad \underbrace{\mathcal{L}(\boldsymbol{\mathcal{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C})}_{\text{Loss}} + \underbrace{\lambda \left(||\mathbf{A}||_F^2 + ||\mathbf{B}||_F^2 + ||\mathbf{C}||_F^2 \right)}_{\text{Regularization}}$$

Two loss functions covered today:

1. Least squares:

$$\mathcal{L}(\boldsymbol{\mathcal{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - \sum_{f=1}^{F} \left(\mathbf{A}(:, f) \circ \mathbf{B}(:, f) \circ \mathbf{C}(:, f) \right) \right\|_F^2$$

2. Missing values:

$$\mathcal{L}(\boldsymbol{\mathcal{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \sum_{\text{nnz}(\boldsymbol{\mathcal{R}})} \left(\boldsymbol{\mathcal{X}}(i, j, k) - \sum_{f=1}^{F} \mathbf{A}(i, f) \mathbf{B}(j, f) \mathbf{C}(k, f) \right)^2$$

Alternating least squares (ALS)

- The case of least-squares loss is usually computed using an alternating approach.
- ALS cyclically updates one factor matrix at a time while holding all others constant.

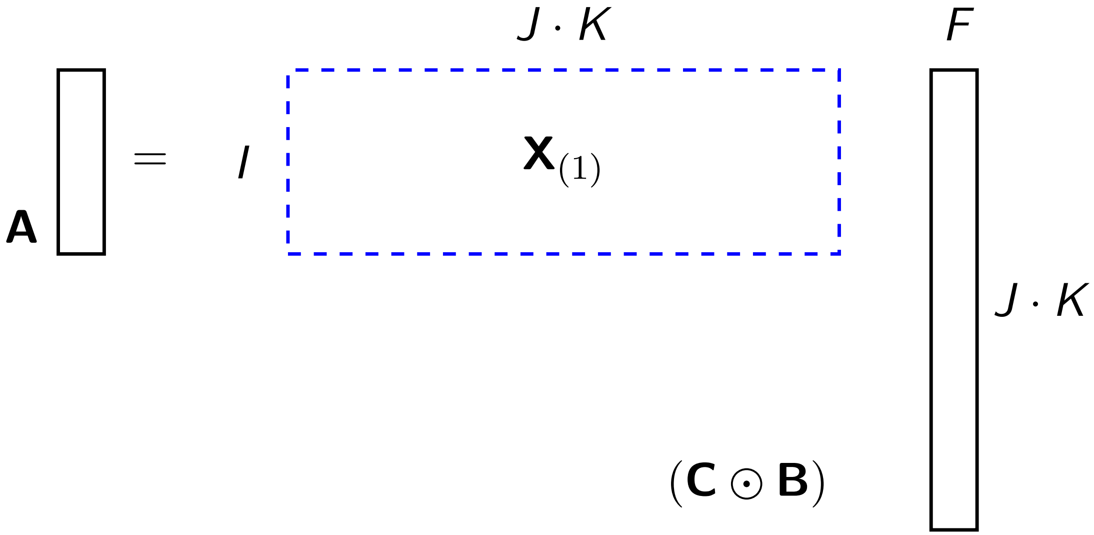
Algorithm 1 CPD-ALS

```
1: while not converged do
2:     $\mathbf{A}^T = (\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I})^{-1} \left(\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})\right)^T$
3:     $\mathbf{B}^T = (\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1} \left(\mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})\right)^T$
4:     $\mathbf{C}^T = \underbrace{(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}}_{\text{Normal equations}} \underbrace{\left(\mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})\right)^T}_{\text{MTTKRP}}$
5: end while
```

Matricized tensor times Khatri-Rao product

MTTKRP is the core computation of each iteration of CPD-ALS:

$$\mathbf{A} = \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$$



MTTKRP – elementwise

Elementwise formulation:

$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j,k)\,[\mathbf{B}(j,:) * \mathbf{C}(k,:)]$$

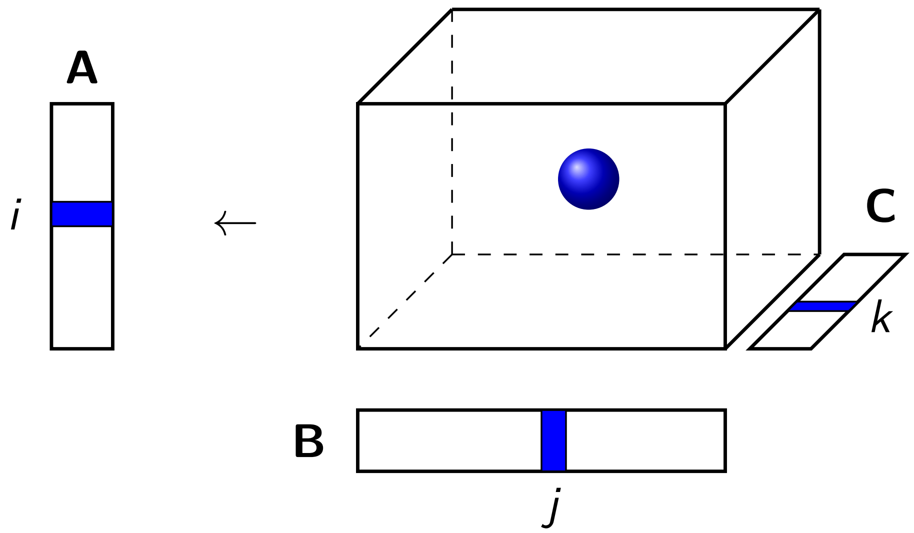


Table of Contents

Can we do better?

- Consider three nonzeros in the fiber $\boldsymbol{\mathcal{X}}(i, j, :)$ (a vector):

$$\begin{aligned}
\mathbf{A}(i,:) &\leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j,k_1)\ \ [\mathbf{B}(j,:) * \mathbf{C}(k_1,:)] \\
\mathbf{A}(i,:) &\leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j,k_2)\ \ [\mathbf{B}(j,:) * \mathbf{C}(k_2,:)] \\
\mathbf{A}(i,:) &\leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j,k_3)\ \ [\mathbf{B}(j,:) * \mathbf{C}(k_3,:)]
\end{aligned}$$

Can we do better?

- Consider three nonzeros in the fiber $\mathcal{X}(i,j,:)$ (a vector):

$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \mathcal{X}(i,j,k_1)\ [\mathbf{B}(j,:) * \mathbf{C}(k_1,:)]$$
$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \mathcal{X}(i,j,k_2)\ [\mathbf{B}(j,:) * \mathbf{C}(k_2,:)]$$
$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \mathcal{X}(i,j,k_3)\ [\mathbf{B}(j,:) * \mathbf{C}(k_3,:)]$$

- A little factoring...

$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \mathbf{B}(j,:) * \left[\sum_{x=1}^{3} \mathcal{X}(i,j,k_x)\mathbf{C}(k_x,:) \right]$$

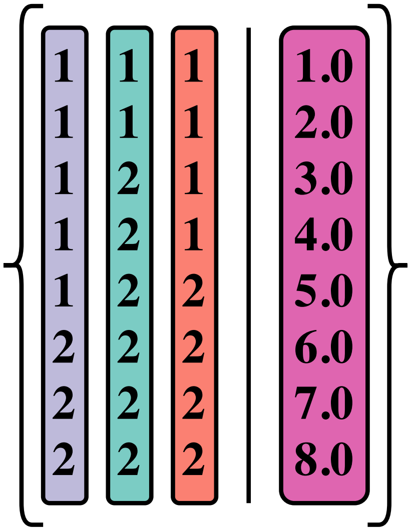
If α is the number of non-zeros in the (i,j) fiber, then from $\alpha F + 2\alpha F$ operations it reduces to $(1+\alpha)F + (1+\alpha)F$.

Challenges

- Factoring out computation relies on sparsity structure.

$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \ \mathbf{B}(j,:) * \left[\sum_{x=1}^{3} \boldsymbol{\mathcal{X}}(i,j,k_x)\mathbf{C}(k_x,:) \right]$$

- Coordinate format does not naturally expose structure.

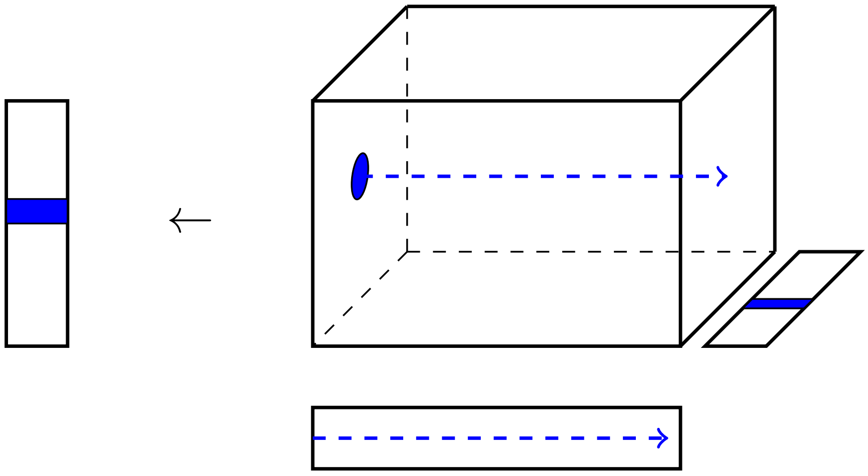


1	1	1	1.0
1	1	1	2.0
1	2	1	3.0
1	2	1	4.0
1	2	2	5.0
2	2	2	6.0
2	2	2	7.0
2	2	2	8.0

Efficient MTTKRP [Smith et al. '15]

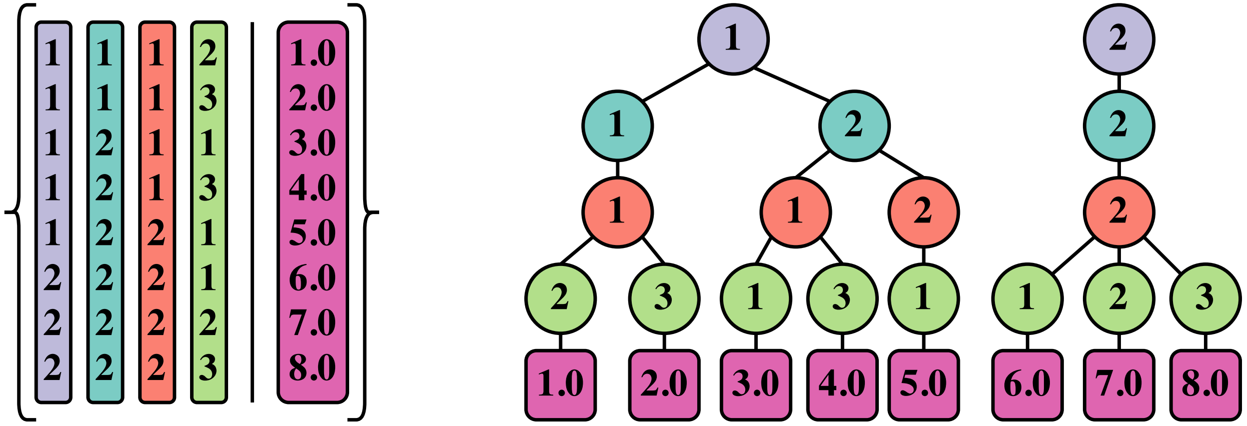
What is an ideal data structure for exposing structure?

- Fibers are stored contiguously.
- Slice $\mathcal{X}(i,:,:)$ is almost a CSR matrix.

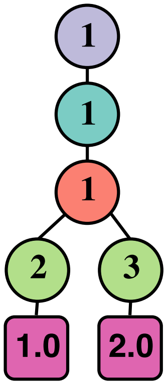


Compressed sparse fiber (CSF) [Smith & Karypis '15]

- Modes are recursively compressed.
- Paths from roots to leaves encode non-zeros.
- The tree structure encodes the opportunities for savings.



MTTKRP example



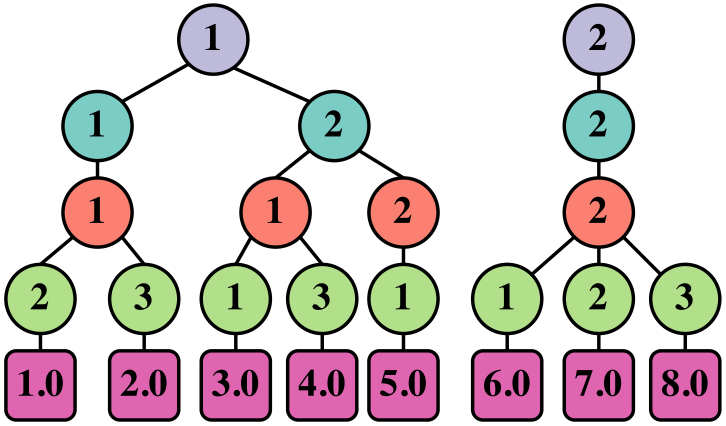
$$\mathbf{A}^{(1)}(1,:) \leftarrow \mathbf{A}^{(1)}(1,:) + 1.0\left(\mathbf{A}^{(2)}(1,:) * \mathbf{A}^{(3)}(1,:) * \mathbf{A}^{(4)}(2,:)\right)$$
$$\mathbf{A}^{(1)}(1,:) \leftarrow \mathbf{A}^{(1)}(1,:) + 2.0\left(\mathbf{A}^{(2)}(1,:) * \mathbf{A}^{(3)}(1,:) * \mathbf{A}^{(4)}(3,:)\right)$$

Becomes:

$$\mathbf{A}^{(1)}(1,:) \leftarrow \mathbf{A}^{(1)}(1,:) + \left(\mathbf{A}^{(2)}(1,:) * \left(\mathbf{A}^{(3)}(1,:) * \left(1.0 \cdot \mathbf{A}^{(4)}(2,:) + 2.0 \cdot \mathbf{A}^{(4)}(3,:)\right)\right)\right)$$

Mode-centric CSF

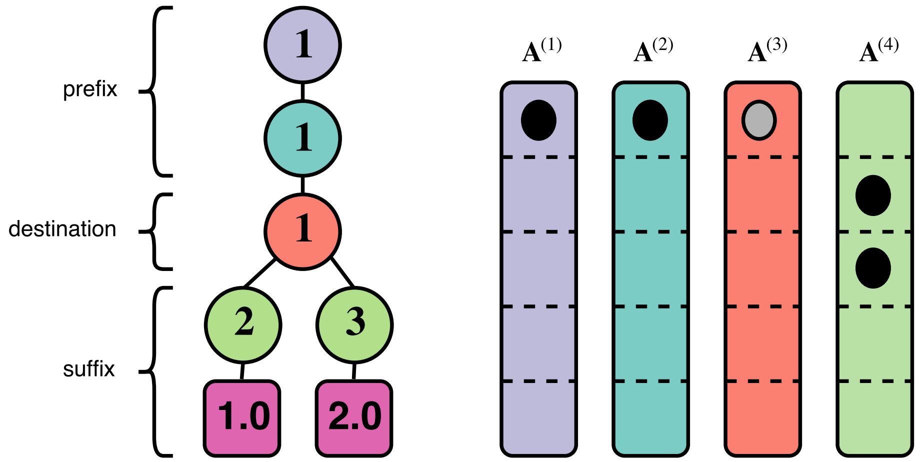
- All of the non-zeros required to compute $\mathbf{A}^{(1)}(i,:)$ are found in the ith tree.
- Parallelism is easy: just distribute the trees to threads.
- This strategy requires a different CSF representation for each mode.
  - The mode-m CSF places the mth mode at the top.



- Can we work with just a single CSF?

MTTKRP example

Each tree is traversed depth-first.



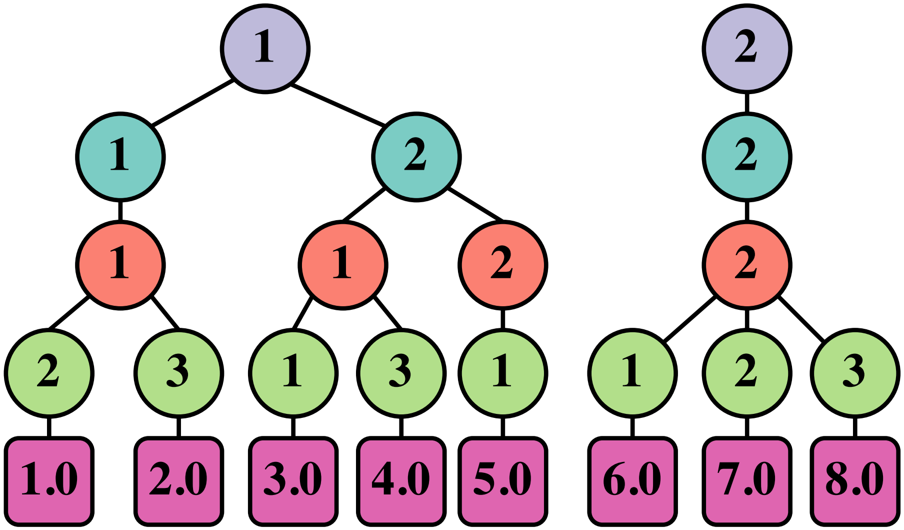
$$\textit{prefix} \leftarrow \mathbf{A}^{(1)}(1,:) * \mathbf{A}^{(2)}(1,:)$$

$$\textit{suffix} \leftarrow 1.0 \cdot \mathbf{A}^{(4)}(2,:) + 2.0 \cdot \mathbf{A}^{(4)}(3,:)$$

$$\mathbf{A}^{(3)}(1,:) \leftarrow \mathbf{A}^{(3)}(1,:) + (\textit{prefix} * \textit{suffix})$$

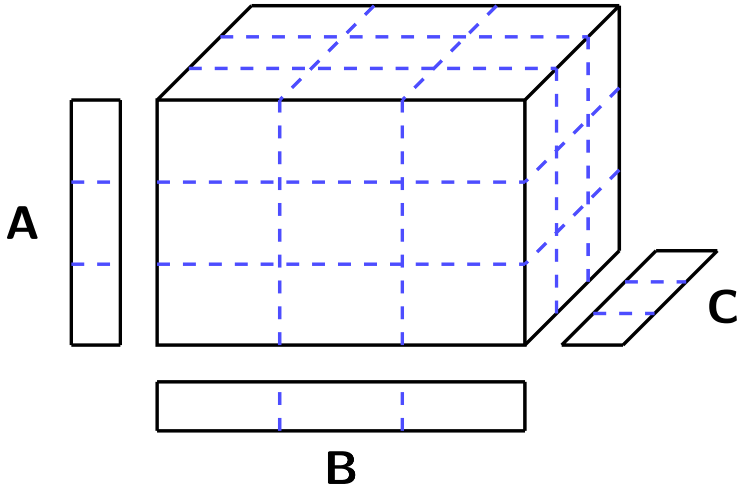
Parallelism – challenges?

- Like before, parallelize over the trees when computing for the root level.
- Internal and leaf modes require more thought.
  - First approach: mutex pool.



Parallelism – tiling

- For p threads, do a p-way tiling of each tensor mode.
- Distributing the tiles allows us to eleminate the need for mutexes.



Experimental setup

Implementation:

- SPLATT: the Surprisingly ParalleL spArse Tensor Toolkit.
- Written in C99 with OpenMP parallelism.

Benchmarks:

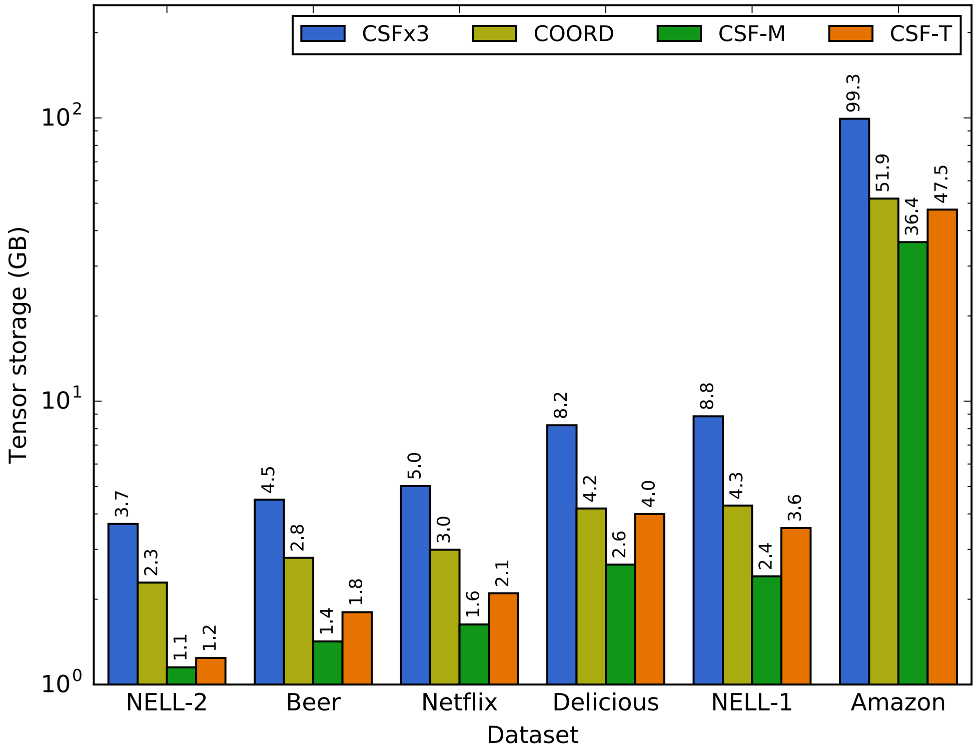
- **COORD**: a coordinate form representation.
- **CSF-M**: a single CSF representations with mutexes.
- **CSF-T**: a single tiled CSF representation.
- **CSFx3**: a separate, untiled CSF representation for each mode.

Machine configuration:

- Experiments performed on the Itasca-sb supercomputer at MSI.
- Nodes have two eight-core Intel processors (Ivy Bridge).

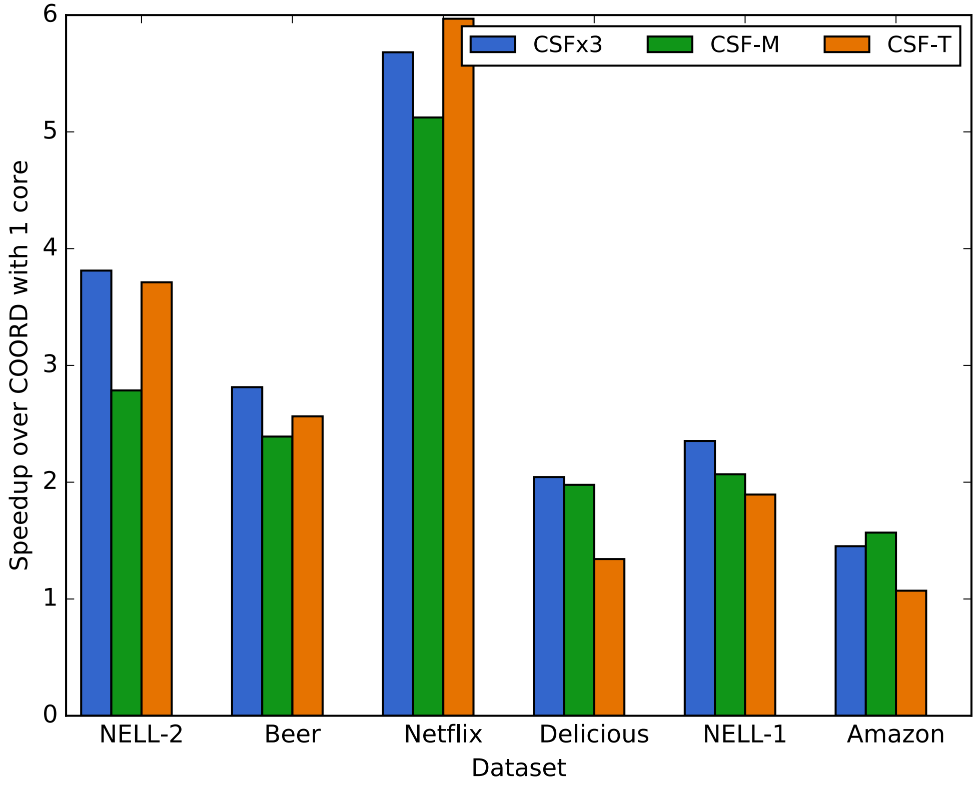
Storage comparison

Tiling overheads never exceed coordinate storage and offer significant parallelism benefits.



Serial MTTKRP

CSF variants achieve $1.5 - 6.0\times$ speedup due to memory bandwidth and operation reduction.



Parallel MTTKRP

CSFx3 and CSF-T achieve $10 - 55\times$ speedup relative to serial COORD.

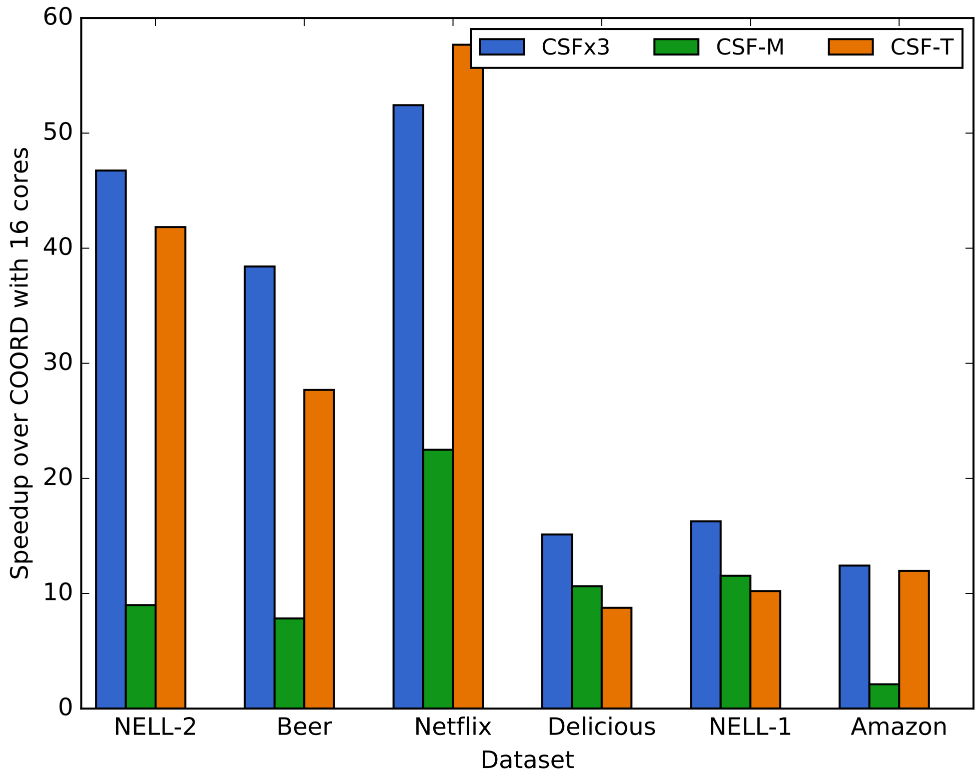


Table of Contents

Tensor reordering

We *reorder* the tensor to improve the access patterns on the factors.

$$\mathbf{X}_{(1)} = \left[\begin{array}{cccc|cccc|cccc}
 & \color{red}3 & & \color{red}3 & & & & & \color{blue}2 & & & \color{blue}2 \\
 & & & & \color{cyan}1 & & \color{cyan}1 & & & & & \\
 & & & & \color{cyan}1 & & \color{cyan}1 & & \color{blue}2 & & & \color{blue}2 \\
 & \color{red}3 & & \color{red}3 & & & & & & & &
\end{array}\right]$$

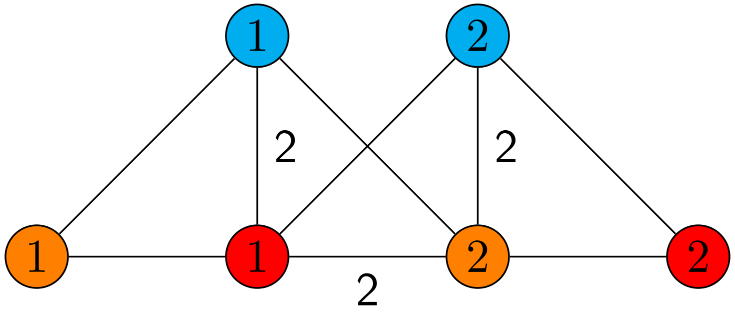
$$\mathbf{X'}_{(1)} = \left[\begin{array}{cccc|cccc|cccc}
\color{red}3 & \color{red}3 & & & & & & & & & & \\
\color{red}3 & \color{red}3 & & & & \color{blue}2 & \color{blue}2 & & & & & \\
 & & & & & \color{blue}2 & \color{blue}2 & & & & \color{cyan}1 & \color{cyan}1 \\
 & & & & & & & & & & \color{cyan}1 & \color{cyan}1
\end{array}\right]$$

Tensor reordering

Graph partitioning

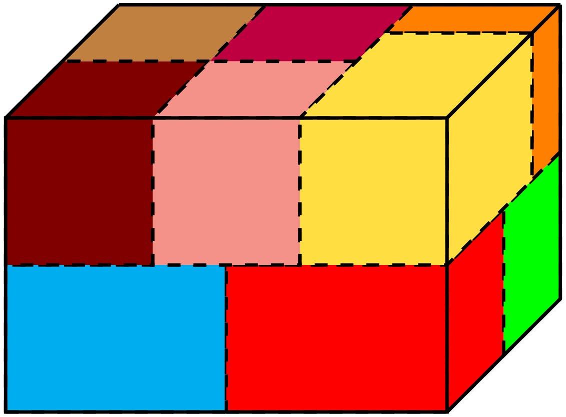
- We model the sparsity structure of $\boldsymbol{\mathcal{X}}$ with a tripartite graph.
  - Slices are vertices, nonzeros connect slices with a triangle.
- Partitioning the graph finds regions with shared indices.
- We reorder the tensor to group indices in the same partition.

$$\left[\begin{array}{ccc} \textbf{i} & \textbf{j} & \textbf{k} \\ \hline 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{array}\right] \rightarrow$$



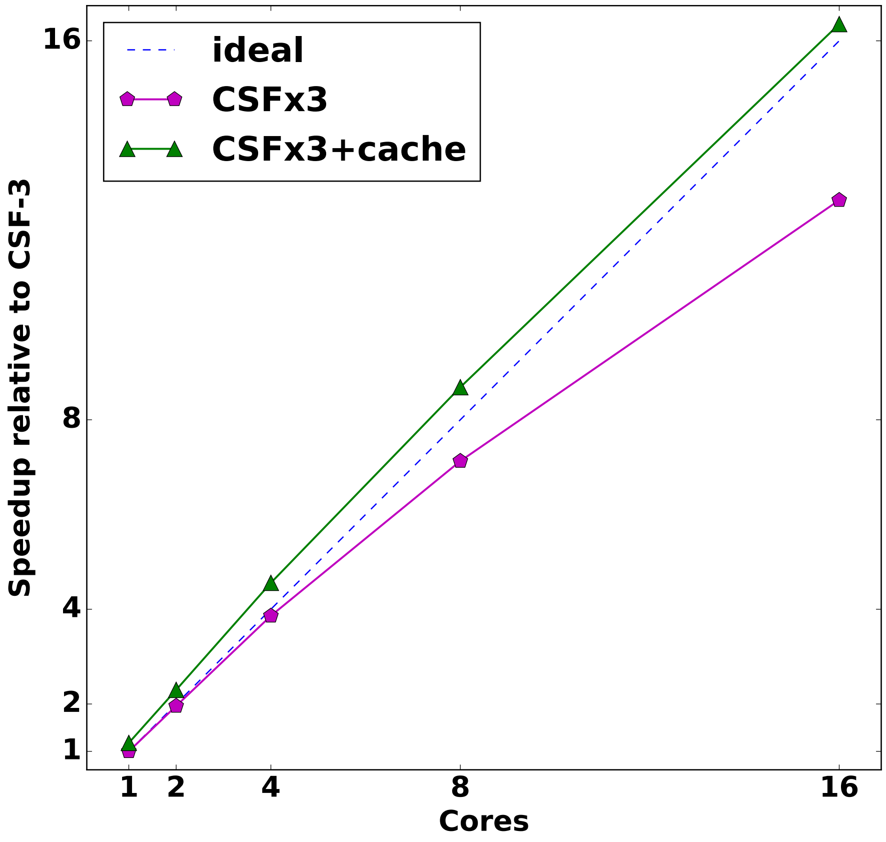
Cache blocking over tensors

- Tiling lets us schedule nonzeros to reuse indices already in cache.
- Cost: more fibers.
- Tensor sparsity forces us to *grow* tiles.
  - Cache tiles are not aligned, and thus do not let us parallelize over any tensor mode like CSF-T.
  - So, we return to CSFx3.



Scaling NELL-2, speedup vs untiled

Optimizations result in small serial improvement and linear speedup.



Scaling Netflix, speedup vs untiled

Linear scalability is achieved on a two-socket NUMA system.

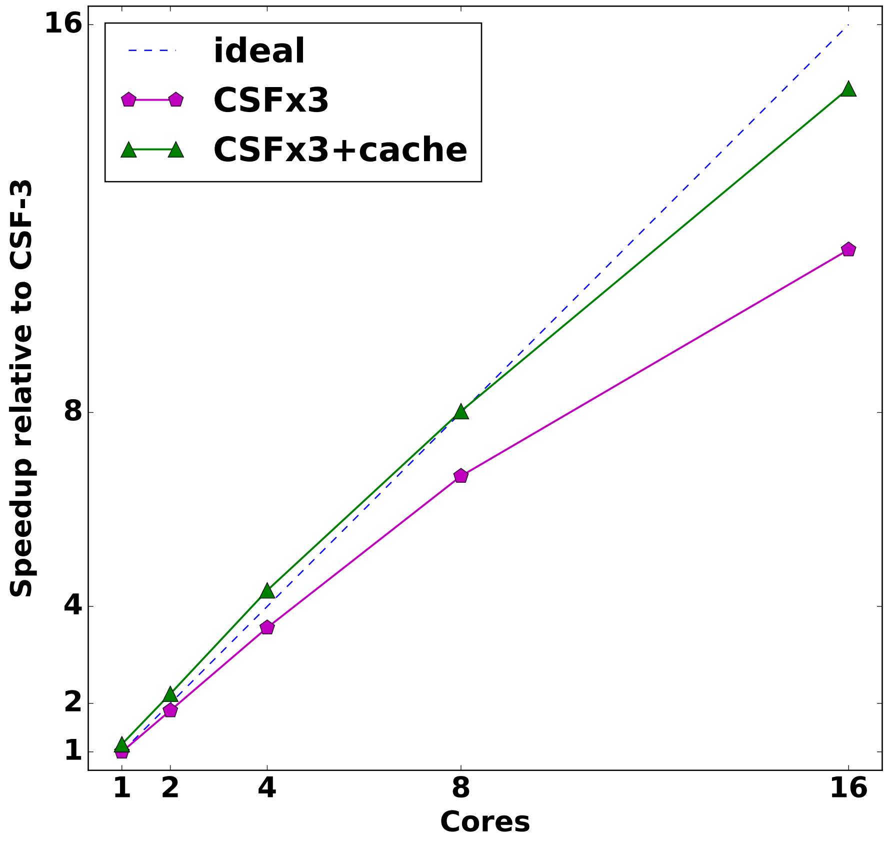
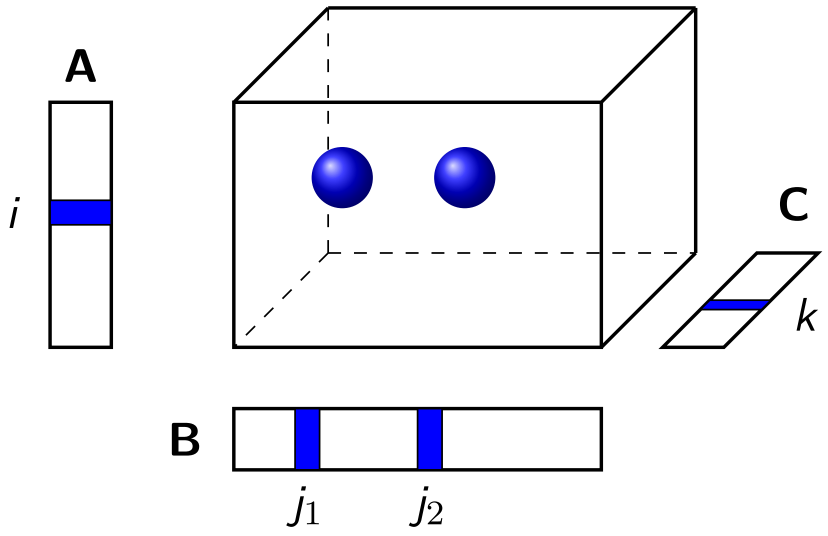


Table of Contents

MTTKRP communication

Non-zeros with shared indices contribute to the same MTTKRP output, and are also used in later ALS steps.

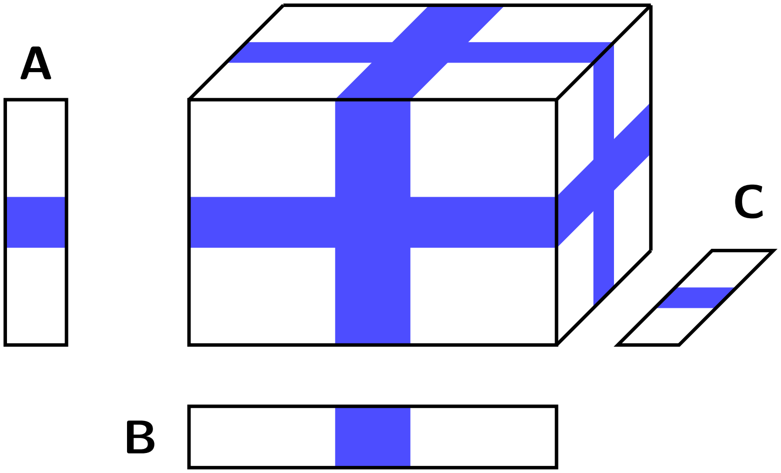


$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j_1,k)\,[\mathbf{B}(j_1,:) * \mathbf{C}(k,:)]$$
$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \boldsymbol{\mathcal{X}}(i,j_2,k)\,[\mathbf{B}(j_2,:) * \mathbf{C}(k,:)]$$

Coarse-grained decomposition

[Choi & Vishwanathan '14, Shin & Kang '14]

- Processes own complete slices of $\boldsymbol{\mathcal{X}}$ and aligned factor rows.
- I/p rows communicated to $p-1$ processes after each update.



Fine-grained decomposition

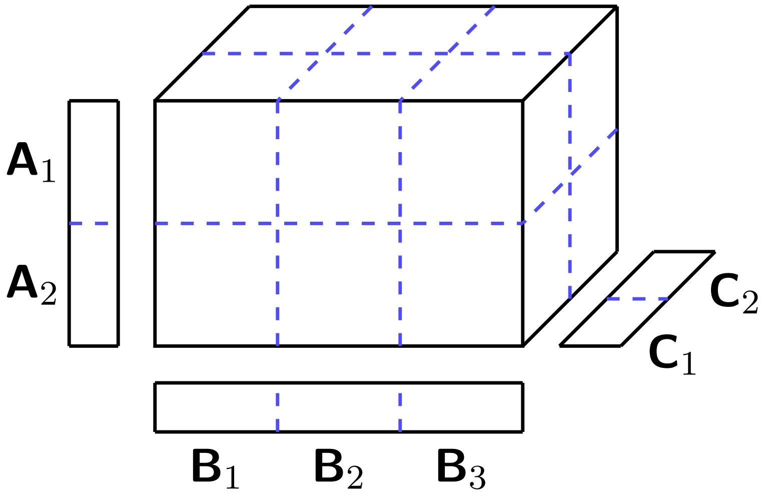
[Kaya & Uçar '15]

- Most flexible: non-zeros individually assigned to processes.
- Two communication steps:
  1. Aggregate partial computations after MTTKRP.
  2. Exchange the updated factor values (dual of step 1).
- Hypergraph partitioning is used to minimize communication:
  - Non-zeros mapped to vertices.
  - $I+J+K$ hyperedges.

Medium-grained decomposition

[Smith & Karypis '16]

- Distribute over a grid of $p = q \times r \times s$ partitions.
- $r \times s$ processes divide each $\mathbf{A}_1, \ldots, \mathbf{A}_q$.
- Two communication steps like fine-grained.
  - $\mathcal{O}(I/p)$ rows communicated to $r \times s$ processes.



Experimental setup

Benchmarks:

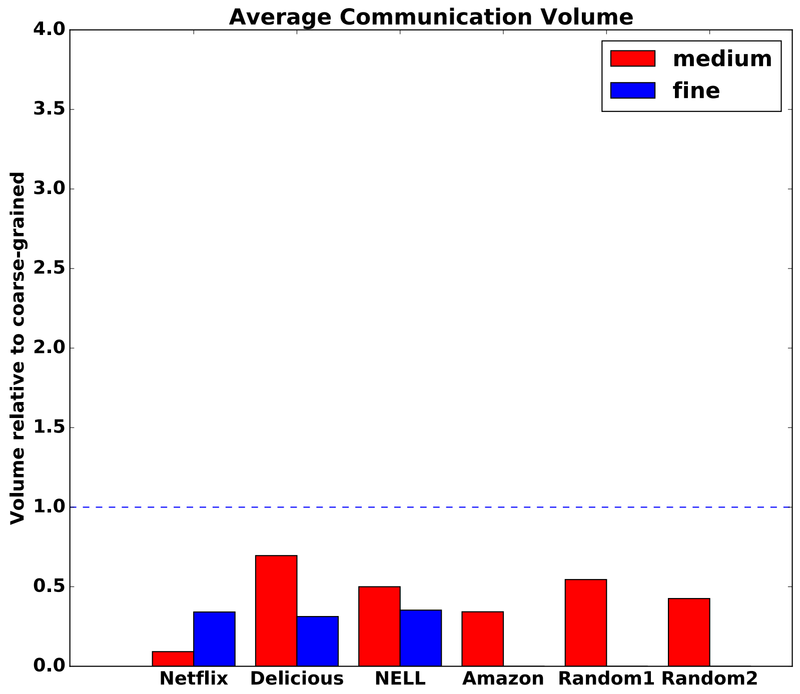
- **DFacTo**: a publicly available coarse-grained MPI code.
- **coarse**: SPLATT with coarse-grained decomposition.
- **medium**: SPLATT with medium-grained decomposition.
- **fine**: SPLATT with fine-grained decomposition.
  - PaToH used for hypergraph partitioning.

Machine configuration:

- All experiments performed on the Itasca supercomputer at MSI.
- Nodes have two quad-core Intel processors (Sandy Bridge).

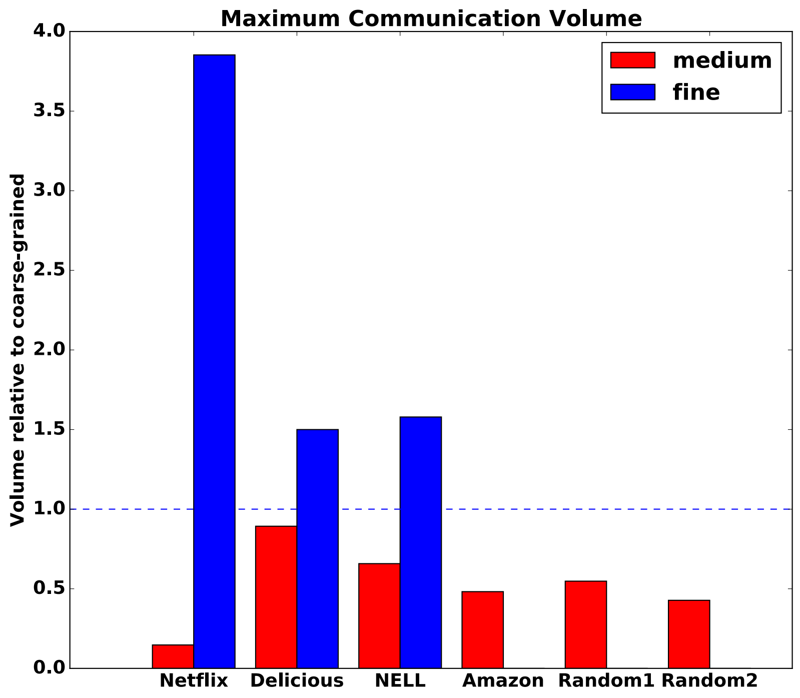
Average comm. volume with 128 MPI ranks

- Fine-grained decompositions have low communication volume when they are feasible.
- Exception: a 1D decomposition on the Netflix user mode.



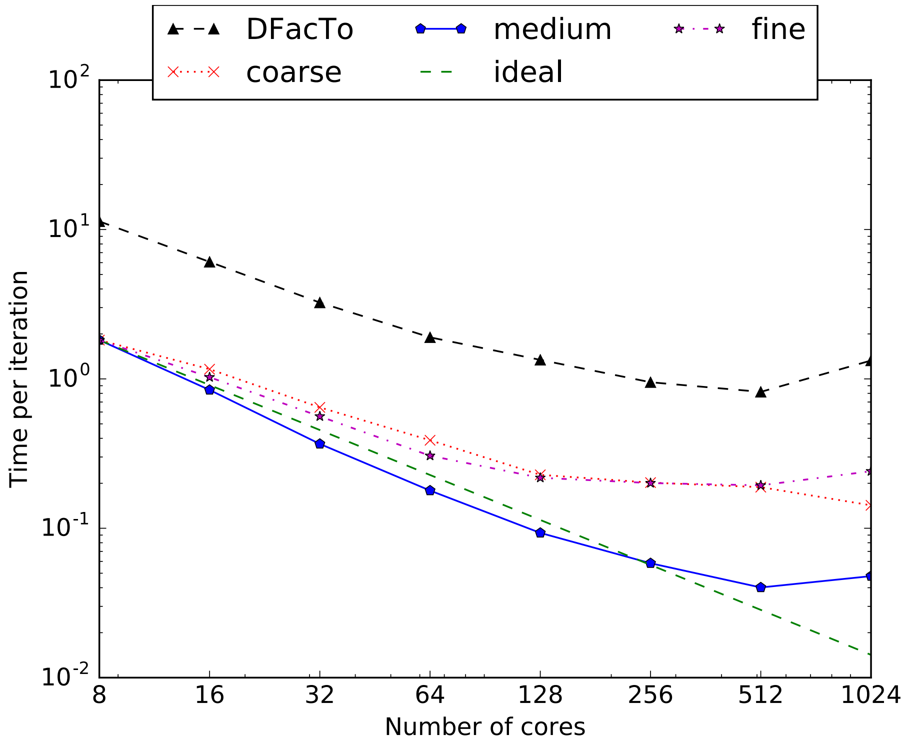
Maximum comm. volume with 128 MPI ranks

Hypergraph partitioning does not guarantee a balanced communication volume.



Strong scaling: Netflix

All methods scale to 512 cores (64 MPI ranks).



Strong scaling: Amazon

Only the medium-grained decomposition scales on the large and sparse Amazon.

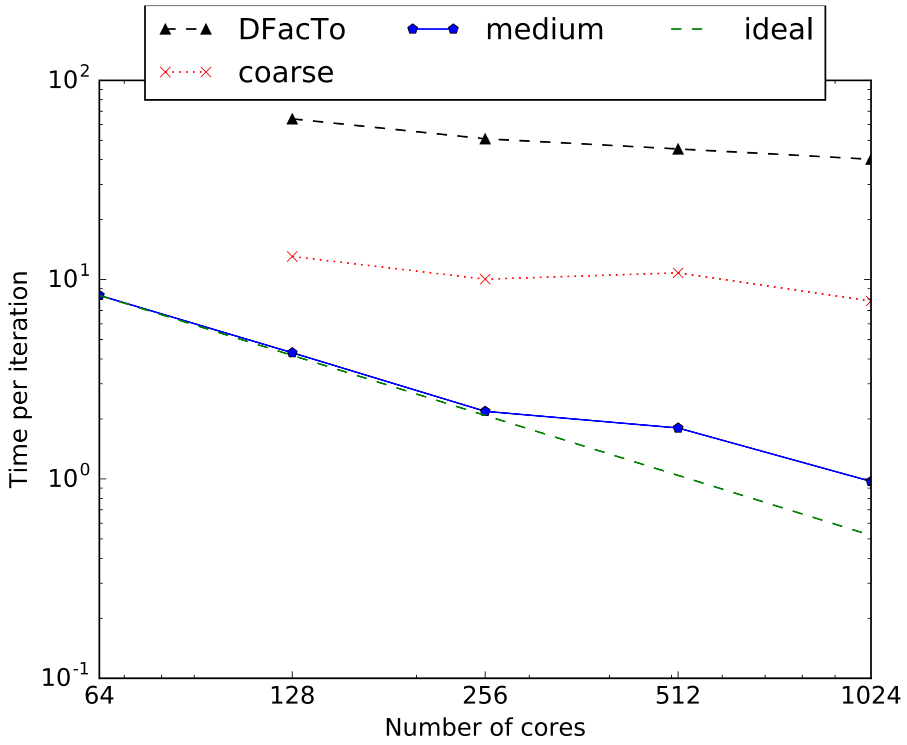


Table of Contents

Tensor completion: loss function

- Objective: predict missing entries of $\boldsymbol{\mathcal{X}}$.
- We only want to model *observed* entries (non-zeros).
  - A least-squares objective would predict zeros!
- The objective is a combination of the predicition ability (the *loss*) and regularization terms (to prevent overfitting).

$$\underset{\mathbf{A},\mathbf{B},\mathbf{C}}{\text{minimize}} \qquad \underbrace{\mathcal{L}(\boldsymbol{\mathcal{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C})}_{\text{Loss}} + \underbrace{\lambda \left(||\mathbf{A}||_F^2 + ||\mathbf{B}||_F^2 + ||\mathbf{C}||_F^2 \right)}_{\text{Regularization}}$$

$$\mathcal{L}(\boldsymbol{\mathcal{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \sum_{\text{nnz}(\boldsymbol{\mathcal{X}})} \left(\boldsymbol{\mathcal{X}}(i,j,k) - \sum_{f=1}^{F} \mathbf{A}(i,f)\mathbf{B}(j,f)\mathbf{C}(k,f) \right)^2$$

Challenges

Optimization algorithms

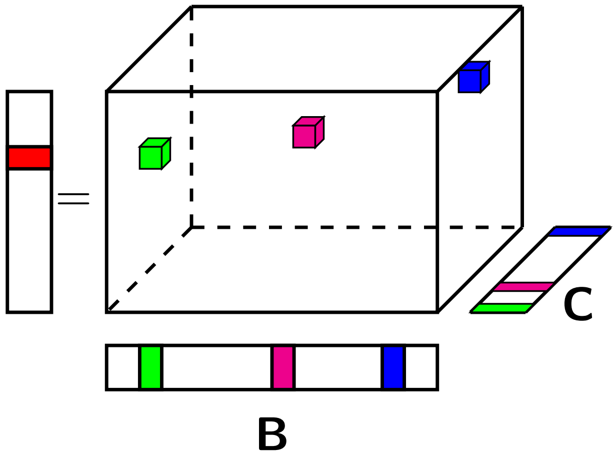
- Algorithms for *matrix* completion are relatively mature.
  - How do they adapt to tensors?
- We must consider multiple properties when comparing algorithms:
  1. Number of operations.
  2. Convergence rate.
  3. Computational intensity.
  4. Parallelism.

Tensor properties

- Most matrix optimization algorithms parallelize over the many rows and columns (e.g., users and items).
- Many domains have a mix of short and long modes.

Alternating least squares (ALS)

- Each row of **A** is a linear least squares problem.
- $\mathbf{H}_i$ is an $|\boldsymbol{\mathcal{X}}(i,:,:)| \times F$ matrix:
  - $\boldsymbol{\mathcal{X}}(i,j,k) \rightarrow \mathbf{B}(j,:) * \mathbf{C}(k,:)$ (elementwise multiplication).
- $\mathbf{A}(i,:) \leftarrow \underbrace{\left(\mathbf{H}_i^T \mathbf{H}_i + \lambda \mathbf{I}\right)^{-1}}_{\text{normal eq.}} \underbrace{\mathbf{H}_i^T \operatorname{vec}(\boldsymbol{\mathcal{X}}(i,:,:))}_{\text{MTTKRP}}.$
- $\mathcal{O}(F^2)$ work per non-zero.



Alternating least squares (ALS)

- Normal equations $\mathbf{N}_i = \mathbf{H}_i^T \mathbf{H}_i$ are formed one non-zero at a time.
- $\mathbf{H}_i^T \operatorname{vec}(\boldsymbol{\mathcal{X}}(i,:,:))$ is similarly accumulated into a vector q_i.

Algorithm 2 ALS: updating $\mathbf{A}(i,:)$

```
1: $\mathbf{N}_i \leftarrow \mathbf{0}^{F\times F}$, $q_i \leftarrow \mathbf{0}^{F\times 1}$
2: for $(i,j,k) \in \boldsymbol{\mathcal{X}}(i,:,:)$ do
3:     $x \leftarrow \mathbf{B}(j,:) * \mathbf{C}(k,:)$
4:     $\mathbf{N}_i \leftarrow \mathbf{N}_i + x^T x$
5:     $q_i \leftarrow q_i + \boldsymbol{\mathcal{X}}(i,j,k) x^T$
6: end for
7: $\mathbf{A}(i,:) \leftarrow (\mathbf{N}_i + \lambda \mathbf{I})^{-1} q_i$
```

Shared-memory parallelism

[Shao '12]

- Least squares problems are solved in batches of size $B = \mathcal{O}(100)$.
- Each core independently accumulates $\mathbf{N}_i$ and q_i.
- Corresponding $\mathbf{N}_i$ and q_i are aggregated.
- Finally, the B inversions and updates are performed in parallel.

[Smith et al. '16]

- Storing multiple representations of $\boldsymbol{\mathcal{X}}$ allows us to parallelize over rows of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.
  - No parallel reductions or synchronization required.
  - Each core only requires $\mathcal{O}(F^2)$ intermediate storage for $\mathbf{N}_i$.
- If mode is short, use method of [Shao '12] with a single batch of size equal to the dimension of that mode.

Level-3 BLAS formulation [Smith et al. '16]

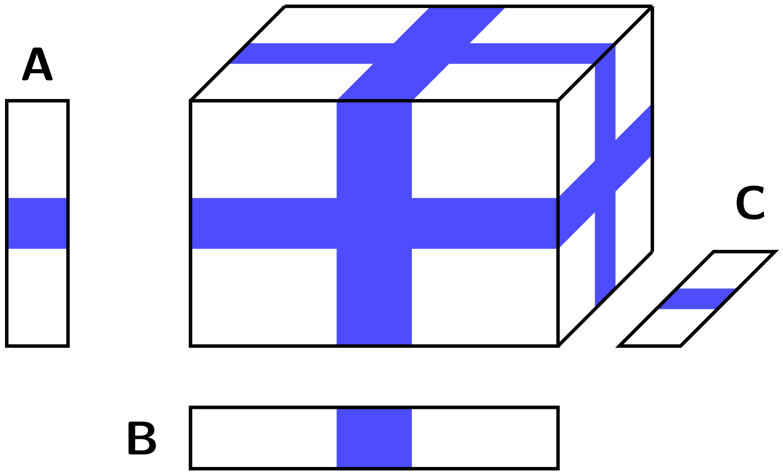
- Element-wise computation is an outer product formulation.
  - $\mathcal{O}(F^2)$ work with $\mathcal{O}(F^2)$ data per non-zero.
- Place $(\mathbf{B}(j,:) * \mathbf{C}(k,:))$ into rows of a thread-local matrix $\mathbf{Z}$.
  - When $\mathbf{Z}$ is full, do a rank-k update: $\mathbf{N}_i \leftarrow \mathbf{N}_i + \mathbf{Z}^T\mathbf{Z}$.

Algorithm 3 ALS: updating $\mathbf{A}(i,:)$

```
 1: $\mathbf{N}_i \leftarrow \mathbf{0}^{F\times F}$, $q_i \leftarrow \mathbf{0}^{F\times 1}$, $\mathbf{Z} \leftarrow \mathbf{0}$
 2: for $(i,j,k) \in \boldsymbol{\mathcal{X}}(i,:,:)$ do
 3:     $x \leftarrow \mathbf{B}(j,:) * \mathbf{C}(k,:)$
 4:     if $\mathbf{Z}$ is full then
 5:         $\mathbf{N}_i \leftarrow \mathbf{N}_i + \mathbf{Z}^T\mathbf{Z}$, $\mathbf{Z} \leftarrow \mathbf{0}$
 6:     end if
 7:     $q_i \leftarrow q_i + \boldsymbol{\mathcal{X}}(i,j,k)x^T$
 8: end for
 9: $\mathbf{N}_i \leftarrow \mathbf{N}_i + \mathbf{Z}^T\mathbf{Z}$
10: $\mathbf{A}(i,:) \leftarrow (\mathbf{N}_i + \lambda\mathbf{I})^{-1} q_i$
```

Distributed-memory parallelism

- Communicating the normal equations is very expensive.
  - $\mathcal{O}(IF^2)$ data communicated per process.
- Use a coarse-grained decomposition.
- Each process owns all necessary non-zeros and only needs to exchange the updated factor rows.
- If mode is short, just communicate the normal equations with an all-to-all reduction.



Stochastic gradient descent (SGD)

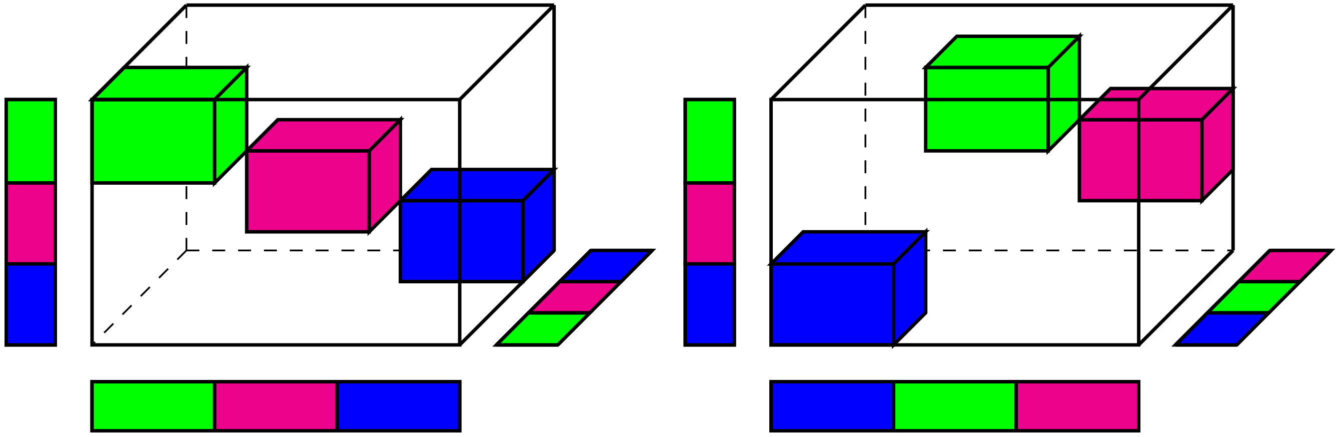
- Randomly select entry $\boldsymbol{\mathcal{X}}(i,j,k)$ and update rows of **A**, **B**, and **C**.
    - $\mathcal{O}(F)$ work per non-zero.
- η is the step size; typically $\mathcal{O}(10^{-3})$.

$$\delta \leftarrow \boldsymbol{\mathcal{X}}(i,j,k) - \sum_{f=1}^{F} \mathbf{A}(i,f)\mathbf{B}(j,f)\mathbf{C}(k,f)$$

$$\mathbf{A}(i,:) \leftarrow \mathbf{A}(i,:) + \eta\left[\delta\left(\mathbf{B}(j,:) * \mathbf{C}(k,:)\right) - \lambda\mathbf{A}(i,:)\right]$$
$$\mathbf{B}(j,:) \leftarrow \mathbf{B}(j,:) + \eta\left[\delta\left(\mathbf{A}(i,:) * \mathbf{C}(k,:)\right) - \lambda\mathbf{B}(j,:)\right]$$
$$\mathbf{C}(k,:) \leftarrow \mathbf{C}(k,:) + \eta\left[\delta\left(\mathbf{A}(i,:) * \mathbf{B}(j,:)\right) - \lambda\mathbf{C}(k,:)\right]$$

SGD: stratification [Beutel '14]

- *Strata* identify independent blocks of non-zeros.
- Each stratum is processed in parallel.

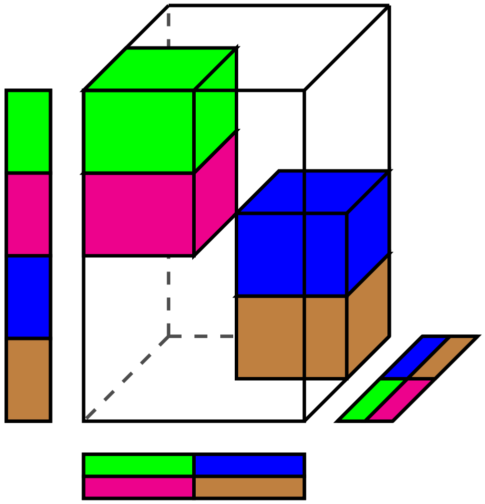


Limitations of stratification:

- There is only as much parallelism as the smallest dimension.
- Sparsely populated strata are communication bound.

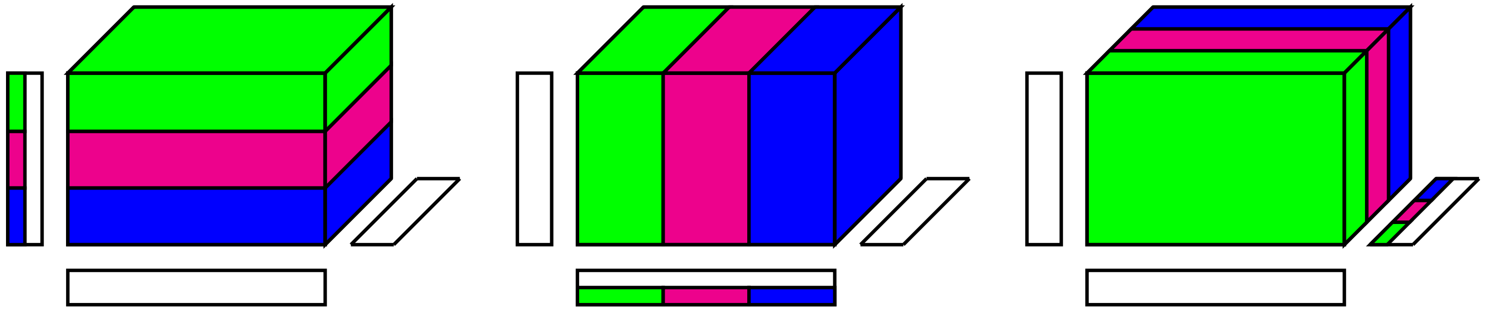
Hybrid stratified/asynchronous SGD [Smith et al. '16]

- Limit the number of strata to reduce communication and handle short modes.
- Assign multiple processes to the same stratum (called a *team*).
- Each performs updates on its own versions of the factors.
- At the end, the updates are exchanged among the team.



Coordinate descent (CCD++)

- Rank-1 factors are updated in sequence.
- $\mathcal{O}(F)$ work per non-zero (same as SGD).



CCD++ parallelism

$$
\begin{aligned}
\delta_{ijk} &\leftarrow \boldsymbol{\mathcal{X}}(i,j,k) - \sum_{f=1}^{F} \mathbf{A}(i,f)\mathbf{B}(j,f)\mathbf{C}(k,f) \\
\alpha_i &\leftarrow \sum_{\boldsymbol{\mathcal{X}}(i,:,:)} \delta_{ijk} \left(\mathbf{B}(j,f)\mathbf{C}(k,f)\right) \\
\beta_i &\leftarrow \sum_{\boldsymbol{\mathcal{X}}(i,:,:)} \left(\mathbf{B}(j,f)\mathbf{C}(k,f)\right)^2 \\
\mathbf{A}(i,f) &\leftarrow \frac{\alpha_i}{\lambda + \beta_i}
\end{aligned}
$$

[Karlsson '15, Shin '15]

- Each entry of $\mathbf{A}(:,f)$ is computed in parallel.
  - Distributing non-zeros requires α_i and β_i to be aggregated.
  - Communication volume is $\mathcal{O}(IF)$ per process.
- All δ_{ijk} are totally parallel - no communication needed.

Shared-memory parallelism with CSF [Smith et al. '16]

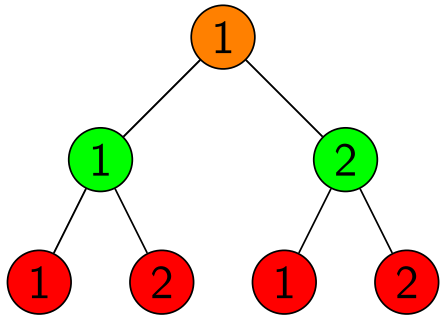
- Column-wise methods require F passes over the sparse tensor.
  - CCD++ requires a high memory bandwidth.
- CSF shrinks the memory footprint of the tensor and structures memory accesses.
  - Fewer operations and a reduced memory bandwidth.

Example: loss computation.

$$\mathbf{v} \leftarrow \mathbf{A}(i,:) * \mathbf{B}(j,:),$$

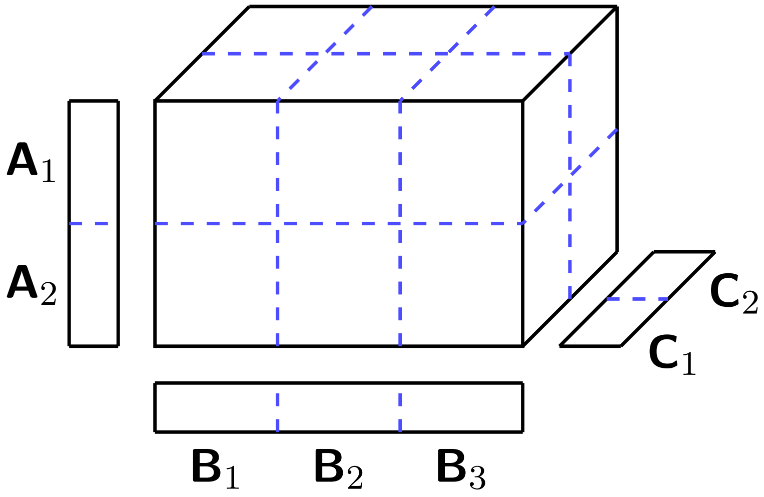
$$\delta_{ijk} \leftarrow \boldsymbol{\mathcal{X}}(i,j,k) - \sum_{f=1}^{F} \mathbf{v}(f)\mathbf{C}(k,f),$$

$$\delta_{ijk'} \leftarrow \boldsymbol{\mathcal{X}}(i,j,k') - \sum_{f=1}^{F} \mathbf{v}(f)\mathbf{C}(k',f).$$



Distributed-memory parallelism

- A medium-grained decomposition limits communication volume to the grid layers.
- For short modes, we use a grid dimension of 1 and fully replicate the factor.
    - Non-zeros are still distributed and processed in parallel.

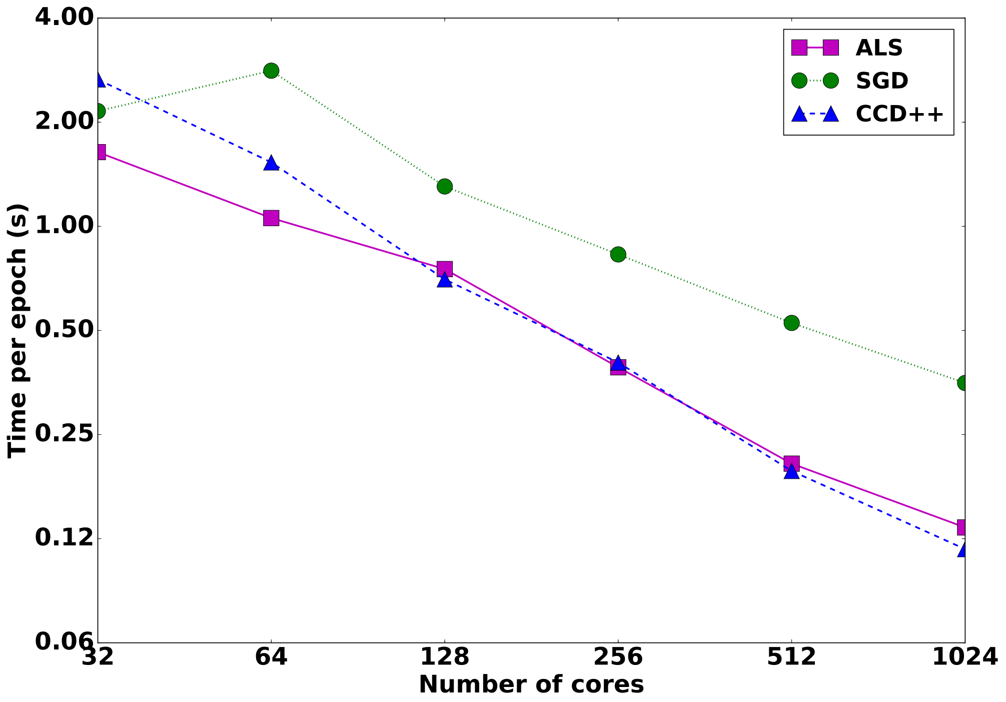


Experimental setup

- ALS, CCD++, and SGD are implemented as part of SPLATT.
- All experiments are on the Yahoo (KDD cup) tensor.
- All experiments performed on the Cori supercomputer at NERSC.
- Nodes have two sixteen-core Intel processors (Haswell).

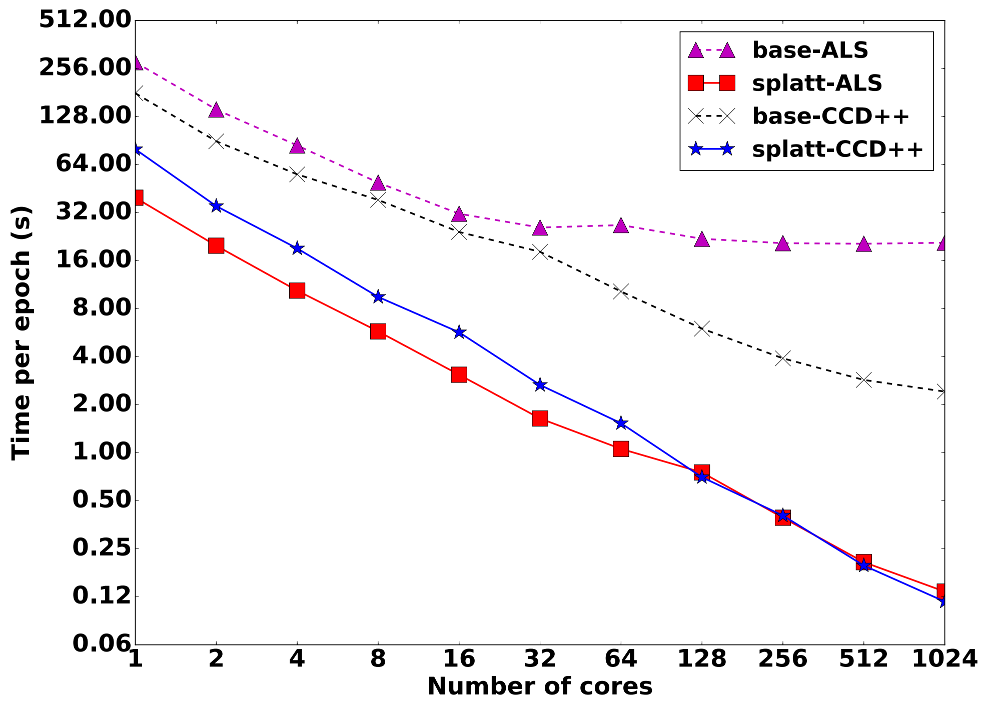
Strong Scaling: rank 10

- SGD exhibits initial slowdown as strata teams are populated.
- All methods scale to (past) 1024 cores.



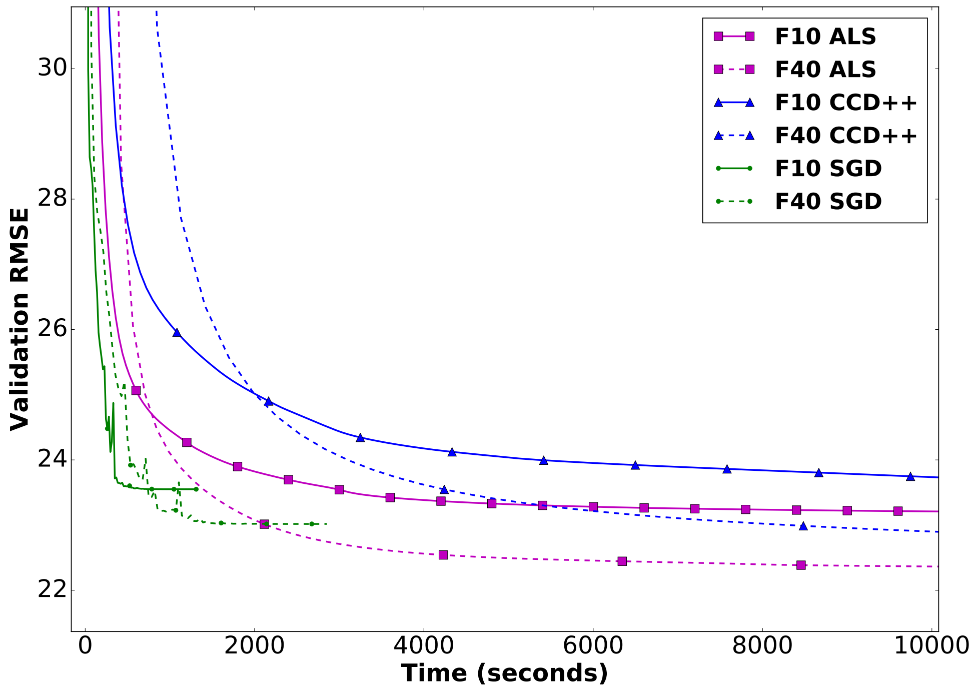
Benchmarking: rank 10

- base-ALS and base-CCD++ from [Karlsson et al. '15] (C++ and MPI).
- ALS and CCD++ are $153\times$ and $21\times$ faster, respectively.



Convergence @ 1 core

- Convergence is detected if the RMSE was not improved after 20 epochs.
- SGD rapidly converges to a high quality solution.



Convergence @ 1024 cores

- Convergence is detected if the RMSE was not improved after 20 epochs.
- ALS and CCD++ converge similarly. ALS has a slight advantage.

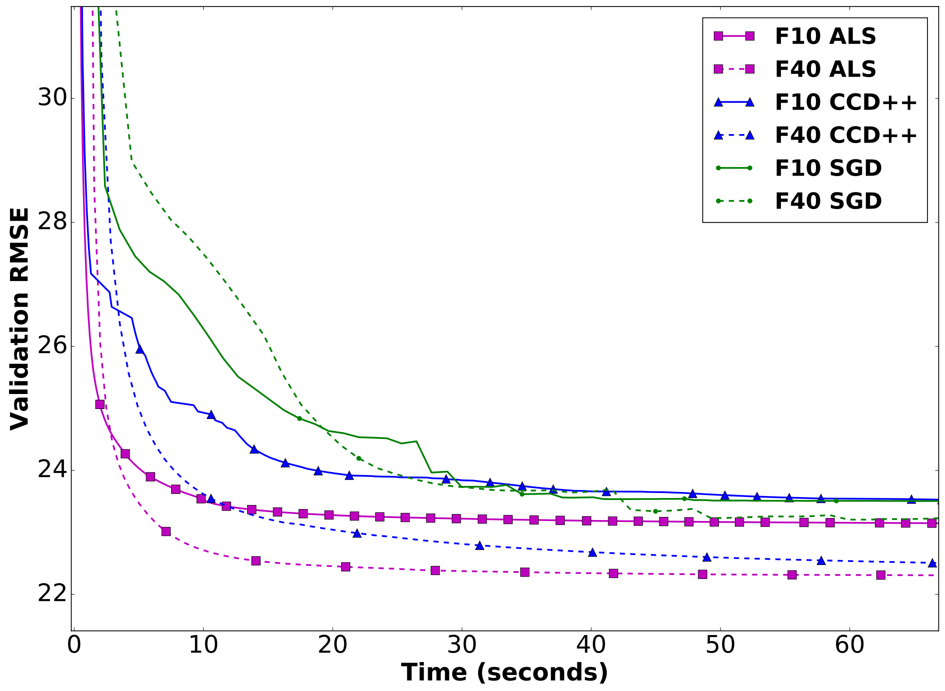


Table of Contents