Accelerating the Tucker Decomposition with Compressed Sparse Tensors

Shaden Smith and George Karypis

Department of Computer Science & Engineering, University of Minnesota
{shaden, karypis}@cs.umn.edu

Euro-Par 2017

Outline

Tensor Background

Computing the Tucker Decomposition

TTMc with a Compressed Sparse Tensor

Utilizing Multiple Compressed Tensors

Experiments

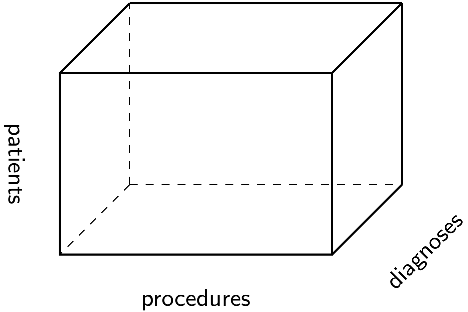
Conclusions

Table of Contents

Tensors

Tensors are the generalization of matrices to higher dimensions.

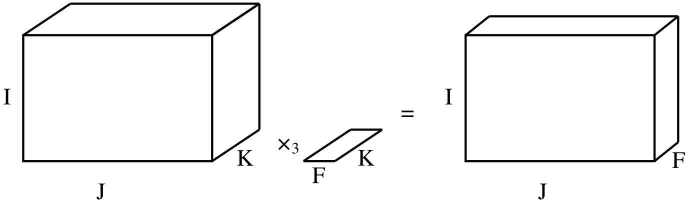
- Allow us to represent and analyze multi-dimensional data
- Applications in precision healthcare, cybersecurity, recommender systems, ...



Essential operation: tensor-matrix multiplication

Tensor-matrix multiplication (TTM; also called the *n*-way product)

- Given: tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ and matrix $\mathbf{M} \in \mathbb{R}^{F \times K}$.
- Operation: $\boldsymbol{\mathcal{X}} \times_3 \mathbf{M}$
- Output: $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I \times J \times F}$



Elementwise:

$$\boldsymbol{\mathcal{Y}}(i,j,f) = \sum_{k=1}^{K} \boldsymbol{\mathcal{X}}(i,j,k)\mathbf{M}(f,k).$$

Chained tensor-matrix multiplication (TTMc)

Tensor-matrix multiplications are often performed in sequence (*chained*).

$$\boldsymbol{\mathcal{Y}}_1 \leftarrow \boldsymbol{\mathcal{X}} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$$

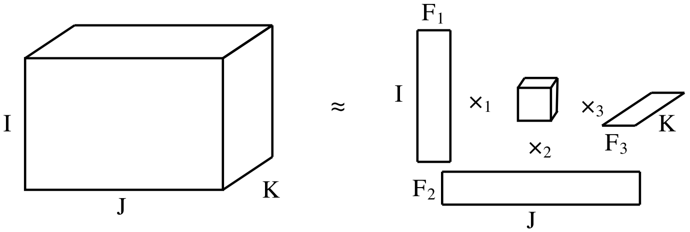
Notation

Tensors can be *unfolded* along one mode to matrix form: $\mathbf{Y}_{(n)}$.

- Mode n forms the rows and the remaining modes become columns.

Tucker decomposition

The Tucker decomposition models a tensor $\boldsymbol{\mathcal{X}}$ as a set of orthogonal factor matrices and a core tensor.



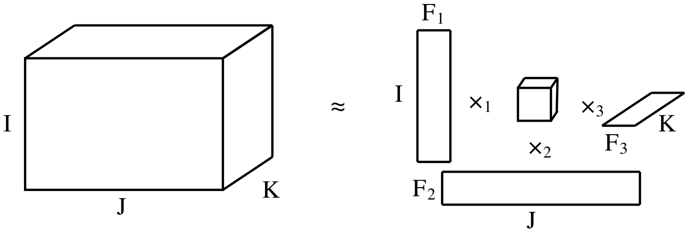
Notation

$\mathbf{A} \in \mathbb{R}^{I \times F_1}$, $\mathbf{B} \in \mathbb{R}^{J \times F_2}$, and $\mathbf{C} \in \mathbb{R}^{K \times F_3}$ denote the factor matrices.

$\boldsymbol{\mathcal{G}} \in \mathbb{R}^{F_1 \times F_2 \times F3}$ denotes the core tensor.

Tucker decomposition

The core tensor, $\mathcal{G}$, can be viewed as weights for the interactions between the low-rank factor matrices.



Elementwise:

$$\mathcal{X}(i,j,k) \approx \sum_{f_1=1}^{F_1} \sum_{f_2=1}^{F_2} \sum_{f_3=1}^{F_3} \mathcal{G}(f_1, f_2, f_3)\mathbf{A}(i, f_1)\mathbf{B}(j, f_2)\mathbf{C}(k, f_3)$$

Example Tucker applications

Dense: data compression

- The Tucker decomposition has long been used to compress (dense) tensor data (think truncated SVD).
- Folks at Sandia have had huge successes in compressing large simulation outputs[1].

Sparse: unstructured data analysis

- More recently, used to discover relationships in unstructured data.
- The resulting tensors are sparse and high-dimensional.
  - These large, sparse tensors are the focus of this talk.

[1] Woody Austin, Grey Ballard, and Tamara G Kolda. "Parallel tensor compression for large-scale scientific data". In: *International Parallel & Distributed Processing Symposium (IPDPS'16)*. IEEE. 2016, pp. 912–922.

Example: dimensionality reduction for clustering

Factor interpretation:

- Each row of a factor matrix represents an object from the original data.
- The ith object is a point in low-dimensional space: $\mathbf{A}(i,:)$.
- These points can be clustered, etc.

Example: dimensionality reduction for clustering

Factor interpretation:

- Each row of a factor matrix represents an object from the original data.
- The ith object is a point in low-dimensional space: $\mathbf{A}(i, :)$.
- These points can be clustered, etc.

Application: citation network analysis [Kolda & Sun, ICDM '08]

- A citation network forms an *author* $\times$ *conference* $\times$ *keyword* sparse tensor.
- The rows of the resulting factors are clustered with k-means to reveal relationships.

Authors: Jiawei Han, Christos Faloutsos, ...
Conferences: KDD, ICDM, PAKDD, ...
Keywords: knowledge, learning, reasoning

Table of Contents

Optimization problem

The resulting optimization problem is non-convex:

$$
\begin{aligned}
\underset{\mathbf{A},\mathbf{B},\mathbf{C},\mathcal{G}}{\text{minimize}} \quad & \frac{1}{2}\left\|\mathcal{X} - \left(\mathcal{G} \times_1 \mathbf{A}^T \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T\right)\right\|_F^2 \\
\text{subject to} \quad & \mathbf{A}^T\mathbf{A} = \mathbf{I} \\
& \mathbf{B}^T\mathbf{B} = \mathbf{I} \\
& \mathbf{C}^T\mathbf{C} = \mathbf{I}
\end{aligned}
$$

Higher-Order Orthogonal Iterations (HOOI)

HOOI is an alternating optimization algorithm.

Tucker Decomposition with HOOI

while not converged **do**
  $\boldsymbol{\mathcal{Y}}_1 \leftarrow \boldsymbol{\mathcal{X}} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$
  $\mathbf{A} \leftarrow F_1$ leading left singular vectors of $\mathbf{Y}_{(1)}$

  $\boldsymbol{\mathcal{Y}}_2 \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A}^T \times_3 \mathbf{C}^T$
  $\mathbf{B} \leftarrow F_2$ leading left singular vectors of $\mathbf{Y}_{(2)}$

  $\boldsymbol{\mathcal{Y}}_3 \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A}^T \times_2 \mathbf{B}^T$
  $\mathbf{C} \leftarrow F_3$ leading left singular vectors of $\mathbf{Y}_{(3)}$

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

Higher-Order Orthogonal Iterations (HOOI)

TTMc is the most expensive kernel in the HOOI algorithm.

Tucker Decomposition with HOOI

while not converged **do**
  $\mathcal{Y}_1 \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$
  $\mathbf{A} \leftarrow F_1$ leading left singular vectors of $\mathbf{Y}_{(1)}$
  $\mathcal{Y}_2 \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_3 \mathbf{C}^T$
  $\mathbf{B} \leftarrow F_2$ leading left singular vectors of $\mathbf{Y}_{(2)}$
  $\mathcal{Y}_3 \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_2 \mathbf{B}^T$
  $\mathbf{C} \leftarrow F_3$ leading left singular vectors of $\mathbf{Y}_{(3)}$
  $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$
end while

Intermediate memory blowup

A first step is to optimize a single TTM kernel and apply in sequence:

$$\mathcal{Y}_1 \leftarrow \left(\left(\mathcal{X} \times_2 \mathbf{B}^T \right) \times_3 \mathbf{C}^T \right)$$

Challenge:

- Intermediate results become more dense after each TTM.
- Memory overheads are dependent on sparsity pattern and factorization rank, but can be several orders of magnitude.

Tamara Kolda and Jimeng Sun. "Scalable tensor decompositions for multi-aspect data mining". In: *International Conference on Data Mining (ICDM)*. 2008.

Intermediate memory blowup

$$\mathcal{Y}_1 \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$$

Solutions:

[2] Tamara Kolda and Jimeng Sun. "Scalable tensor decompositions for multi-aspect data mining". In: *International Conference on Data Mining (ICDM)*. 2008.

[3] Oguz Kaya and Bora Uçar. *High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors*. Tech. rep. RR-8801. Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

Intermediate memory blowup

$$\boldsymbol{\mathcal{Y}}_1 \leftarrow \boldsymbol{\mathcal{X}} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$$

Solutions:

1. Tile over $\boldsymbol{\mathcal{Y}}_1$ to constrain blowup[2].
   - Requires multiple passes over the input tensor and many FLOPs.

[2] Tamara Kolda and Jimeng Sun. "Scalable tensor decompositions for multi-aspect data mining". In: *International Conference on Data Mining (ICDM)*. 2008.

[3] Oguz Kaya and Bora Uçar. *High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors*. Tech. rep. RR-8801. Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

Intermediate memory blowup

$$\mathcal{Y}_1 \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$$

Solutions:

1. Tile over $\mathcal{Y}_1$ to constrain blowup[2].
   - Requires multiple passes over the input tensor and many FLOPs.

2. Instead, fuse the TTMs and use a formulation based on non-zeros[3].
   - Only a single pass over the tensor!

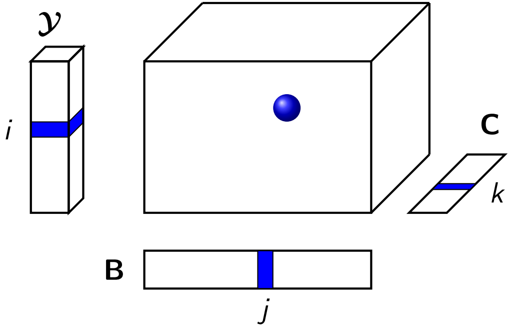
[2] Tamara Kolda and Jimeng Sun. "Scalable tensor decompositions for multi-aspect data mining". In: *International Conference on Data Mining (ICDM)*. 2008.

[3] Oguz Kaya and Bora Uçar. *High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors*. Tech. rep. RR-8801. Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

Elementwise formulation

Processing each non-zero individually has cost $\mathcal{O}(\text{nnz}(\boldsymbol{\mathcal{X}})F_2F_3)$ and $\mathcal{O}(F_2F_3)$ memory overhead.

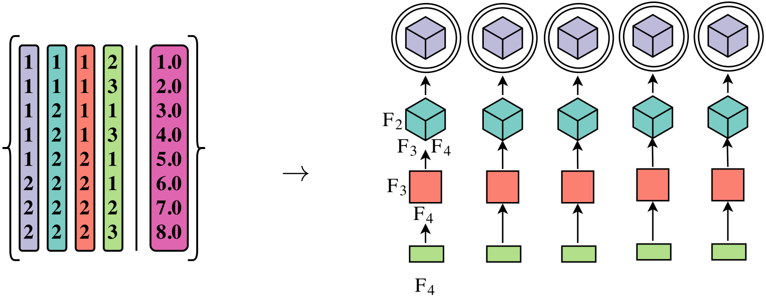
$$\boldsymbol{\mathcal{Y}}_1(i,:,:) \mathrel{+}= \boldsymbol{\mathcal{X}}(i,j,k)\,[\mathbf{B}(j,:) \circ \mathbf{C}(k,:)]$$



Oguz Kaya and Bora Uçar. *High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors*. Tech. rep. RR-8801. Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

TTMc with coordinate form

The elementwise formulation of TTMc naturally lends itself to a coordinate storage format:



Memoization

Some of the intermediate results across TTMc kernels can be reused:

$$\boldsymbol{\mathcal{Y}}_1 \leftarrow \boldsymbol{\mathcal{X}} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T \times_4 \mathbf{D}^T$$
$$\boldsymbol{\mathcal{Y}}_2 \leftarrow \boldsymbol{\mathcal{X}} \times_1 \mathbf{A}^T \times_3 \mathbf{C}^T \times_4 \mathbf{D}^T$$

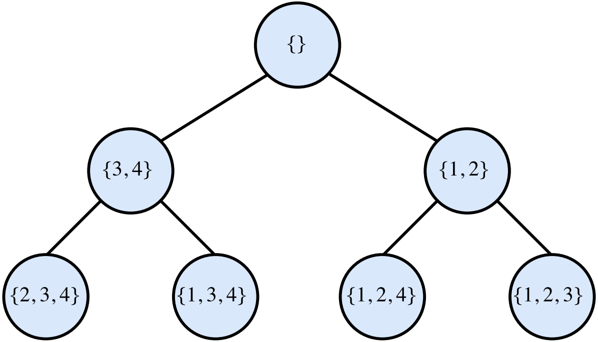
becomes:

$$\boldsymbol{\mathcal{Z}} \leftarrow \boldsymbol{\mathcal{X}} \times_3 \mathbf{C}^T \times_4 \mathbf{D}^T$$
$$\boldsymbol{\mathcal{Y}}_1 \leftarrow \boldsymbol{\mathcal{Z}} \times_2 \mathbf{B}^T$$
$$\boldsymbol{\mathcal{Y}}_2 \leftarrow \boldsymbol{\mathcal{Z}} \times_1 \mathbf{A}^T$$

Muthu Baskaran et al. "Efficient and scalable computations with sparse tensors". In: *High Performance Extreme Computing (HPEC)*. 2012.

TTMc with dimension trees

State-of-the-art TTMc:



Each node in the tree stores intermediate results from a set of modes.

Oguz Kaya and Bora Uçar. *High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors*. Tech. rep. RR-8801. Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

TTMc with dimension trees

Parallelism:

- Independent units of work within each node are indentified.
- For flat dimension trees, this equates to parallelizing over $\mathcal{Y}_1(i,:,:)$ slices.

Oguz Kaya and Bora Uçar. *High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors*. Tech. rep. RR-8801. Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

Table of Contents

Motivation

Existing algorithms either:

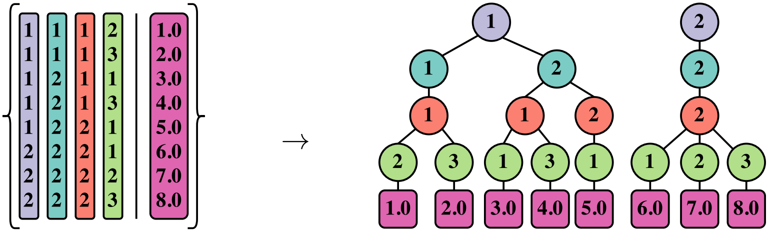
- have intermediate data blowup
- perform many operations
- trade memory for performance (i.e., memoization)
  - Overheads depend on the sparsity pattern and factorization rank

Can we accelerate TTMc without memory overheads?

Compressed Sparse Fiber (CSF)

CSF encodes a sparse tensor as a forest.

- Each path from root to leaf encodes a non-zero.
- CSF can be viewed as a generalization of CSR.



Shaden Smith and George Karypis. “Tensor-Matrix Products with a Compressed Sparse Tensor”. In: *5th Workshop on Irregular Applications: Architectures and Algorithms*. 2015.

Arithmetic redundancies in TTMc

Going back to the non-zero formulation:

$$\boldsymbol{\mathcal{Y}}_1(i,:,:) \mathrel{+}= \boldsymbol{\mathcal{X}}(i,j,k)\,[\mathbf{B}(j,:) \circ \mathbf{C}(k,:)]$$

There are two arithmetic redundancies we can exploit:

1. Distributive outer products
2. Redundant outer products

Distributive outer products

Consider two non-zeros in the same fiber $\boldsymbol{\mathcal{X}}(i, j, :)$

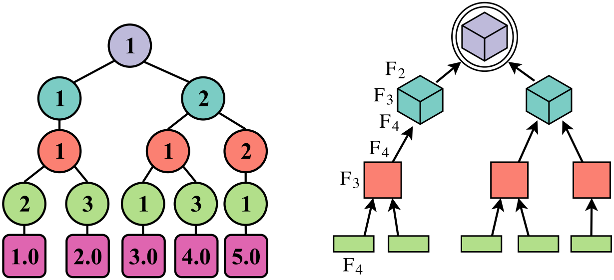
$$\boldsymbol{\mathcal{Y}}_1(i, :, :) \mathrel{+}= \boldsymbol{\mathcal{X}}(i, j, k_1) \left[\mathbf{B}(j, :) \circ \mathbf{C}(k_1, :)\right]$$
$$\boldsymbol{\mathcal{Y}}_1(i, :, :) \mathrel{+}= \boldsymbol{\mathcal{X}}(i, j, k_2) \left[\mathbf{B}(j, :) \circ \mathbf{C}(k_2, :)\right]$$

We can factor out $\mathbf{B}(j, :)$

$$\boldsymbol{\mathcal{Y}}_1(i, :, :) \mathrel{+}= \mathbf{B}(j, :) \circ \left[\boldsymbol{\mathcal{X}}(i, j, k_1)\mathbf{C}(k_1, :) + \boldsymbol{\mathcal{X}}(i, j, k_2)\mathbf{C}(k_2, :)\right]$$

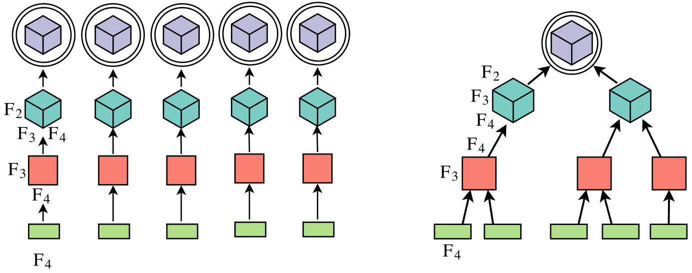
Distributive outer products with CSF

- Children nodes are summed and then expanded with an outer product.
- Intermediate memory is kept negligible with a post-order depth-first traversal.
  - We only need to keep a single stack of intermediate results.



Distributive outer products with CSF

Compare to computing with coordinate format:



Savings: The cost of each non-zero (leaf) is now linear in the rank.

Redundant outer products

Suppose we're now computing $\boldsymbol{\mathcal{Y}}_3$:

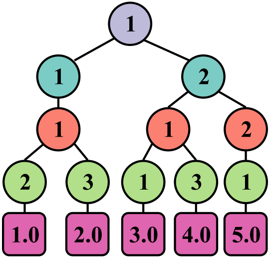
$$\boldsymbol{\mathcal{Y}}_3(:,:,k_1) \mathrel{+}= \boldsymbol{\mathcal{X}}(i,j,k_1)\left[\mathbf{A}(i,:) \circ \mathbf{B}(j,:)\right],$$
$$\boldsymbol{\mathcal{Y}}_3(:,:,k_2) \mathrel{+}= \boldsymbol{\mathcal{X}}(i,j,k_2)\left[\mathbf{A}(i,:) \circ \mathbf{B}(j,:)\right].$$

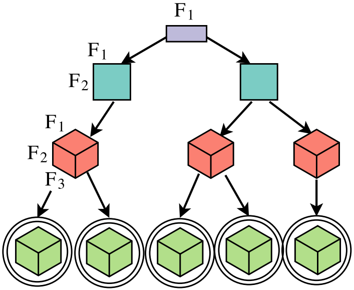
The outer product can be reused:

$$\begin{aligned} \mathbf{S} &\leftarrow \mathbf{A}(i,:) \circ \mathbf{B}(j,:) \\ \boldsymbol{\mathcal{Y}}_3(:,:,k_1) &\mathrel{+}= \boldsymbol{\mathcal{X}}(i,j,k_1) \cdot \mathbf{S} \\ \boldsymbol{\mathcal{Y}}_3(:,:,k_2) &\mathrel{+}= \boldsymbol{\mathcal{X}}(i,j,k_2) \cdot \mathbf{S} \end{aligned}$$

Redundant outer products with CSF

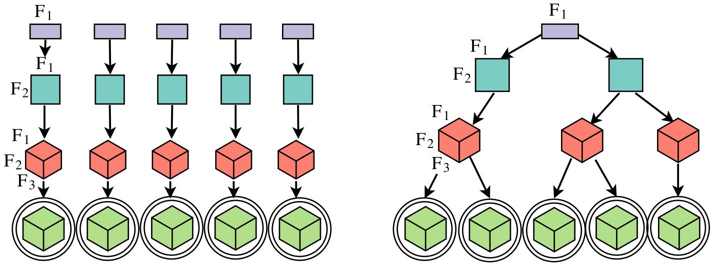
- Each child node uses its parent in an outer product.
- Use a pre-order depth-first traversal to manage memory.





Redundant outer products with CSF

Compare to computing with coordinate format:

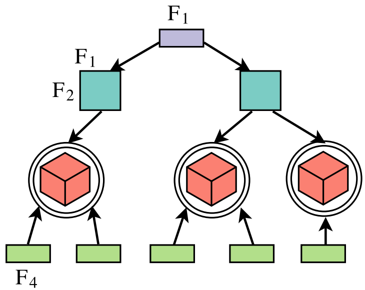


Savings: outer products are constructed less often, but non-zeros still have the same asymptotic cost as coordinate form.

Putting it all together

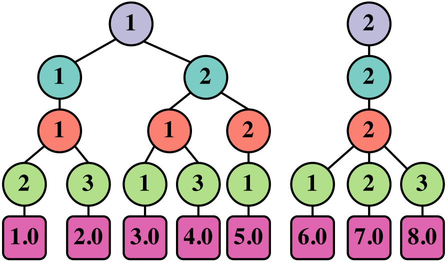
These two optimizations can be combined within the same kernel.

- Traversal is still depth-first
  - Use a pre-order traversal for levels above the mode of interest.
  - Use a post-order traversal for levels below the mode of interest.



Parallelism with CSF

We distribute trees to threads and use dynamic load balancing.



Race conditions are dependent on the mode of interest:

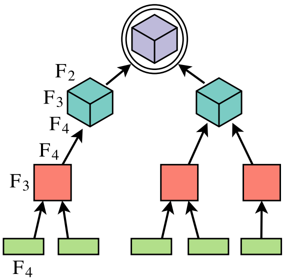
- Root nodes are unique, so no race conditions
- Otherwise, use a mutex to lock the slice of $\boldsymbol{\mathcal{Y}}$

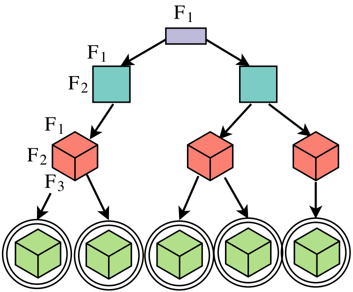
Table of Contents

Motivation

TTMc is significantly more expensive when computing for the lower-level modes.

- This is due to FLOPs and synchronization costs.

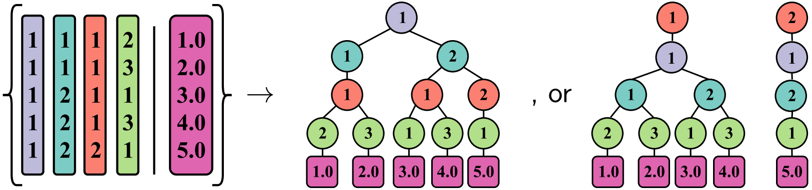




Multiple CSF representations

We can reorder the modes of $\mathcal{X}$ and store additional copies of the tensor.

- Selectively place modes near the top which were previously expensive.



CSF selection

Given storage for K copies of the tensor, how do we select them from the $N!$ mode possible orderings?

Greedy, heuristic algorithm:

- The first CSF always sorts the modes based on their lengths.
- For each of the $K - 1$ remaining CSF representations:
  - Select the mode which is estimated to be the most expensive based on FLOPs.
  - Place that mode at the top of the next CSF, with the remaining modes sorted by length.
  - Examine the new CSF and update the new best cost estimates.

Table of Contents

Experimental Setup

Source code:

- Part of the **S**urprisingly **P**aralle**L** sp**A**rse **T**ensor **T**oolkit[4]
- Written in C and parallelized with OpenMP
- Compiled with `icc` v16.0.3 and linked with Intel MKL

HyperTensor

- Implements dimenson tree-based methods
- Written in C++ and parallelized with OpenMP

Machine specifications:

- 2x 12-core Intel E5-2680v3 processors (Haswell)
- Double-precision floats and 32-bit integers

[4] SPLATT: `https://github.com/ShadenSmith/splatt`

Experimental Setup

Experiments

- All measurements are for a sequence of TTMc kernels forming one iteration.
- We fix $F_1 = F_2 = \ldots = 20$.

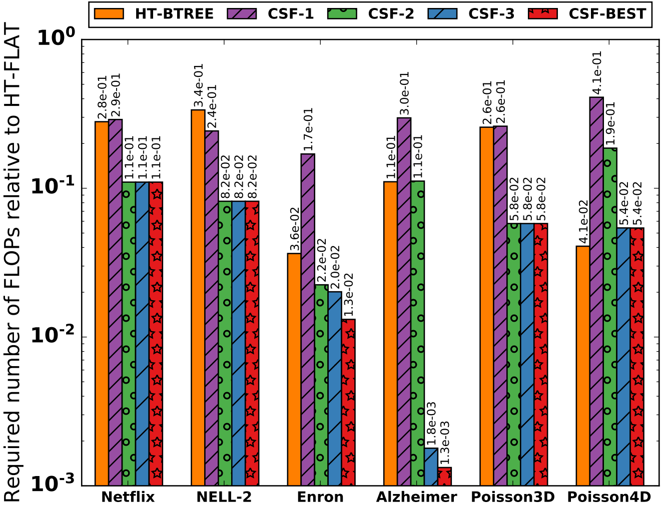
Datasets

Dataset	Non-zeros	Modes	Dimensions
`NELL-2`	77M	3	12K, 9K, 29K
`Netflix`	100M	3	480K, 18K, 2K
`Enron`	54M	4	6K, 6K, 244K, 1K
`Alzheimer`	6.27M	5	5, 1K, 156, 1K, 396
`Poisson3D`, `Poisson4D`	100M	3,4	3K, ..., 3K

K and **M** stand for thousand and million, respectively.

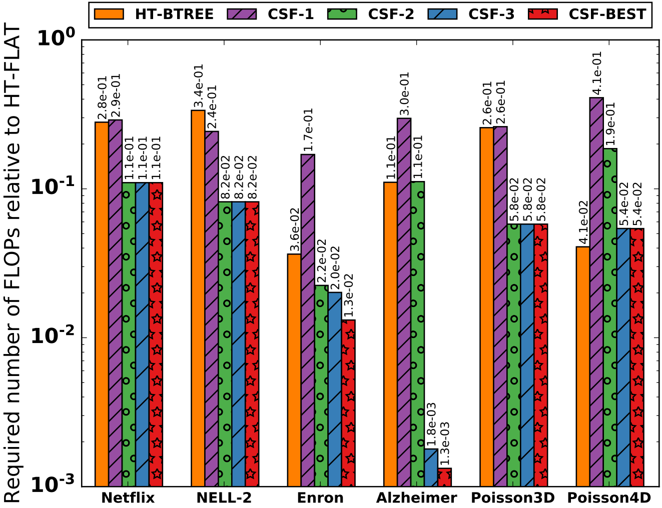
Relative FLOP cost of TTMc

The greedy algorithm usually matches or gets close to the best possible CSF configuration.



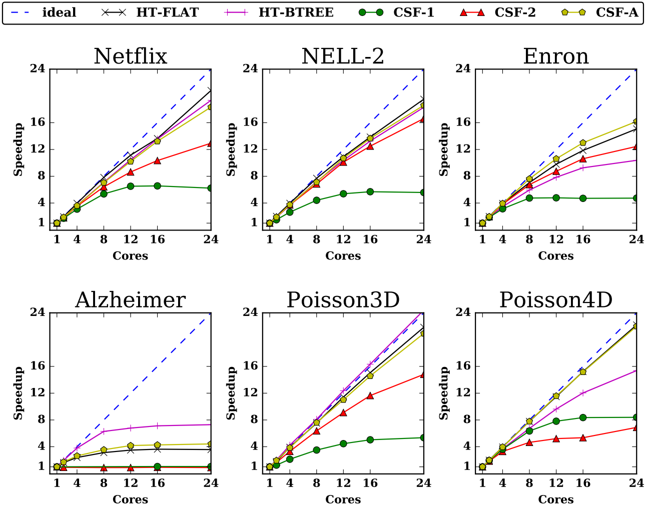
Relative FLOP cost of TTMc

CSF benefits more as dimensionality increases.



Parallel Scalability

Adding CSF representations improves scalability due to fewer and smaller critical regions.



Performance tradeoffs

Selecting the number of CSFs provides tuning for memory vs. speed.

- CSF always provides the options for the smallest and fastest executions.

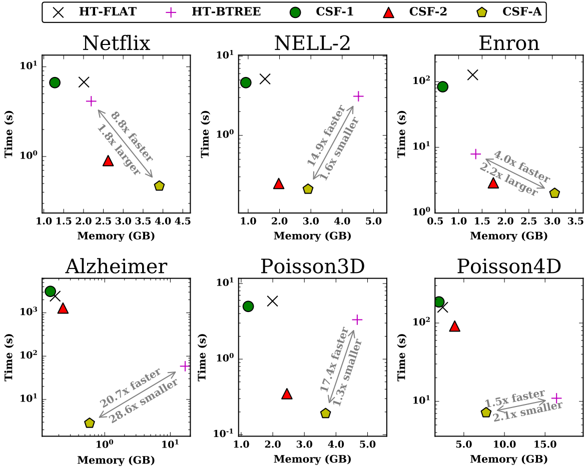


Table of Contents

Wrapping up

Contributions:

- We optimized TTMc kernels via a compressed tensor representation
  - CSF naturally exposes arithmetic redundancies in TTMc
- Multiple CSF tensors can further accelerate computation
  - Up to $20\times$ speedup over the state-of-the-art while using $28\times$ less memory
  - Choosing the number of data copies offers *tunable* computation/memory tradeoff

Wrapping up

Contributions:

- We optimized TTMc kernels via a compressed tensor representation
  - CSF naturally exposes arithmetic redundancies in TTMc
- Multiple CSF tensors can further accelerate computation
  - Up to $20\times$ speedup over the state-of-the-art while using $28\times$ less memory
  - Choosing the number of data copies offers *tunable* computation/memory tradeoff

Future work:

- Alternative decompositions to reduce synchronization costs
- Memoization is also applicable to CSF formulation!
  - *Li et al., IPDPS '17*

Reproducibility

All of our work is open source (to be updated soon):

`https://github.com/ShadenSmith/splatt`

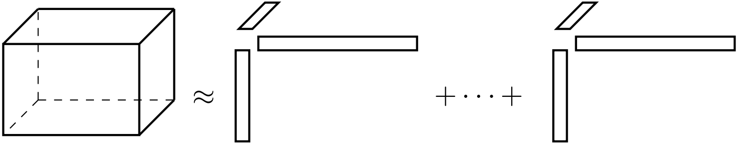
Datasets are freely available:

`http://frostt.io/`

Backup

Canonical polyadic decomposition (CPD)

The CPD models a tensor as the summation of rank-1 tensors.



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

Notation
$\mathbf{A} \in \mathbb{R}^{I \times F}$, $\mathbf{B} \in \mathbb{R}^{J \times F}$, and $\mathbf{C} \in \mathbb{R}^{K \times F}$ denote the factor matrices for a 3D tensor.

TTMc with a CSF tensor (1)

Algorithm 1 $\textsc{TTMc}()$

```
function TTMc(𝓧, mode)
    for i_1 = 1, ..., I_N in parallel do
        CONSTRUCT(𝓧(i_1, :, ..., :), mode, 1)
    end for
end function
```

TTMc with a CSF tensor (2)

Algorithm 2 CONSTRUCT()

▷ Construct Kronecker products and *push* them down to level *mode*−1.
function CONSTRUCT(*node*, *mode*, *above*)
    $d \leftarrow$ level(*node*) ▷ The level in the tree (i.e., distance from the root).
    $i_d \leftarrow$ node_id(*node*) ▷ The partial coordinate of a non-zero.

    if $d <$ mode **then**
        $\mathit{above} \leftarrow \mathit{above} \otimes \mathbf{A}^{(d)}(i_d, :)$
        for $c \in$ children(*node*) **do**
            CONSTRUCT(c, mode, *above*)
        end for

    else if $d =$ mode **then**
        $\mathit{below} \leftarrow \sum_{c \in \mathrm{children}(\mathit{node})} \mathrm{ACCUMULATE}(c)$
        Lock mutex i_d.
        $\mathbf{Y}_{(d)}(i_d, :) \leftarrow \mathbf{Y}_{(d)}(i_d, :) + (\mathit{above} \otimes \mathit{below})$ ▷ Update $\mathbf{Y}_{(d)}$.
        Unlock mutex i_d.
    end if
end function

TTMc with a CSF tensor (3)

Algorithm 3 ACCUMULATE()

$\triangleright$ *Pull* Kronecker products up from the leaf nodes.

function ACCUMULATE(*node*)
    $i_d \leftarrow \text{node\_id}(node)$
    if $\text{level}(node) = N$ **then**
        return $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_d) \cdot \mathbf{A}^{(N)}(i_d, :)$
    else
        return $\mathbf{A}^{(d)}(i_d, :) \otimes \sum_{c \in \text{children}(node)} \text{ACCUMULATE}(c)$
    end if
end function