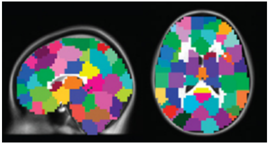
Parallel Algorithms for CP, Tucker, and Tensor Train Decompositions

Grey Ballard

PACO 2019: 3rd Workshop on Power-Aware Computing
MPI Magdeburg
Nov 6, 2019

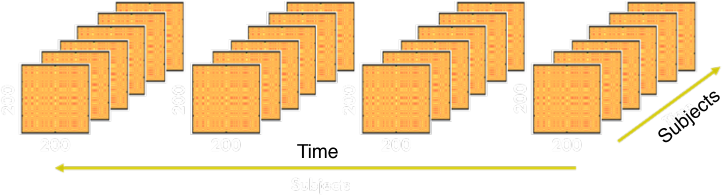
WAKE FOREST
UNIVERSITY

Motivation: multidimensional data analysis requires scalable algorithms



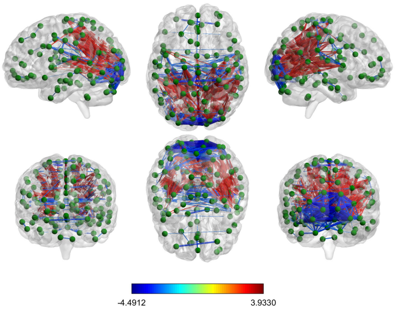
Dynamic functional connectivity fMRI data

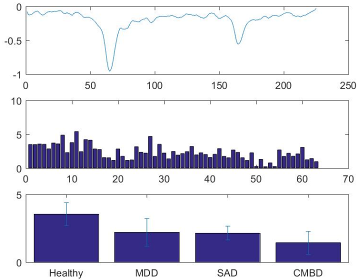
- measures correlation between regions of the brain over time
- experiments can include cognitive task
- study multiple subjects across groups



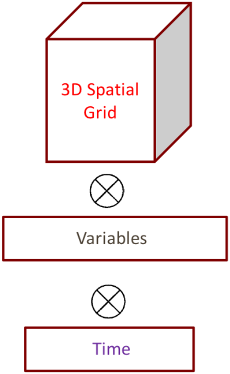
200 regions $\times$ 200 regions $\times$ 225 time steps $\times$ 59 subjects
4 GB of data

CP decomposition discovers patterns of synchronization across brain networks





Motivation: Numerical simulations producing more data than we can handle

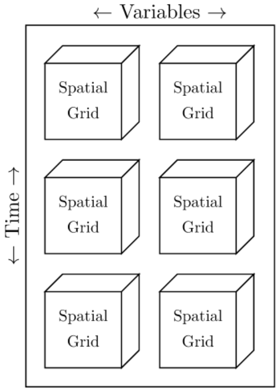


$512 \times 512 \times 512$ 3D grid,
128 time steps, 64 variables:
8 terabytes of data
(double precision)

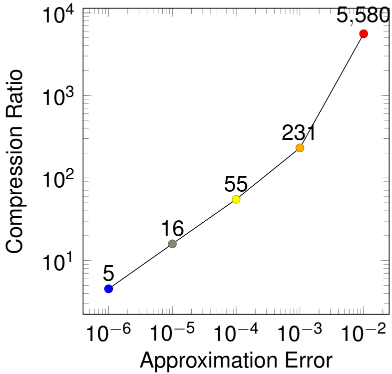
S3D MPI-based Combustion Code

- direct numerical simulation of engine combustion
- run on supercomputers
- single experiment produces terabytes of data
- storage resolution much less than computed resolution
- difficult to analyze or even transfer data

Tucker decomposition yields huge compression for combustion simulation data



Natural five-way multiway structure of scientific data



Compression rates as fidelity varies for 550GB simulation dataset

Motivation: what if you have to solve many PDEs?

A single PDE simulation can already create a ton of data...
what if we have design/uncertain parameters?

Suppose you have 10 parameters, each with 10 possible values

- now you have to run your simulation 10^{10} times...
- and store all this data...

If the resulting data could be compressed, why not compute the compressed representation from the start?

Tensor Train (TT) can break “curse of dimensionality”

For N-way problems that exhibit this compressible structure, the Tensor Train format can reduce the number of parameters from *exponential* to *linear* in N

- e.g., Tucker reduces I^N data to $O(R^N)$ for some small R
- e.g., TT reduces I^N data to $O(NIL^2)$ for some other small L

For moderately large N, full format is typically infeasible

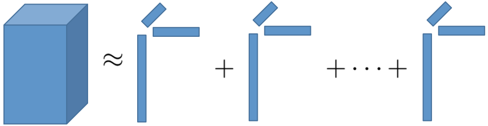
- start in TT format, perform arithmetic in TT format
- key is to maintain low ranks using rounding procedure
- TT makes some very high dimensional problems tractable

Parallel Computation of CP Decompositions with Nonnegativity Constraints

joint work with Srinivas Eswar[1], Koby Hayashi[1], Ramakrishnan Kannan[2], and Haesun Park[1]

[1] Georgia Tech
[2] Oak Ridge National Lab

CP Notation



$$\mathcal{X} \approx \mathbf{u}_1 \circ \mathbf{v}_1 \circ \mathbf{w}_1 + \cdots + \mathbf{u}_R \circ \mathbf{v}_R \circ \mathbf{w}_R, \qquad \mathcal{X} \in \mathbb{R}^{I \times J \times K}$$

$$\mathcal{X} \approx [\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]\,, \qquad \mathbf{U} \in \mathbb{R}^{I \times R}, \mathbf{V} \in \mathbb{R}^{J \times R}, \mathbf{W} \in \mathbb{R}^{K \times R}$$

are *factor matrices*

$$x_{ijk} \approx \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr}\,, \qquad 1 \leq i \leq I, 1 \leq j \leq J, 1 \leq k \leq K$$

Notation convention: scalar dimension N, index n with $1 \leq n \leq N$

Alternating Optimization (AO)

Fixing all but one factor matrix, we have a linear nonnegative least squares (NNLS) problem:

$$\operatorname*{arg\,min}_{\mathbf{V}\geq 0} \left\| \mathcal{X} - \sum_{r=1}^{R} \hat{\mathbf{u}}_r \circ \mathbf{v}_r \circ \hat{\mathbf{w}}_r \right\|$$

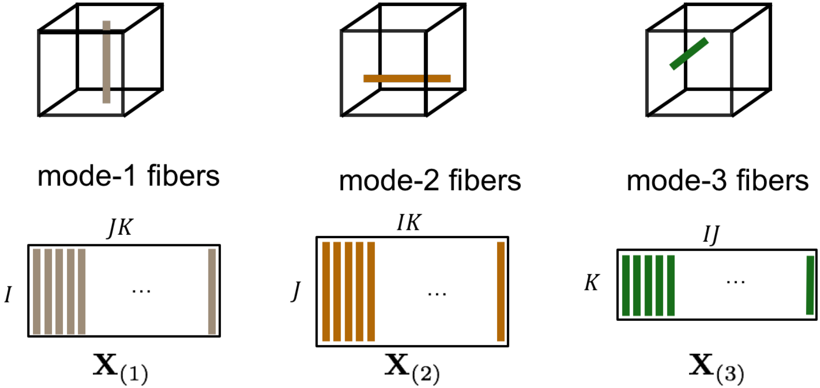
or equivalently

$$\operatorname*{arg\,min}_{\mathbf{V}\geq 0} \left\| \mathbf{X}_{(2)} - \mathbf{V}(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})^{\mathsf{T}} \right\|_F$$

$\odot$ is the *Khatri-Rao* product, a column-wise Kronecker product

AO works by alternating over factor matrices, updating one at a time by solving the corresponding linear NNLS problem

Matricization/Unfolding: Viewing a tensor as a matrix



Nonnegative (Linear) Least Squares

Our subproblem:

$$\arg\min_{\mathbf{V}\geq 0} \left\| \mathbf{X}_{(2)} - \mathbf{V}(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})^{\mathsf{T}} \right\|_F$$

Many possible NNLS algorithms

- Multiplicative Updates [LS99]
- Hierarchical Alternating Least Squares [CZPA09]
- Block Principal Pivoting [KP11]
- Alternating Direction Method of Multipliers [LS15]
- Nesterov-type Algorithm [LKL+17]

Nonnegative (Linear) Least Squares

Our subproblem:

$$\underset{\mathbf{V}\geq 0}{\arg\min} \left\| \mathbf{X}_{(2)} - \mathbf{V}(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})^{\mathsf{T}} \right\|_F$$

Most NNLS algorithms are bottlenecked by computing

$$\mathbf{X}_{(2)}(\hat{\mathbf{W}} \odot \hat{\mathbf{U}}) \quad \text{and} \quad (\hat{\mathbf{W}} \odot \hat{\mathbf{U}})^T(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})$$

- $\mathbf{X}_{(2)}(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})$ is called Matricized-Tensor Times Khatri-Rao Product (MTTKRP) and is expensive to compute
- $(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})^T(\hat{\mathbf{W}} \odot \hat{\mathbf{U}})$ can be computed relatively cheaply as $\hat{\mathbf{W}}^T\hat{\mathbf{W}} * \hat{\mathbf{U}}^T\hat{\mathbf{U}}$, where $*$ is elementwise product

Parallelizing MTTKRP

Our goal is to perform MTTKRP in parallel as fast as possible

- How do we distribute the tensor across processors?
- How do we distribute the matrices across processors?
- How do we divide up the computation?
- How much interprocessor communication will that require?

Parallel Communication Lower Bound

Theorem ([BKR18])

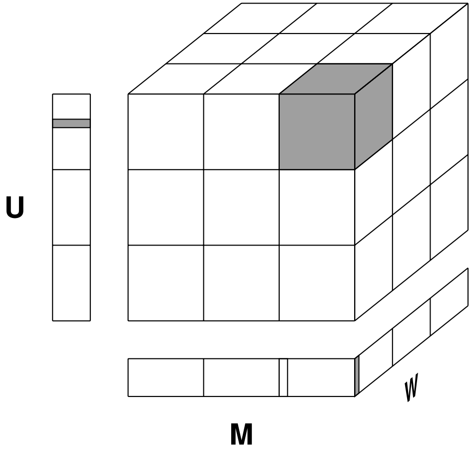
Any parallel MTTKRP algorithm involving a tensor with $I_n = I^{1/N}$ for all n and that evenly distributes one copy of the input and output performs at least

$$\Omega\left(\left(\frac{NIR}{P}\right)^{\frac{N}{2N-1}} + NR\left(\frac{I}{P}\right)^{1/N}\right)$$

sends and receives. (Second term will typically dominate.)

- N is the number of modes
- I is the number of tensor entries
- I_n is the dimension of the nth mode
- R is the rank of the CP model
- P is the number of processors

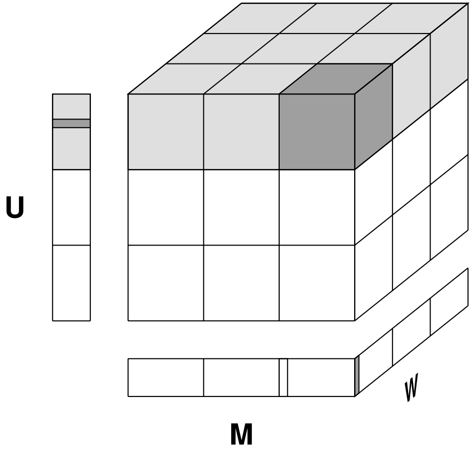
Communication-Optimal Parallel Algorithm (3D)



Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix

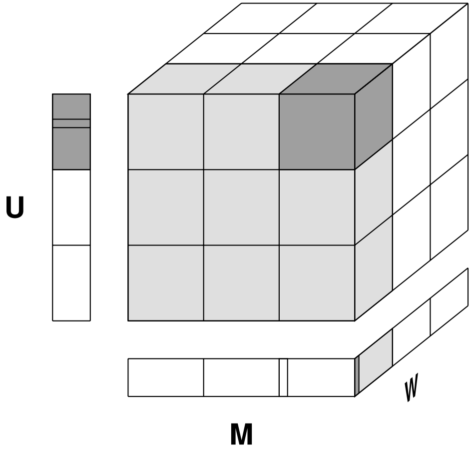
Communication-Optimal Parallel Algorithm (3D)



Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix
2. All-Gathers all the rows needed from **U**

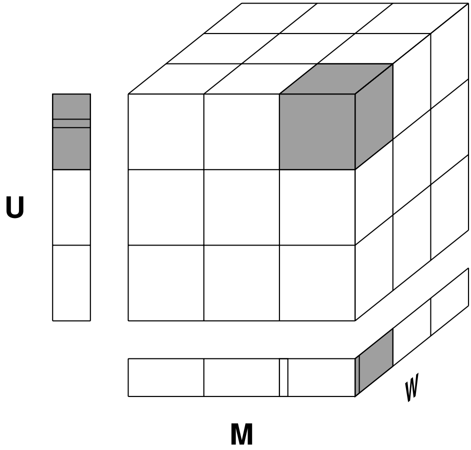
Communication-Optimal Parallel Algorithm (3D)



Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix
2. All-Gathers all the rows needed from **U**
3. All-Gathers all the rows needed from **W**

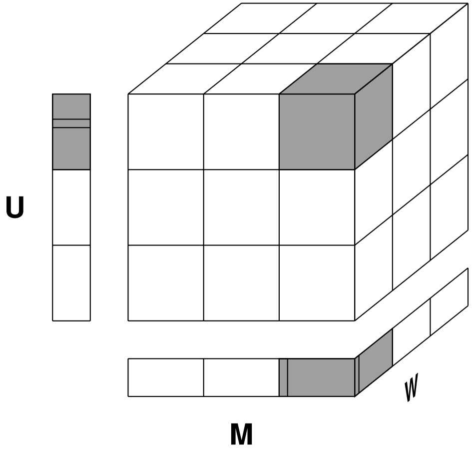
Communication-Optimal Parallel Algorithm (3D)



Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix
2. All-Gathers all the rows needed from **U**
3. All-Gathers all the rows needed from **W**
4. Computes its contribution to rows of **M** (local MTTKRP)

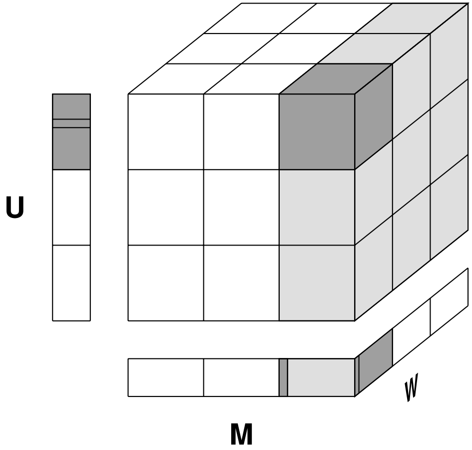
Communication-Optimal Parallel Algorithm (3D)



Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix
2. All-Gathers all the rows needed from **U**
3. All-Gathers all the rows needed from **W**
4. Computes its contribution to rows of **M** (local MTTKRP)

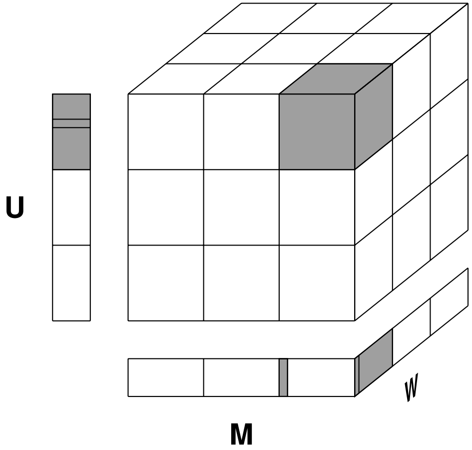
Communication-Optimal Parallel Algorithm (3D)



Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix
2. All-Gathers all the rows needed from **U**
3. All-Gathers all the rows needed from **W**
4. Computes its contribution to rows of **M** (local MTTKRP)
5. Reduce-Scatters to compute and distribute **M** evenly

Communication-Optimal Parallel Algorithm (3D)



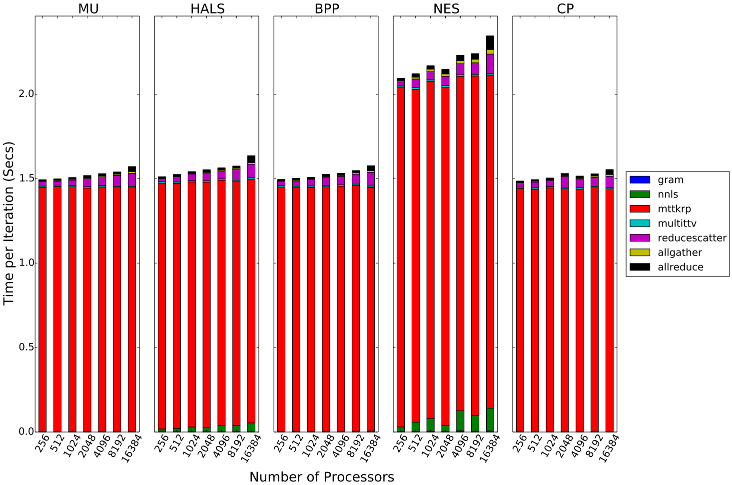
Each processor

1. Starts with one subtensor and subset of rows of each input factor matrix
2. All-Gathers all the rows needed from **U**
3. All-Gathers all the rows needed from **W**
4. Computes its contribution to rows of **M** (local MTTKRP)
5. Reduce-Scatters to compute and distribute **M** evenly
6. Use **M** to solve NNLS problem for **V**

Rest of the Algorithm

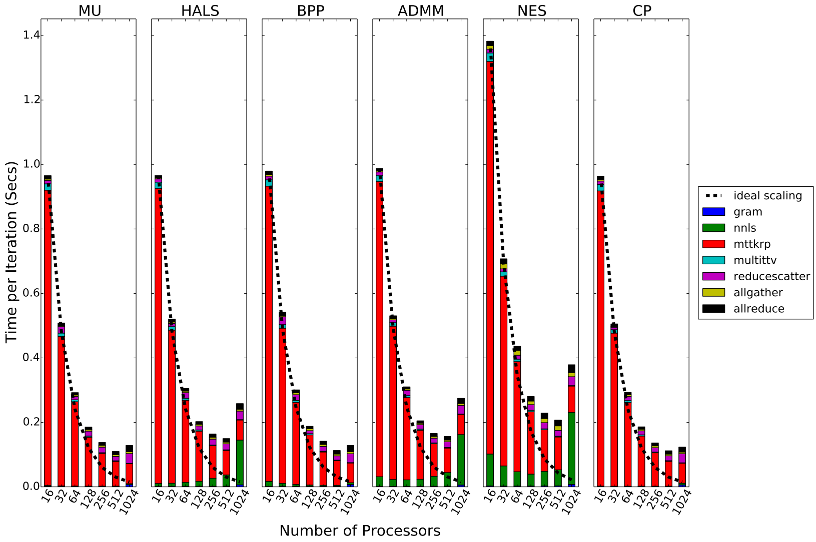
- With correct processor grid, MTTKRP algorithm achieves communication lower bound
- Also need to compute $\mathbf{G} = \mathbf{U}^T\mathbf{U} * \mathbf{W}^T\mathbf{W}$
  - involves communication
  - generally lower order cost
- Lots of overlap across MTTKRP computations
  - save communication: keep temporary copies around
  - save computation: use dimension tree optimization
  - $O(N)$ savings, where N is the number of modes
- Can choose algorithm to compute **V** from **M** and **G**
  - for some algorithms, this is all local computation
  - some algorithms require extra computation of global information, can add significant cost

Weak Scaling Results for 4D Synthetic Data



- local tensor is fixed at $128 \times 128 \times 128 \times 128$

Strong Scaling Results for Mouse Brain Data



Software: PLANC

Parallel Low-rank Approximations with Non-negativity Constraints

`https://github.com/ramkikannan/planc`

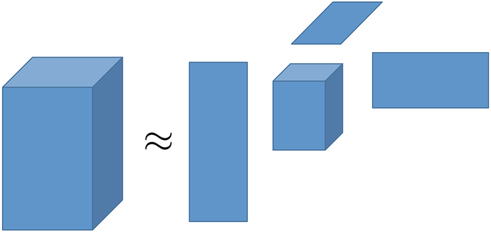
- Open source code for computing NMF and NNCP
- MPI/BLAS/LAPACK/C++11
- Designed for dense tensors and dense/sparse matrices
- Can offload computation to GPUs if available

Efficient Parallel Algorithm for Tucker Decompositions of Dense Tensors

joint work with Woody Austin[4], Alicia Klinvex[5], Tammy Kolda[6], and Hemanth Kolla[6]

[4] UT Austin
[5] Bettis Atomic Power Laboratory
[6] Sandia National Labs

Tucker Notation



$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{U} \times_2 \mathbf{V} \times_3 \mathbf{W}$$

$\mathcal{X} \in \mathbb{R}^{I \times J \times K}, \mathcal{G} \in \mathbb{R}^{P \times Q \times R}$
is *core tensor*

$$\mathcal{X} \approx [\![\mathcal{G}; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!],$$

$\mathbf{U} \in \mathbb{R}^{I \times P}, \mathbf{V} \in \mathbb{R}^{J \times Q}, \mathbf{W} \in \mathbb{R}^{K \times R}$
are *factor matrices*

$$x_{ijk} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} u_{ip} v_{jq} w_{kr}, \quad 1 \le i \le I, 1 \le j \le J, 1 \le k \le K$$

Algorithm: ST-HOSVD [VVM12]

ST-HOSVD($\mathcal{X}, \varepsilon$)

1. Compute $\mathbf{U}$ with dimension $I \times P$
   (a) Compute Gram matrix $\mathbf{X}_{(1)}\mathbf{X}_{(1)}^T$
   (b) Use eigendecomposition to determine P and $\mathbf{U}$
   (c) TTM to shrink to size $P \times J \times K$: $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{U}^T$
2. Compute $\mathbf{V}$ with dimension $J \times Q$
   (a) Compute Gram matrix $\mathbf{Y}_{(2)}\mathbf{Y}_{(2)}^T$
   (b) Use eigendecomposition to determine Q and $\mathbf{V}$
   (c) TTM to shrink to size $P \times Q \times K$: $\mathcal{Z} = \mathcal{Y} \times_2 \mathbf{V}^T$
3. Compute $\mathbf{W}$ with dimension $K \times R$
   (a) Compute Gram matrix $\mathbf{Z}_{(3)}\mathbf{Z}_{(3)}^T$
   (b) Use eigendecomposition to determine R and $\mathbf{W}$
   (c) TTM to shrink to size $P \times Q \times R$: $\mathcal{G} = \mathcal{Z} \times_3 \mathbf{W}^T$

Key kernels

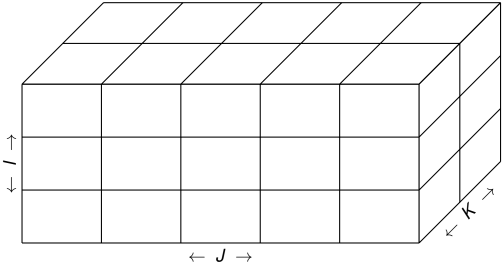
Key kernels of ST-HOSVD are

- Gram: short, fat matrix times its transpose ($\mathbf{X}_{(1)}\mathbf{X}_{(1)}^T$)
- Evecs: eigendecomposition of small symmetric matrix
- TTM: tensor times matrix to shrink problem ($\mathbf{U}^T\mathbf{X}_{(1)}$)

Our goal is to parallelize Gram and TTM efficiently

Tensor data distribution across processors

For N-way tensor, we use N-way processor grid with Cartesian block distribution (same as for CP)

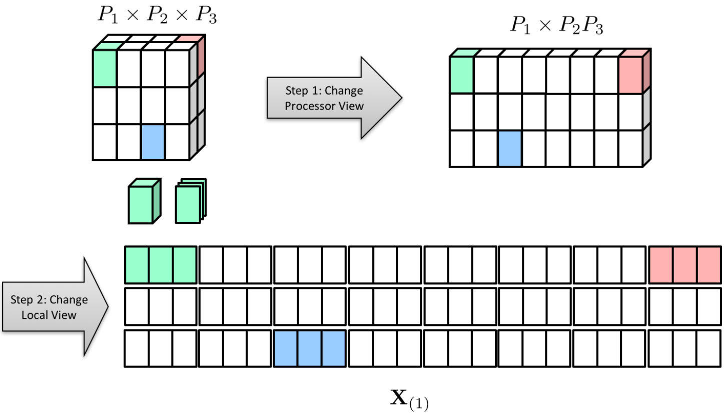


Example: $P_1 \times P_2 \times P_3 = 3 \times 5 \times 2$

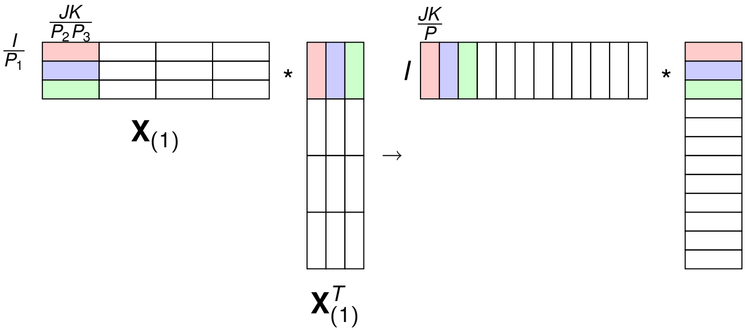
Local tensor size: $\dfrac{I}{P_1} \times \dfrac{J}{P_2} \times \dfrac{K}{P_3}$

Parallel matricization

Matricizing distributed tensor requires no data movement: matricized tensor already in standard matrix distribution

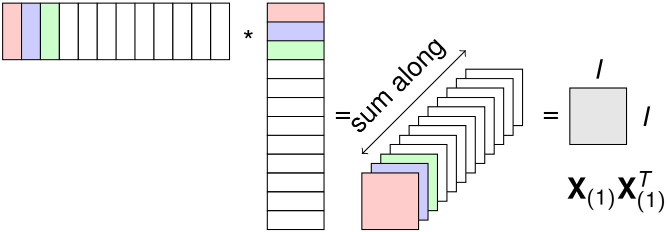


Parallel Gram Computation



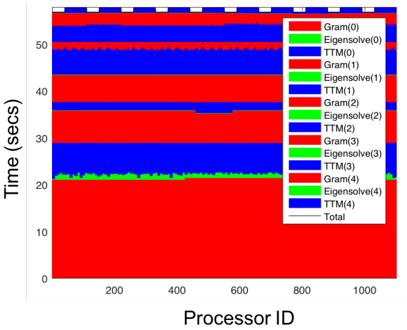
- each processor column redistributes its tensor data

Parallel Gram Computation



- each processor column redistributes its tensor data
- each processor computes local outer product
- sum across all processors via All-Reduce

Time Breakdown of Parallel ST-HOSVD



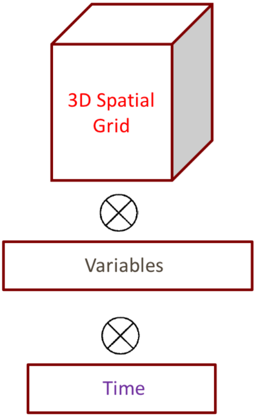
Parallel running time example

- 5-way tensor of size 4.4 TB
- reduced to 10 GB (410X)
- 1100 processors (cores)
- 55 seconds total

Observations

- load-balanced execution
- cycle of Gram-Eig-TTM shrinks over time
- writing original tensor to disk is slower by 10X

Combustion Simulation (S3D) Data



Stat-Planar dataset

- $500 \times 500 \times 500 \times 11 \times 400$
- 4.4 TB of total storage
- use 250 nodes to process

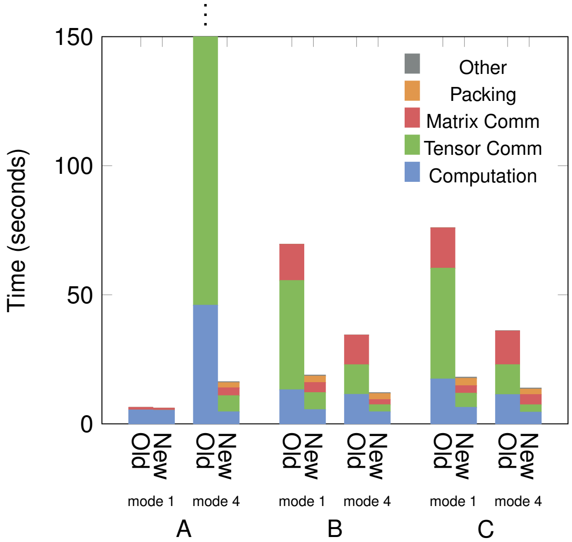
Two compression scenarios

- High: 1e-2 error, 20,000X comp.
- Low: 1e-4 error, 400X comp.

Three processor grids

- A: $1\times1\times40\times1\times100$
- B: $10\times8\times5\times1\times10$
- C: $40\times10\times1\times1\times10$

Gram Algorithm Comparison



- Old algorithm from our previous work [ABK16]

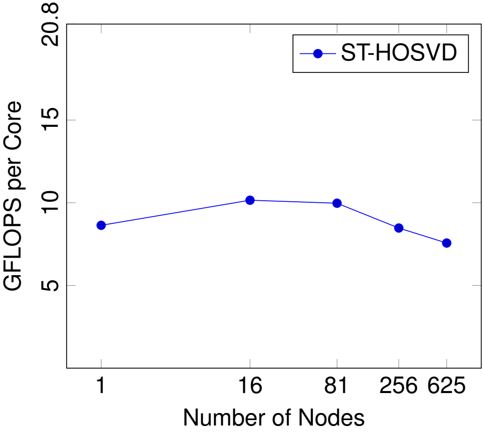
Weak Scaling on Synthetic Data

Problem Setup

- local tensor fixed at
  $200 \times 200 \times 200 \times 200$
- local core fixed at
  $20 \times 20 \times 20 \times 20$

Result

- as problem size grows with number of processors,
  high efficiency maintained up to 10K cores



Software: TuckerMPI

`https://gitlab.com/tensors/TuckerMPI`

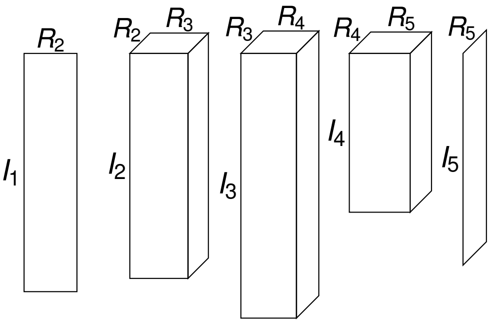
- Open source code for computing Tucker compression
- MPI/BLAS/LAPACK/C++11
- Designed for dense tensors

Communication-Efficient Parallel Algorithms for Tensor Train Rounding

joint work with Hussam Al Daas[7] and Peter Benner[7]

[7] MPI Magdeburg

Tensor Train Notation



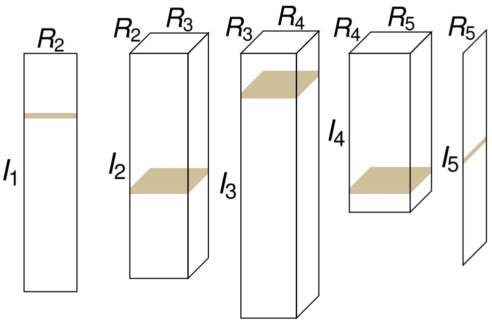
$\mathcal{X} \approx \{\mathcal{T}_{X,k}\}, \mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4 \times I_5}$

$\mathcal{T}_{X,k} \in \mathbb{R}^{R_k \times I_k \times R_{k+1}}$ are *TT cores*

$$x_{ijklm} \approx \sum_{\alpha=1}^{R_2} \sum_{\beta=1}^{R_3} \sum_{\gamma=1}^{R_4} \sum_{\delta=1}^{R_5} \mathcal{T}_{X,1}(i,\alpha)\mathcal{T}_{X,2}(\alpha,j,\beta)\mathcal{T}_{X,3}(\beta,k,\gamma)\mathcal{T}_{X,4}(\gamma,l,\delta)\mathcal{T}_{X,5}(\delta,m)$$

$\mathcal{V}(\mathcal{T}_{\mathcal{X},k}) \in \mathbb{R}^{R_k I_k \times R_{k+1}}$ and $\mathcal{H}(\mathcal{T}_{\mathcal{X},k}) \in \mathbb{R}^{R_k \times I_k R_{k+1}}$ are vertical and horizontal unfoldings of kth core

Tensor Train Notation



$\mathfrak{X} \approx \{\mathfrak{T}_{X,k}\}, \mathfrak{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4 \times I_5}$

$\mathfrak{T}_{X,k} \in \mathbb{R}^{R_k \times I_k \times R_{k+1}}$ are *TT cores*

$$x_{ijklm} \approx \sum_{\alpha=1}^{R_2} \sum_{\beta=1}^{R_3} \sum_{\gamma=1}^{R_4} \sum_{\delta=1}^{R_5} \mathfrak{T}_{X,1}(i,\alpha)\mathfrak{T}_{X,2}(\alpha,j,\beta)\mathfrak{T}_{X,3}(\beta,k,\gamma)\mathfrak{T}_{X,4}(\gamma,l,\delta)\mathfrak{T}_{X,5}(\delta,m)$$

$\mathcal{V}(\mathfrak{T}_{\mathfrak{X},k}) \in \mathbb{R}^{R_k I_k \times R_{k+1}}$ and $\mathcal{H}(\mathfrak{T}_{\mathfrak{X},k}) \in \mathbb{R}^{R_k \times I_k R_{k+1}}$ are vertical and horizontal unfoldings of kth core

TT Rounding

Given a tensor in TT format, want to compress the ranks

- algebraic operations on TT formats over-extend ranks
- recompression subject to some error threshold

Rounding done in two phases: orthogonalization and truncation

- orthogonalization done core-by-core in sequence
- truncation is done core-by-core (opposite direction)

TT Rounding

Given a tensor in TT format, want to compress the ranks

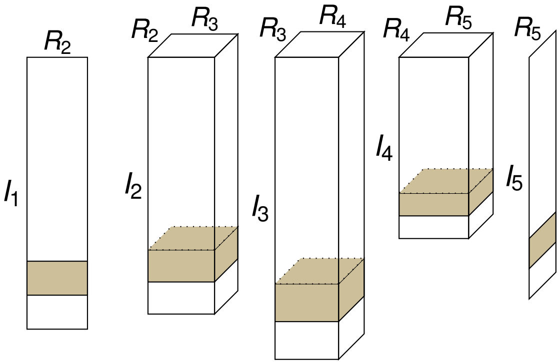
- algebraic operations on TT formats over-extend ranks
- recompression subject to some error threshold

Rounding done in two phases: orthogonalization and truncation

- orthogonalization done core-by-core in sequence
- truncation is done core-by-core (opposite direction)

for $n = N$ down to 2 **do**
  $L \cdot \mathcal{H}(\mathfrak{Q}) = \mathcal{H}(\mathfrak{T}_{\mathcal{X},n})$ ▷ LQ factorization of short-fat matrix
  $\mathfrak{T}_{\mathcal{X},n} = \mathfrak{Q}$
  $\mathcal{V}(\mathfrak{T}_{\mathcal{X},n-1}) = \mathcal{V}(\mathfrak{T}_{\mathcal{X},n-1})L$
for $n = 1$ to $N-1$ **do**
  $\mathcal{V}(\mathfrak{U}_n) \cdot \Sigma_n \cdot V_n^T = \mathcal{V}(\mathfrak{T}_{\mathcal{X},n})$ ▷ truncated SVD of tall-skinny matrix
  $\mathfrak{T}_{\mathcal{X},n} = \mathfrak{U}_n$
  $\mathcal{H}(\mathfrak{T}_{\mathcal{X},n+1}) = \Sigma_n V_n^\top \mathcal{H}(\mathfrak{T}_{\mathcal{X},n+1})$

Parallel Distribution

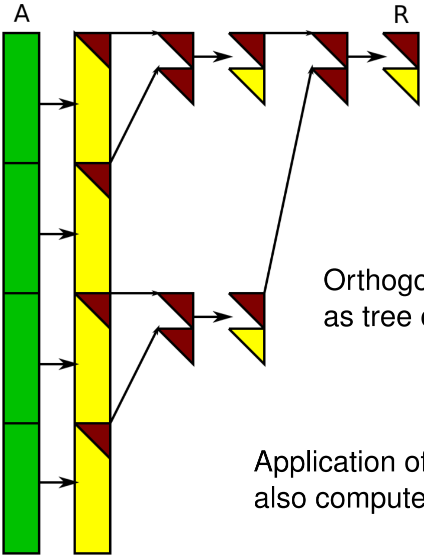


- Each core distributed across all P processors
- Local nth core dimensions are $R_n \times \frac{I_n}{P} \times R_{n+1}$
- Vertical and horizontal unfoldings are 1D-distributed

Parallel TT Rounding Algorithm

function $\{\mathcal{T}^{(p)}_{\mathcal{Y},n}\} = \textsc{Par-TT-Rounding}(\{\mathcal{T}^{(p)}_{\mathcal{X},n}\})$
  for $n = N$ down to 2 **do**
    $[\{Y^{(\ell)}_p\}_n, R_n] = \textbf{TSQR}(\mathcal{H}(\mathcal{T}^{(p)}_{\mathcal{X},n})^T)$ ▷ QR factorization
    $R^{(p)} = \textbf{Broadcast}(R_n, \text{root})$ ▷ Broadcast R to all procs
    $\mathcal{V}(\mathcal{T}^{(p)}_{\mathcal{X},n-1}) = \textbf{Mult}(\mathcal{V}(\mathcal{T}^{(p)}_{\mathcal{X},n-1}), R^{(p)})$ ▷ Apply R to previous core
  $\mathcal{Y} = \mathcal{X}$
  for $n = 1$ to $N - 1$ **do**
    $[\{Y^{(\ell)}_p\}_n, R_n] = \textbf{TSQR}(\mathcal{V}(\mathcal{T}^{(p)}_{\mathcal{Y},n}))$ ▷ QR factorization
    if $p =$ root **then**
      $[\hat{U}_R, \hat{\Sigma}, \hat{V}] = \textbf{tSVD}(R_n)$ ▷ Truncated SVD of R
    $\mathcal{V}(\mathcal{T}^{(p)}_{\mathcal{Y},n}) = \textbf{TSQR-Apply-Q}(\{Y^{(\ell)}_p\}_n, \hat{U}_R)$ ▷ Form explicit $\hat{U}$
    $\mathcal{H}(\mathcal{T}^{(p)}_{\mathcal{X},n+1})^T = \textbf{TSQR-Apply-Q}(\{Y^{(\ell)}_p\}_{n+1}, \hat{V})$ ▷ Apply $\hat{V}$ to next core

Tall-Skinny QR (TSQR) Algorithm [DGHL12]



Key benefit of TSQR: one parallel reduction

Orthogonal factor stored implicitly as tree of Householder vectors

Application of implicit orthogonal factor also computed via tree (backwards)

Cost Analysis of TT-Rounding

Assuming $I_n = I$ and $R_n = R$, and R is reduced by factor of 2...
computational cost is

$$6\frac{NIR^3}{P} + O(NR^3 \log P) \text{ flops}$$

communication cost is

$$O(NR^2 \log P) \text{ words and } O(N \log P) \text{ messages}$$

Cost Analysis of TT-Rounding

Assuming $I_n = I$ and $R_n = R$, and R is reduced by factor of 2...
computational cost is

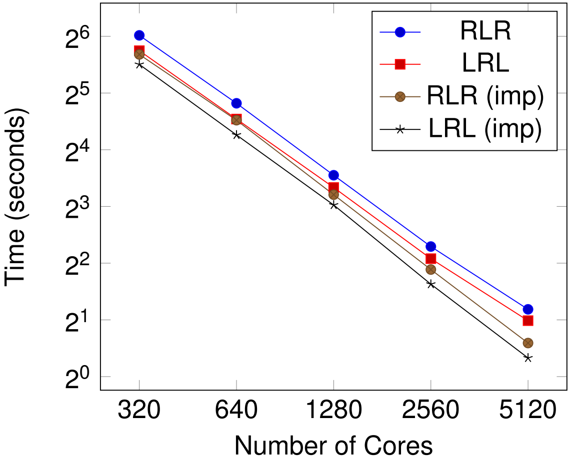
$$6\frac{NIR^3}{P} + O(NR^3 \log P) \text{ flops}$$

communication cost is

$$O(NR^2 \log P) \text{ words and } O(N \log P) \text{ messages}$$

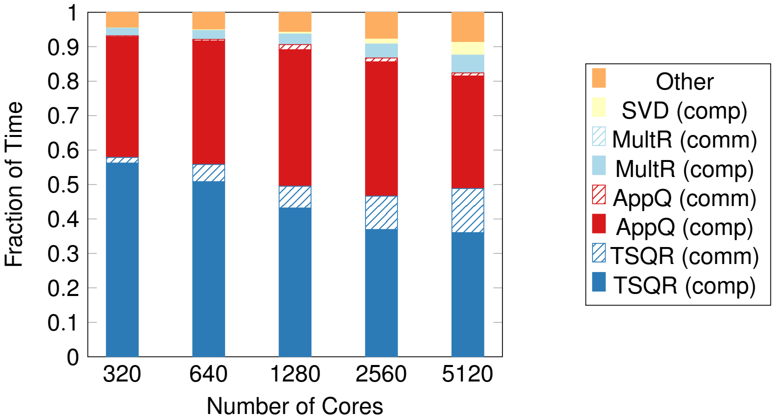
- costs are linear not exponential in N (property of TT)
- communication cost is independent of I (good)
- communication cost increases slightly with P (bad)

Strong Scaling Performance



- $I_n = 512{,}000$, $R_n = 60 \rightarrow 30$, $N = 50$
- strong scaling is (slightly) superlinear
- implicit optimization yields up to 60% improvement

Performance Breakdown



- 70-80% of time spent in local TSQR computations
- communication cost doesn't scale with processors
  - but still not a bottleneck at 128 nodes (5120 cores)

Summary

- Parallel CP bottlenecked by MTTKRP
  - our algorithm's communication cost matches lower bound
  - we avoid redundant computation and communication across modes
- Parallel Tucker bottlenecked by SVD and TTM
  - need communication-efficient distributions and algorithms
  - we can tune the processor grid for efficiency
- Parallel TT-Rounding bottlenecked by Tall-Skinny QR
  - use TSQR algorithm (tree-based reduction technique)
  - need to compute and apply implicit Q matrices
  - communication costs independent of tensor dimensions

For more details:

PLANC: Parallel Low Rank Approximation with Non-negativity Constraints
Srinivas Eswar, Koby Hayashi, Grey Ballard, Ramakrishnan Kannan, Michael Matheson, and Haesun Park
`https://arxiv.org/abs/1909.01149`
[EHB$^+$19]

TuckerMPI: Efficient Parallel Software for Tucker Decompositions of Dense Tensors
Grey Ballard, Alicia Klinvex, and Tamara G. Kolda
arXiv 2019
`https://arxiv.org/abs/1901.06043`
[BKK19]

Communication-Efficient Parallel Algorithms for Tensor Train Orthogonalization and Rounding
Hussam Al Daas, Grey Ballard, Peter Benner
Coming soon...

Mouse Brain Data

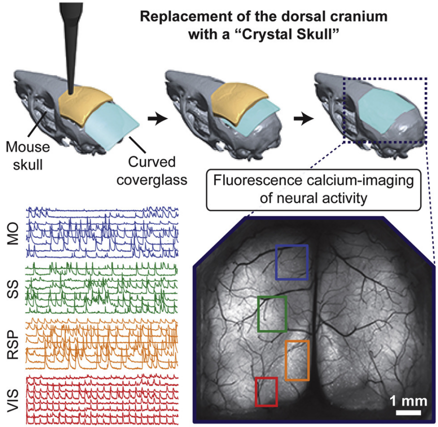
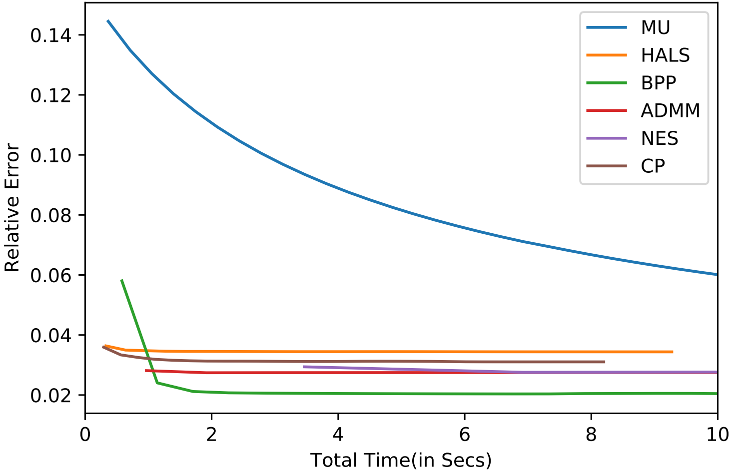


Image from [KZL$^+$16]

- tensor is pixels $\times$ time $\times$ trial: $1.4M \times 69 \times 25$
- about 20 GB when stored in double precision

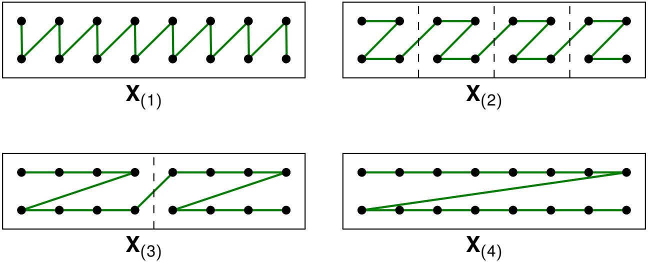
Convergence Results for Mouse Brain Data



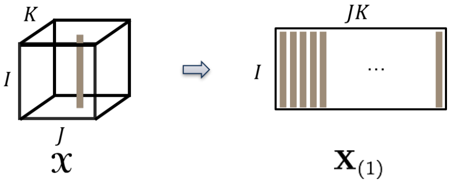
Local tensor data layout in memory

Local matricizations (with no data movement)
lead to matrices in funny layouts

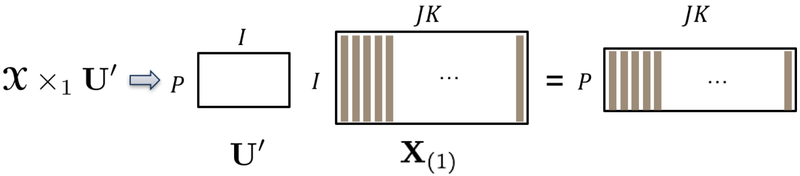
Example: $2 \times 2 \times 2 \times 2$ tensor's matricization layouts



Tensor Times Matrix

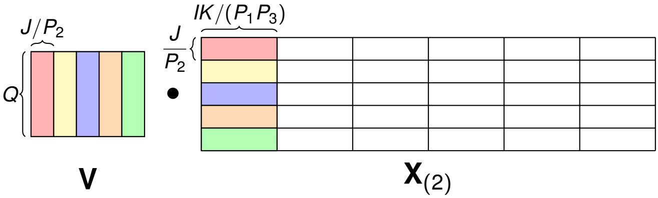


Tensor-times-matrix (TTM) is matrix multiplication with matricized tensor



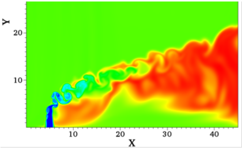
Parallel Tensor Times Matrix

Example: $P_1 \times P_2 \times P_3 = 3 \times 5 \times 2$

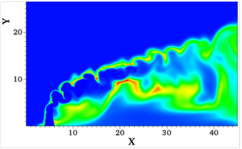


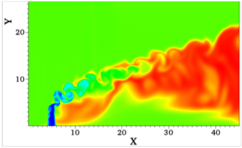
- Matrix $\mathbf{V}$ distributed conformally to 2nd mode of tensor $\mathcal{X}$
- Matrix $\mathbf{V}$ distributed redundantly on processor columns
- Local computation is matrix multiplication
- Communication pattern is reduce-scatter (MPI collective)

More eyeball norm comparisons (JICF)



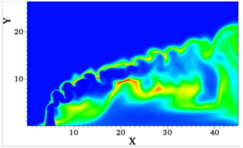
Original

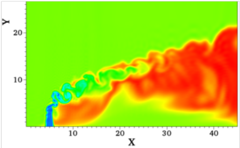




$\epsilon = 10^{-4}$

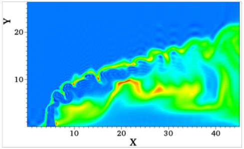
(110X)



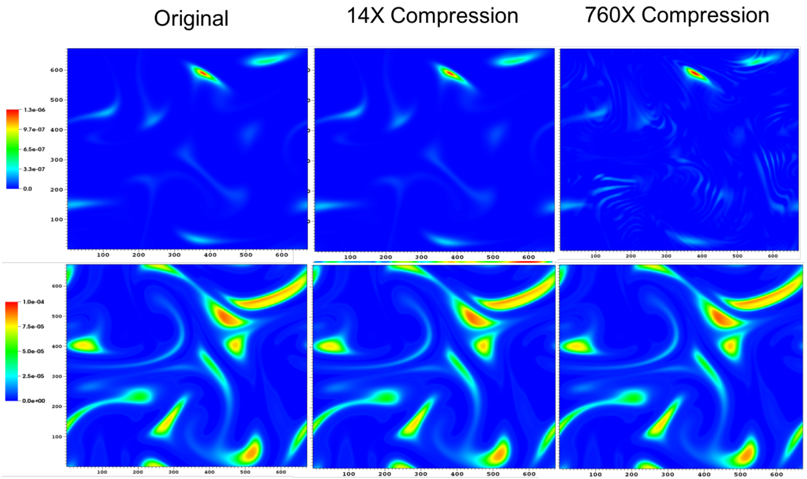


$\epsilon = 10^{-2}$

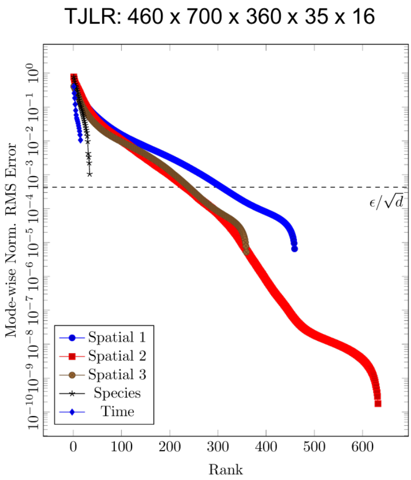
(40000X)

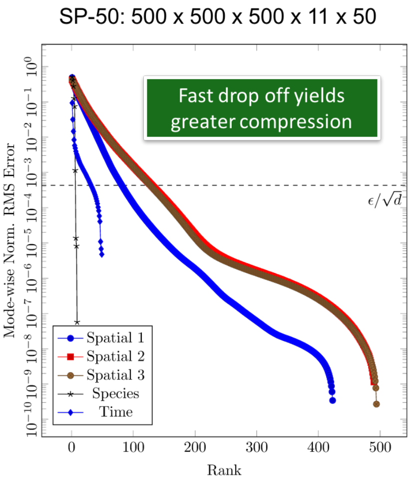


More eyeball norm comparisons (HCCI)



Compressibility depends on data (sing. value decay)



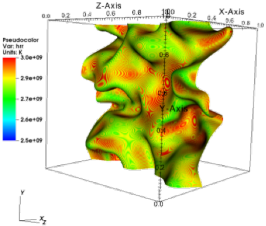


Fast drop off yields greater compression

Reconstruction requires as much space as the original data!

$$\hat{\mathfrak{X}} = \mathfrak{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)} \times_4 \mathbf{U}^{(4)} \times_5 \mathbf{U}^{(5)}$$

$N_1 \times N_2 \times N_3 \times N_4 \times N_5$



But we can just reconstruct the portion that we need at the moment:

$$\bar{\mathfrak{X}} = \mathfrak{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{C}^{(3)}\mathbf{U}^{(3)} \times_4 \mathbf{C}^{(4)}\mathbf{U}^{(4)} \times_5 \mathbf{C}^{(5)}\mathbf{U}^{(5)}$$

$N_1 \times N_2 \times \frac{N_3}{2} \times 1 \times 1$

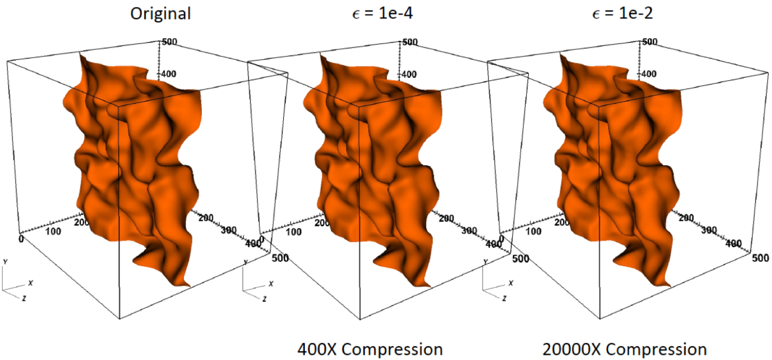
$$\mathbf{C}^{(3)} = \begin{bmatrix} 1/2 & 0 & \cdots & 0 \\ 1/2 & 0 & \cdots & 0 \\ 0 & 1/2 & \cdots & 0 \\ 0 & 1/2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \qquad \mathbf{C}^{(4)} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{bmatrix} \qquad \mathbf{C}^{(5)} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

Downsample

Pick single variable

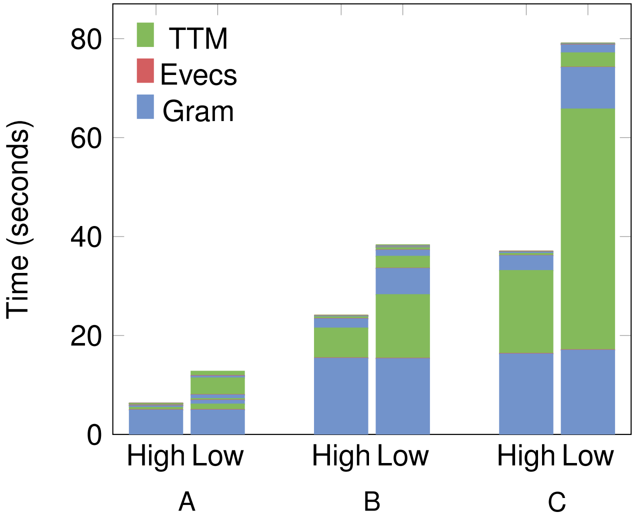
Pick single time step

Flame Surface Reconstruction

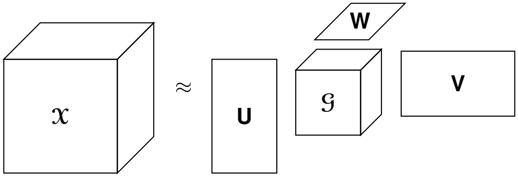


Flame surface at single time step.
Using temperature variable (iso-value is 2/3 of max).

Processor Grid Comparison



Tucker compression

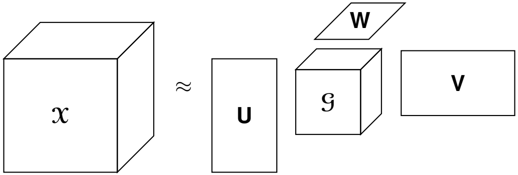


$$\underbrace{\mathcal{X}}_{I \times J \times K} \approx \underbrace{\mathcal{G}}_{P \times Q \times R} \times_1 \underbrace{\mathbf{U}}_{I \times P} \times_2 \underbrace{\mathbf{V}}_{J \times Q} \times_3 \underbrace{\mathbf{W}}_{K \times R}$$

Compression ratio

$$C = \frac{IJK}{PQR + IP + JQ + KR} \approx \frac{IJK}{PQR}$$

Tucker approximation error



$$x_{ijk} \approx \tilde{x}_{ijk} = \sum_{p,q,r} g_{pqr} u_{ip} v_{jq} w_{kr}$$

Approximation error

$$\frac{\|\mathcal{X} - \tilde{\mathcal{X}}\|}{\|\mathcal{X}\|} = \frac{\left(\sum_{i,j,k}\left(x_{ijk} - \sum_{p,q,r} g_{pqr} u_{ip} v_{jq} w_{kr}\right)^2\right)^{1/2}}{\left(\sum_{i,j,k} x_{ijk}^2\right)^{1/2}}$$

Strong scaling benchmark

Problem Setup

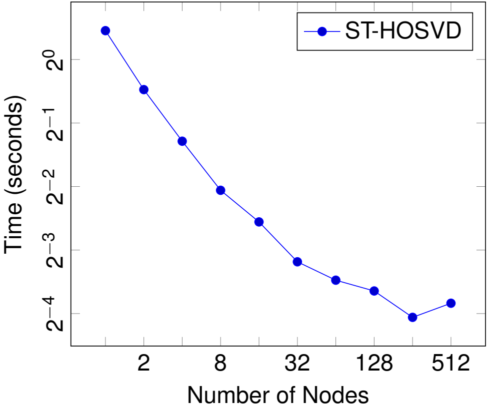
- $200 \times 200 \times 200 \times 200$ data tensor (12 GB)
- $20 \times 20 \times 20 \times 20$ core tensor
- $24 \cdot 2^k$ processors (cores)

Result

- small problem, but running time decreases with up to 6144 cores

Compute Platform

- Edison (NERSC), Cray XC30
- 24-core nodes



References I

Woody Austin, Grey Ballard, and Tamara G. Kolda.
Parallel tensor compression for large-scale scientific data.
In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium*, pages 912–922, May 2016.

Grey Ballard, Alicia Klinvex, and Tamara G. Kolda.
TuckerMPI: Efficient parallel software for Tucker decompositions of dense tensors.
Technical Report 1901.06043, arXiv, 2019.

Grey Ballard, Nicholas Knight, and Kathryn Rouse.
Communication lower bounds for matricized tensor times Khatri-Rao product.
In *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium*, pages 557–567, May 2018.

- Andrzej Cichocki, Rafal Zdunek, Anh-Huy Phan, and Shun-ichi Amari.
  Nonnegative Matrix and Tensor Factorizations: Applications to exploratory multi-way data analysis and blind source separation.
  John Wiley & Sons, 2009.

- J. Demmel, L. Grigori, M. Hoemmen, and J. Langou.
  Communication-optimal parallel and sequential QR and LU factorizations.
  SIAM Journal on Scientific Computing, 34(1):A206–A239, 2012.

- Srinivas Eswar, Koby Hayashi, Grey Ballard, Ramakrishnan Kannan, Michael A. Matheson, and Haesun Park.
  PLANC: Parallel low rank approximation with non-negativity constraints.
  Technical Report 1909.01149, arXiv, 2019.

Jingu Kim and Haesun Park.
Fast nonnegative matrix factorization: An active-set-like method and comparisons.
SIAM Journal on Scientific Computing, 33(6):3261–3281, 2011.

Tony Hyun Kim, Yanping Zhang, Jérôme Lecoq, Juergen C. Jung, Jane Li, Hongkui Zeng, Cristopher M. Niell, and Mark J. Schnitzer.
Long-term optical access to an estimated one million neurons in the live mouse cortex.
Cell Reports, 17(12):3385 – 3394, 2016.

A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos.
Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementation.
IEEE Transactions on Signal Processing, Nov 2017.

Daniel D Lee and H Sebastian Seung.
Learning the parts of objects by non-negative matrix factorization.
Nature, 401(6755):788, 1999.

A. P. Liavas and N. D. Sidiropoulos.
Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers.
IEEE Transactions on Signal Processing, 63(20):5450–5463, Oct 2015.

Nick Vannieuwenhoven, Raf Vandebril, and Karl Meerbergen.
A new truncation strategy for the higher-order singular value decomposition.
SIAM Journal on Scientific Computing, 34(2):A1027–A1052, 2012.