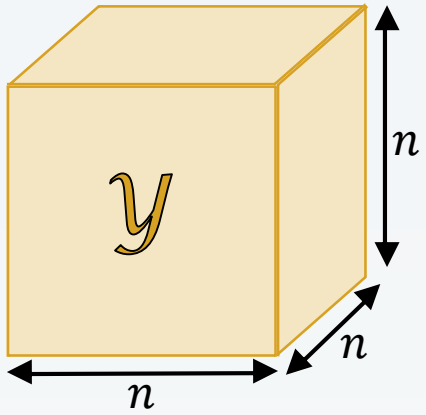
Efficient CP-ALS and Reconstruction From CP

Jed A. Duersch & Tamara G. Kolda
Sandia National Laboratories
Livermore, CA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This research was funded by LDRD project 199986.

Tensor Notation



y is a tensor.

d is the order.

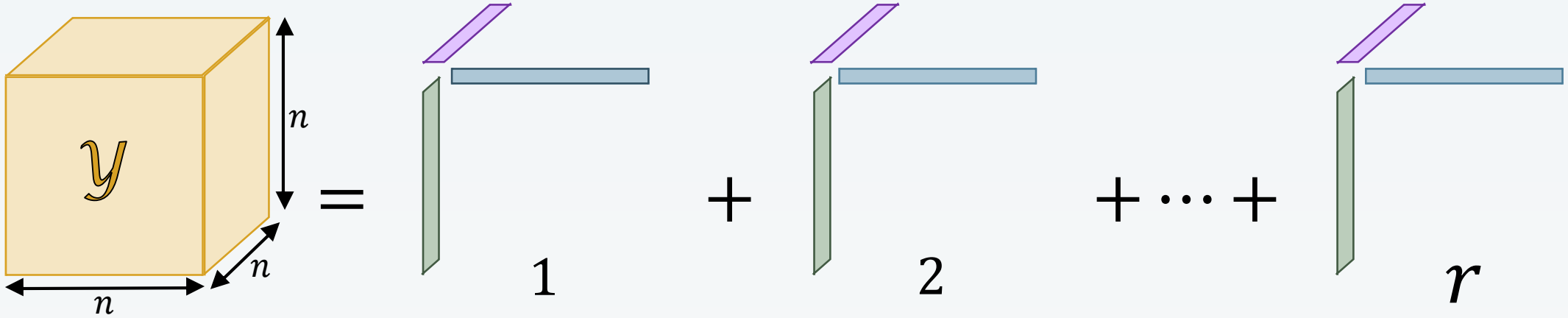
Number of indices to locate an element. $y_{(i_1,i_2,\ldots,i_d)}$

n is the size of each mode.

Generally mode k has size n_k, but all set to n for simplicity.

(It's a hypercube!)

Canonical Polyadic Decomposition (CPD)



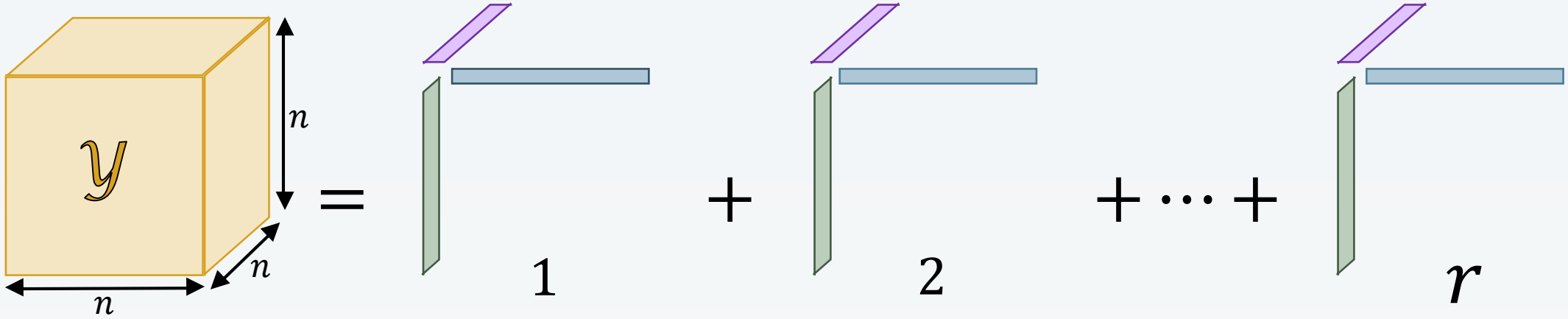
CPD (aka CANDECOMP/PARAFAC) expresses y as a sum of rank-1 tensor products.

d is the order.

n is the size of each mode.

r is the number of components.

Example Sizes



As we discuss computations and memory requirements, keep this example in mind.

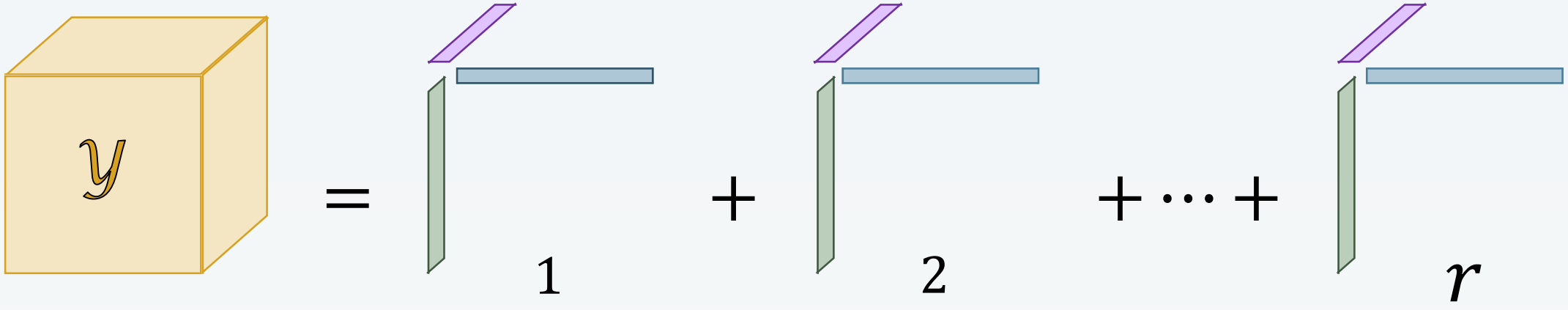
Order: $d = 4$

Mode-size: $\mathrm{n} = 100$

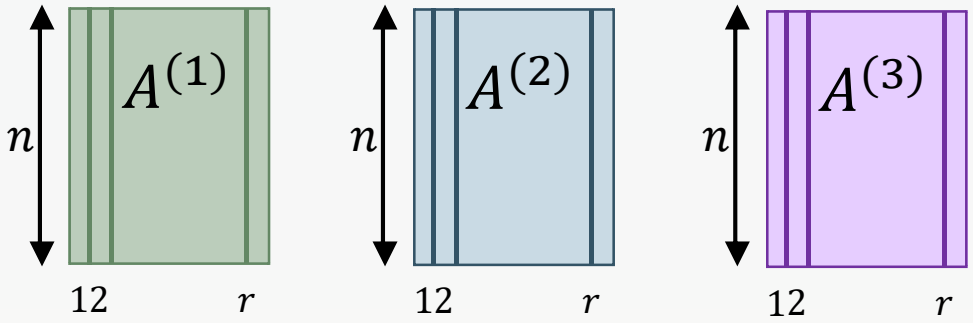
Components: $\mathrm{r} = 10$

Tensor Size: $\mathrm{n}^{\mathrm{d}} = 10^8$

Factor Matrices



Ex: 10^8



Ex: 10^3 each

We can arrange vector components as columns in *factor matrices*.

Ex: $d = 4$ $\mathrm{n} = 100$ $\mathrm{r} = 10$

Problem: Reconstruct from CPD

Sometimes you need to reconstruct the full tensor.
This is `full()` in Tensor Toolbox for MATLAB.

Orange = $O(n^d)$
Ex: 10^8

$$\mathcal{Y}_{(i_1,i_2,\ldots,i_d)} = \sum_{j=1}^{r} \lambda_j \left[\prod_{k=1}^{d} A^{(k)}_{i_k\, j} \right]$$

Blue = $O(rn)$
Ex: 10^3

Sum over each component

Optionally include scaling factors for each component

Product over factor matrices

Compute: $r\, n^d = 10^9$
fused multiply-accumulates (FMAs)

How can we compute this efficiently?

Ex: $d = 4$ $\mathrm{n} = 100$ $\mathrm{r} = 10$

Unfolding and Index Flattening

We can unfold $\mathcal{Y}$ into a vector without memory movement.

Natural descending element order:

$\mathcal{Y}_{(i_1,i_2,\ldots,i_d)}$ is located at array offset

$y[i_1 + (i_2 - 1)n + (i_3 - 1)n^2 + \ \ldots + (i_d - 1)n^{d-1}]$.

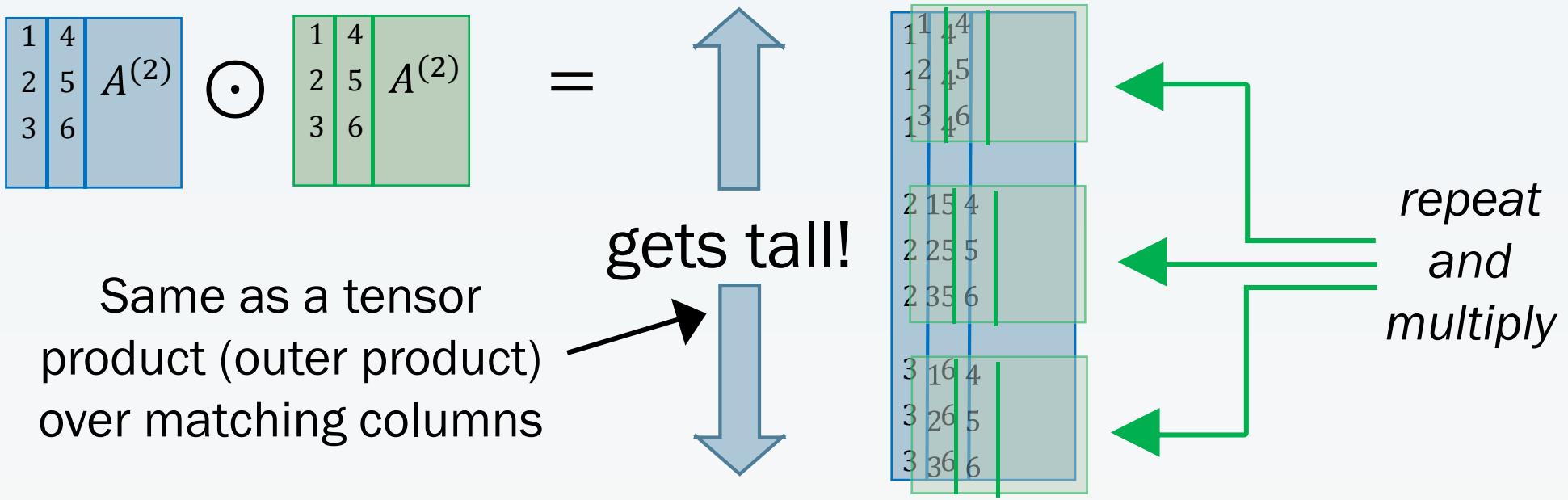
Vectorization:

Take $y = \mathrm{vec}(\mathcal{Y})$ by flattening the multiindex

$\hat{\imath} = i_1 + (i_2 - 1)n + (i_3 - 1)n^2 + \ \ldots + (i_d - 1)n^{d-1}$.

$$y_{\hat{\imath}} = \mathcal{Y}_{(i_1,i_2,\ldots,i_d)}$$

Khatri-Rao Product (KRP)



$$K = A^{(2)} \odot A^{(1)}$$

means

Columnwise Kronecker Product

$$K_{\hat{\imath}\, j} = A^{(2)}_{i_2\, j}\, A^{(1)}_{i_1\, j} \qquad \text{where } \hat{\imath} = i_1 + (i_2 - 1)n.$$

Vectorized Full is Expensive in Memory!

Using $y = \text{vec}(\mathcal{Y})$

Ex: 10^8

Red $= O(rn^d)$

Ex: 10^9

and $K = A^{(d)} \odot \cdots \odot A^{(2)} \odot A^{(1)}$

Ex: 10^9 Ex: 10^3 each

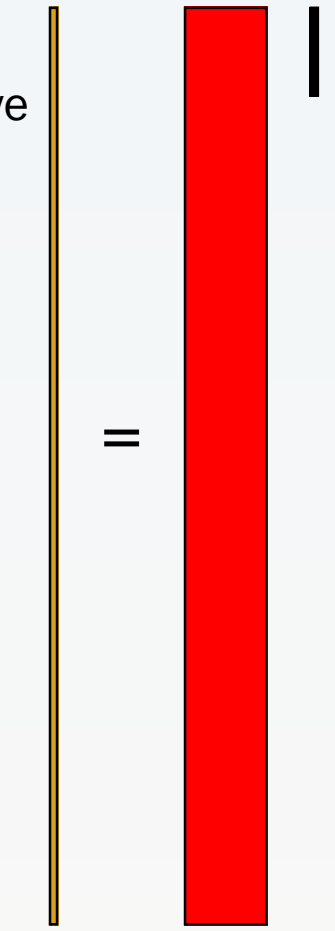
we can rewrite this $\mathcal{Y}_{(i_1,i_2,\ldots,i_d)} = \sum_{j=1}^{r} \lambda_j \left[\prod_{k=1}^{d} A^{(k)}_{i_k\, j} \right]$

as matrix-vector multiply. $y = K\,\lambda$

10^9 FMAs

Matrix-vector multiply (gemv)

1 FMA/move



Ex: $d = 4$ $\mathrm{n} = 100$ $\mathrm{r} = 10$

Unfolding Revisited

Rather than unrolling *all* modes into the row index,

$$\hat{\imath} = i_1 + (i_2 - 1)n + (i_3 - 1)n^2 + \ \ldots + (i_d - 1)n^{d-1} \qquad y_{\hat{\imath}} = \mathcal{Y}_{(i_1, i_2, \ldots, i_d)}$$

we can split modes between rows and columns.

Modes 1 to s go into rows and the rest in columns.

1:s $\quad \hat{\imath}_1 = i_1 + (i_2 - 1)n + \ \ldots (i_s - 1)n^{s-1}$

s+1:d $\quad \hat{\imath}_2 = i_{s+1} + (i_{s+2} - 1)n + \ \ldots (i_d - 1)n^{d-s-1}$

Still no memory movement

$$Y = \mathrm{mat}_{(1:s)}(\mathcal{Y}) \qquad Y_{\hat{\imath}_1 \hat{\imath}_2} = \mathcal{Y}_{(i_1, i_2, \ldots, i_d)}$$

Key Idea: Matricized Reconstruction

Using $Y = \text{mat}_{(1:s)}(\mathcal{Y})$ with

Ex: 10^8

Green = $O(rn^{d/2})$

Ex: 10^5

Left modes $L = A^{(s)} \odot \cdots \odot A^{(2)} \odot A^{(1)}$

Right modes $R = A^{(d)} \odot \cdots \odot A^{(s+2)} \odot A^{(s+1)}$

Ex: 10^5 Ex: 10^3 each

changes $\mathcal{Y}_{(i_1,i_2,\ldots,i_d)} = \sum_{j=1}^{r} \lambda_j \left[\prod_{k=1}^{d} A^{(k)}_{i_k\, j} \right]$ Ex: 10^9

into matrix multiply $Y = L\, \text{diag}(\lambda)\, R^T$

10^9 FMAs

Ex: $d = 4$ $n = 100$ $r = 10$

Reduced Memory, Higher Computational Intensity

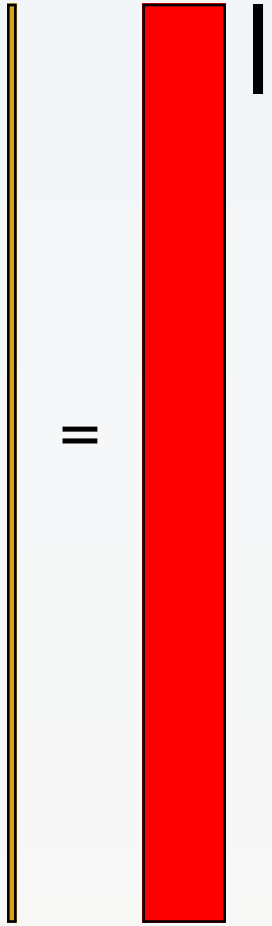
$y = K\ \lambda$

Ex: 10^8 Ex: 10^9

Construct:
$r\ n^d = 10^9$ floats

Compute:
$r\ n^d = 10^9$ FMAs

Output:
$n^d = 10^8$ floats

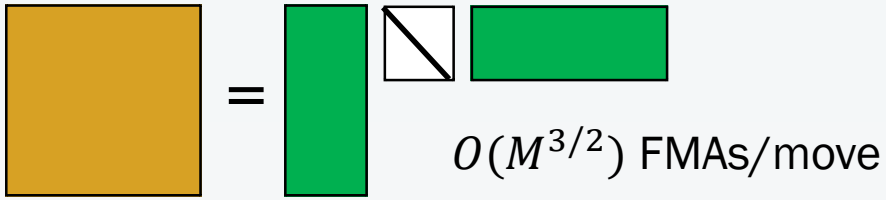


Matrix-vector multiply (gemv)

1 FMA/move

$Y = L\ \mathrm{diag}(\lambda)\ R^T$

Ex: 10^8 Ex: 10^5 Ex: 10^5



$O(M^{3/2})$ FMAs/move

Matrix-matrix multiply (gemm)

Construct:
$2\ r\ n^{d/2} = 2 \times 10^5$ floats

Compute: same

Output: same

Ex: $d = 4$ $\mathrm{n} = 100$ $\mathrm{r} = 10$

Choosing the optimal splitting

Fixed cost: Matrix-matrix multiply.

Strong implementations of gemm() typically hit 90%+ of peak.

Memory movement is hidden.

Controllable cost: Khatri-Rao product formation.

Requires at least 1 pass over output memory:

$$s^* = \arg\min_s \left(r \left[\prod_{k=1}^{s} n_k \right] + r \left[\prod_{k=s+1}^{d} n_k \right] \right)$$

L R

Matricized full uses far less memory!

Sandia National Laboratories

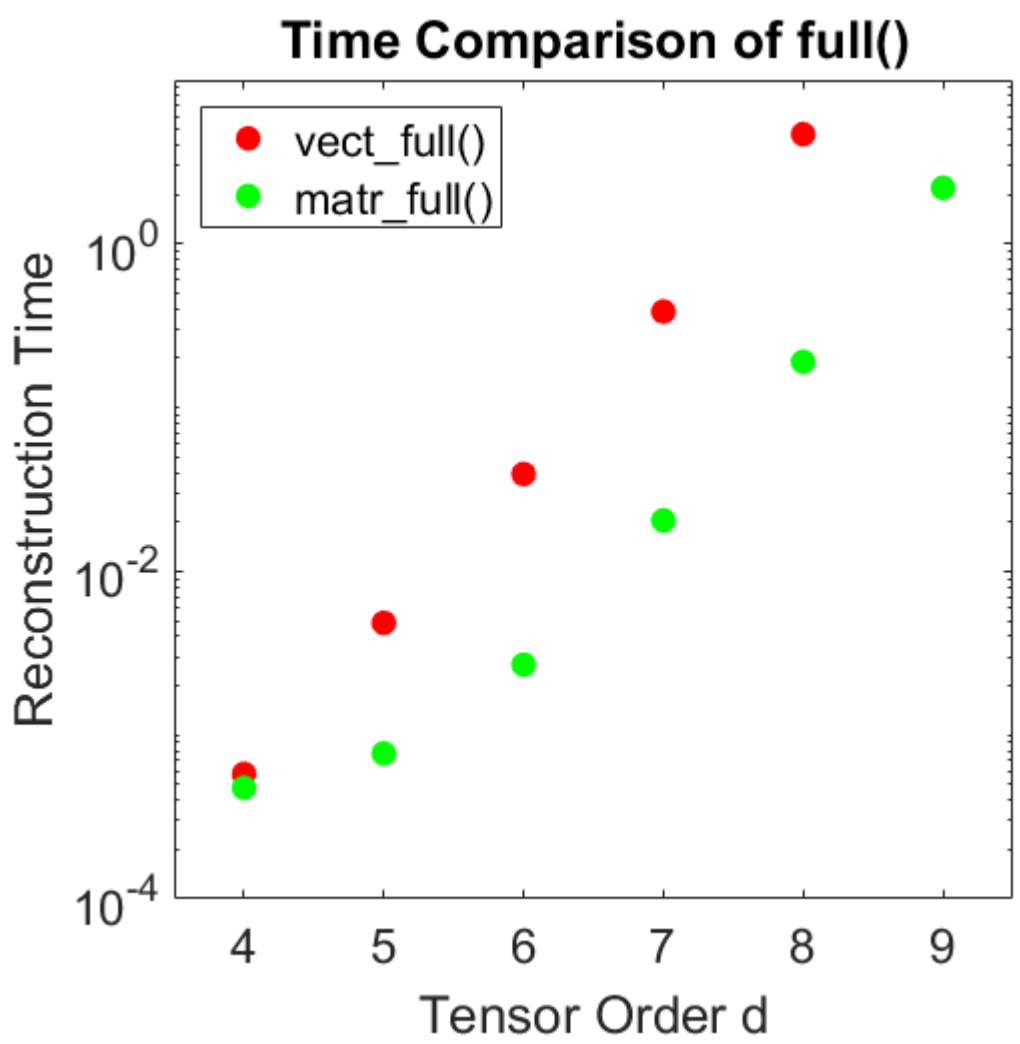
Smaller n lets us test higher orders.

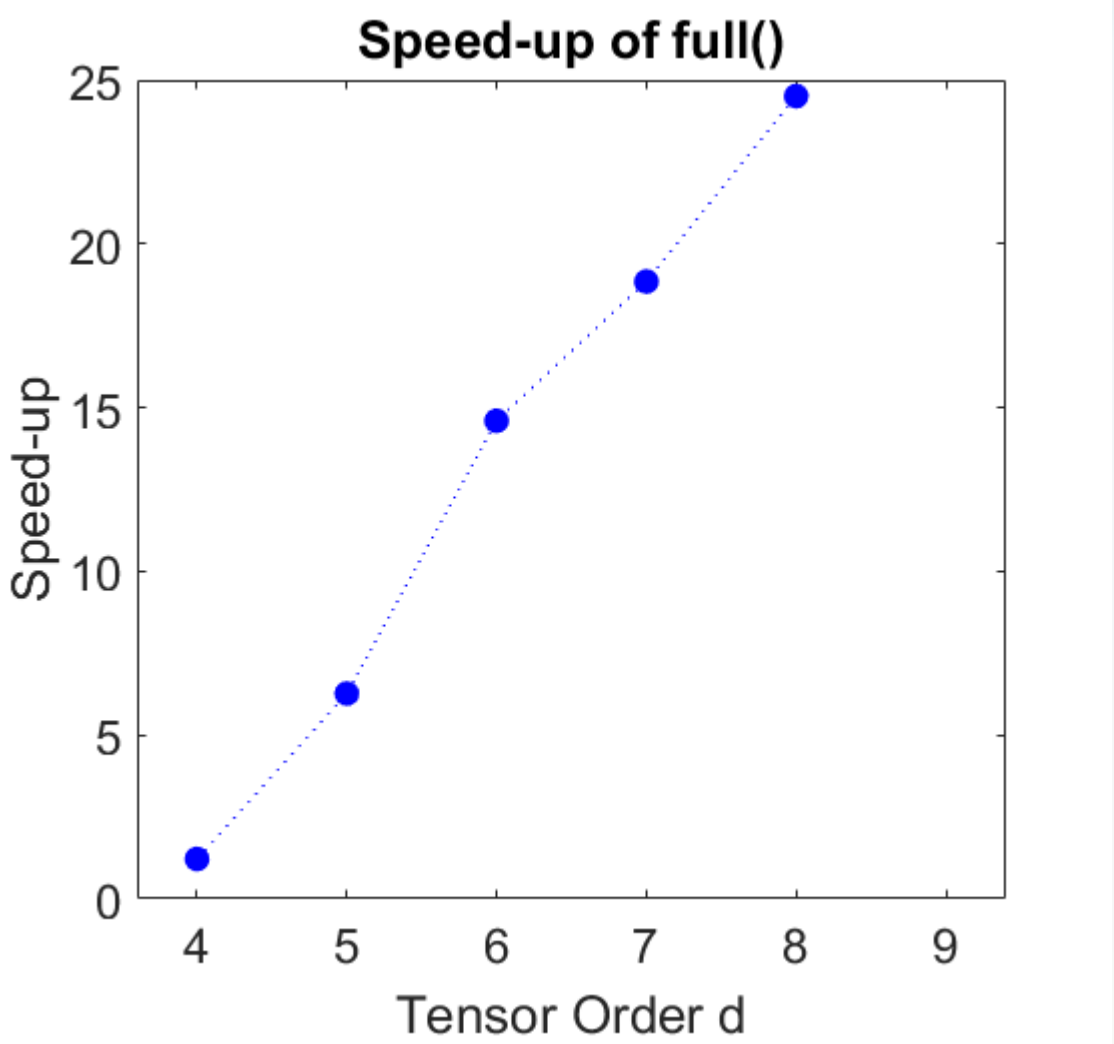
Tests use: $n = 10$ $r = 10$

Order d	Tensor Elts	Vect Mem	Matr Mem	Vect Time	Matr Time	Speed Up
4	10^4	781 kB	15.6 kB	$5.8 \times 10^{-4}\ s$	$4.8 \times 10^{-4}\ s$	1.2
5	10^5	7.63 MB	85.9 kB	$4.8 \times 10^{-3}\ s$	$7.7 \times 10^{-3}\ s$	6.3
6	10^6	76.3 MB	156 kB	$3.9 \times 10^{-2}\ s$	$2.7 \times 10^{-3}\ s$	15
7	10^7	763 MB	859 kB	$3.9 \times 10^{-1}\ s$	$2.0 \times 10^{-2}\ s$	19
8	10^8	7.45 GB	1.53 MB	$4.7 \times 10^{0}\ s$	$1.9 \times 10^{-1}\ s$	25
9	10^9	74.5 GB	8.39 MB	FAILED	$2.2\ s$	N/A

Test were run on Dell laptop, Intel Core i7 vPro, with MATLAB R2017a 64-bit.

Larger Tensors – Better Speed-up





Tests use $n = 10$ $r = 10$

Run on Dell laptop, Intel Core i7 vPro, with MATLAB R2017a 64-bit.

CP-Alternating Least Squares (CP-ALS)

Sandia National Laboratories

Now we switch focus to extracting $\mathcal{Y} = [\![\lambda; A^{(1)}, \ldots, A^{(d)}]\!]$ from data tensor $\mathcal{X}$.

- repeat
  - for k = 1 : d
    - Fix factor matrices $A^{(1)}, \ldots, A^{(k-1)}, A^{(k+1)}, \ldots, A^{(d)}$.
    - Solve $A^{(k)}$.
  - end for
  - Compute fit.
- until fit change is below threshold

Gauss-Seidel iterative technique cycles through subproblems.

Computation Bottleneck

Matricized tensor times Khatri-Rao product (MTTKRP)

Updating $A^{(k)}$ *requires this computation:*

$$Z^{(k)} = X_{(k)}\left[A^{(d)} \odot \cdots \odot A^{(k+1)} \odot A^{(k-1)} \odot \cdots \odot A^{(1)}\right]\mathrm{diag}(\lambda)$$

Mode-k unfolding

We just saw that even constructing this may be quite expensive!

Construct:
$r\,n^{d-1} = 10^7$ floats

Compute:
$r\,n^{d} = 10^9$ FMAs

Optimal splitting allowed us to avoid forming a large KRP.
Can we use the same technique here?

Splitting the Khatri-Rao Product

If $k \leq s$, we can rewrite

$$Z^{(k)} = X^{(k)}\left[A^{(d)} \odot \cdots \odot A^{(k+1)} \odot A^{(k-1)} \odot \cdots \odot A^{(1)}\right]\mathrm{diag}(\lambda)$$

Ex: 10^3 Ex: 10^8 Ex: 10^7

with an intermediate computation

$$X = \mathrm{mat}_{(1:s)}(\mathcal{X}) \qquad W^{(\mathrm{R})} = X\left[A^{(d)} \odot \cdots \odot A^{(s+1)}\right]$$

Modes 1:s in rows

Ex: 10^5 Ex: 10^8 Ex: 10^5

Finishing from this form is fast. Details omitted.
~1 pass $= rn^{d/2} = 10^5$ moves and FMAs

Both matrix multiplies do $rn^d = 10^9$ FMAs.
Key difference is cost to construct inputs.

Ex: $d = 4$ $\mathrm{n} = 100$ $\mathrm{r} = 10$

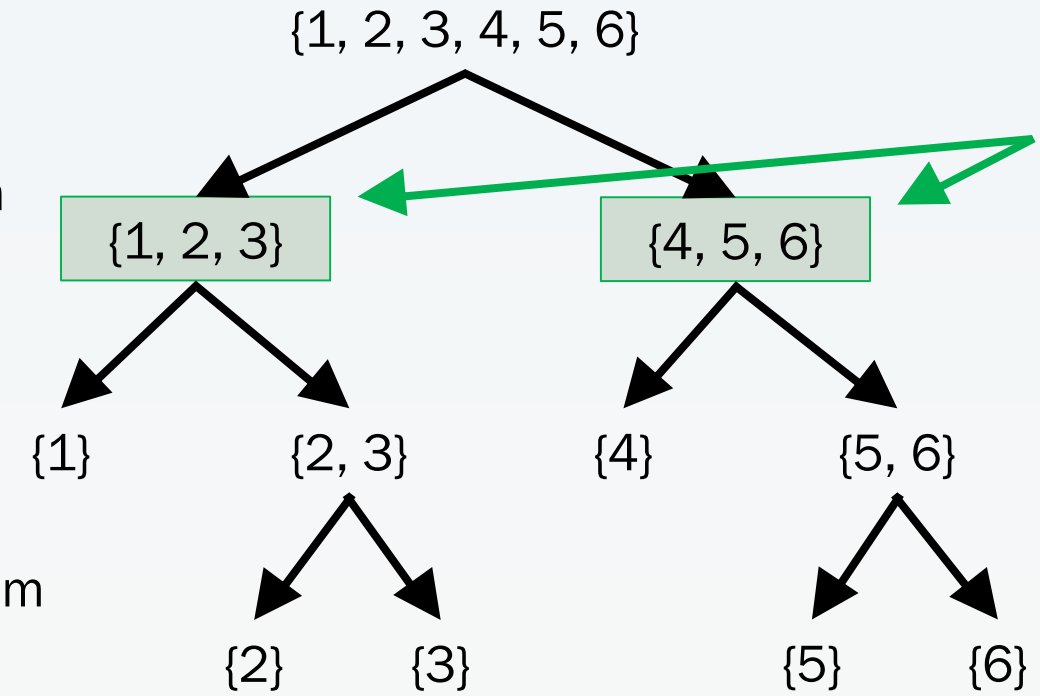
Connection to Dimension Trees

Each node lists remaining (unreduced) modes.

Each leaf is an MTTKRP on a different mode.

Dimension trees save all intermediate results.

Bulk of speedup comes from keeping first reductions.



Minimal total construction from optimal splitting.
Memory: $2rn^{d/2}$
Compute: $2rn^{d}$

Follow-up ops are lower complexity (multi-Tensor-Times-Vector).
Compute: $\sim drn^{d/2}$

A. Phan, P. Tichavsky, A. Cichocki
"Fast Alternating LS Algorithms for High Order CANDECOMP/PARAFAC Tensor Factorizations," 2013

K.. Rouse, G. Ballard, N. Knight
"Communication Lower Bounds for Matricized Tensor Times Khatri-Rao Product," 2018

Optimal Split Reuse in Update Sequence

Update:	Matrices used in efficient MTTKRP formulation							
$A^{(1)} \mapsto \hat{A}^{(1)}$		$A^{(2)}$	$\cdots$	$A^{(s)}$	$W^{(R)} = X \left[\bigodot_{k'=s+1}^{d} A^{(k')} \right]$			
$A^{(2)} \mapsto \hat{A}^{(2)}$	$\hat{A}^{(1)}$		$\ddots$	$A^{(s)}$				
$\vdots$	$\hat{A}^{(1)}$	$\hat{A}^{(2)}$		$A^{(s)}$				
$A^{(s)} \mapsto \hat{A}^{(s)}$	$\hat{A}^{(1)}$	$\hat{A}^{(2)}$	$\ddots$					
$A^{(s+1)} \mapsto \hat{A}^{(s+1)}$	$W^{(L)} = \left[\bigodot_{k'=1}^{s} \hat{A}^{(k')} \right]^{T} X$					$A^{(s+2)}$	$\cdots$	$A^{(d)}$
$A^{(s+2)} \mapsto \hat{A}^{(s+2)}$					$\hat{A}^{(s+1)}$		$\ddots$	$A^{(d)}$
$\vdots$					$\hat{A}^{(s+1)}$	$\hat{A}^{(s+2)}$		$A^{(d)}$
$A^{(d)} \mapsto \hat{A}^{(d)}$					$\hat{A}^{(s+1)}$	$\hat{A}^{(s+2)}$	$\ddots$	

Original factor matrices

Updated factor matrices

Efficient convergence check uses $W^{(L)}$ with $\left[\bigodot_{k'=s+1:d} \hat{A}^{(k')}\right]$ (Reuse next iteration!)

MTTKRP-Sequence Cost Comparison

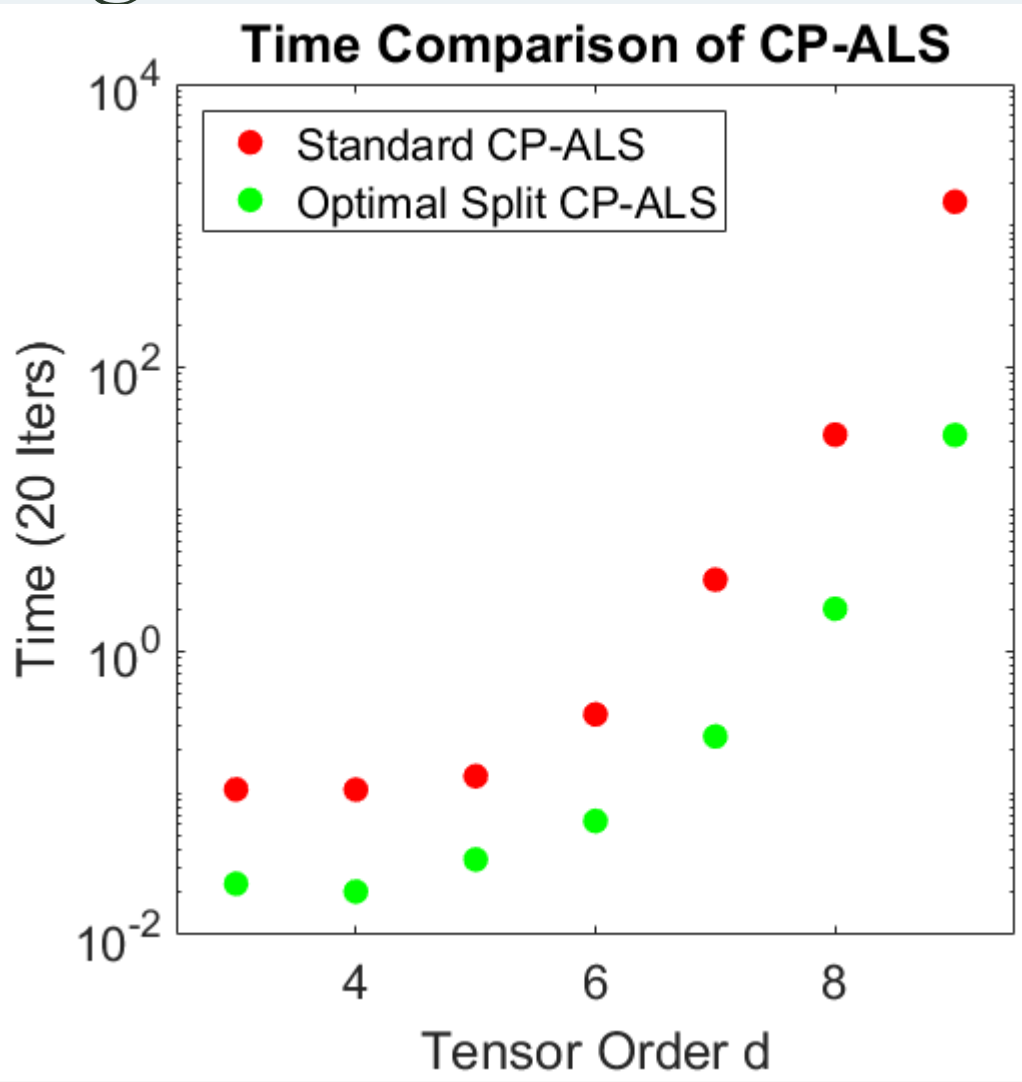
Sandia National Laboratories

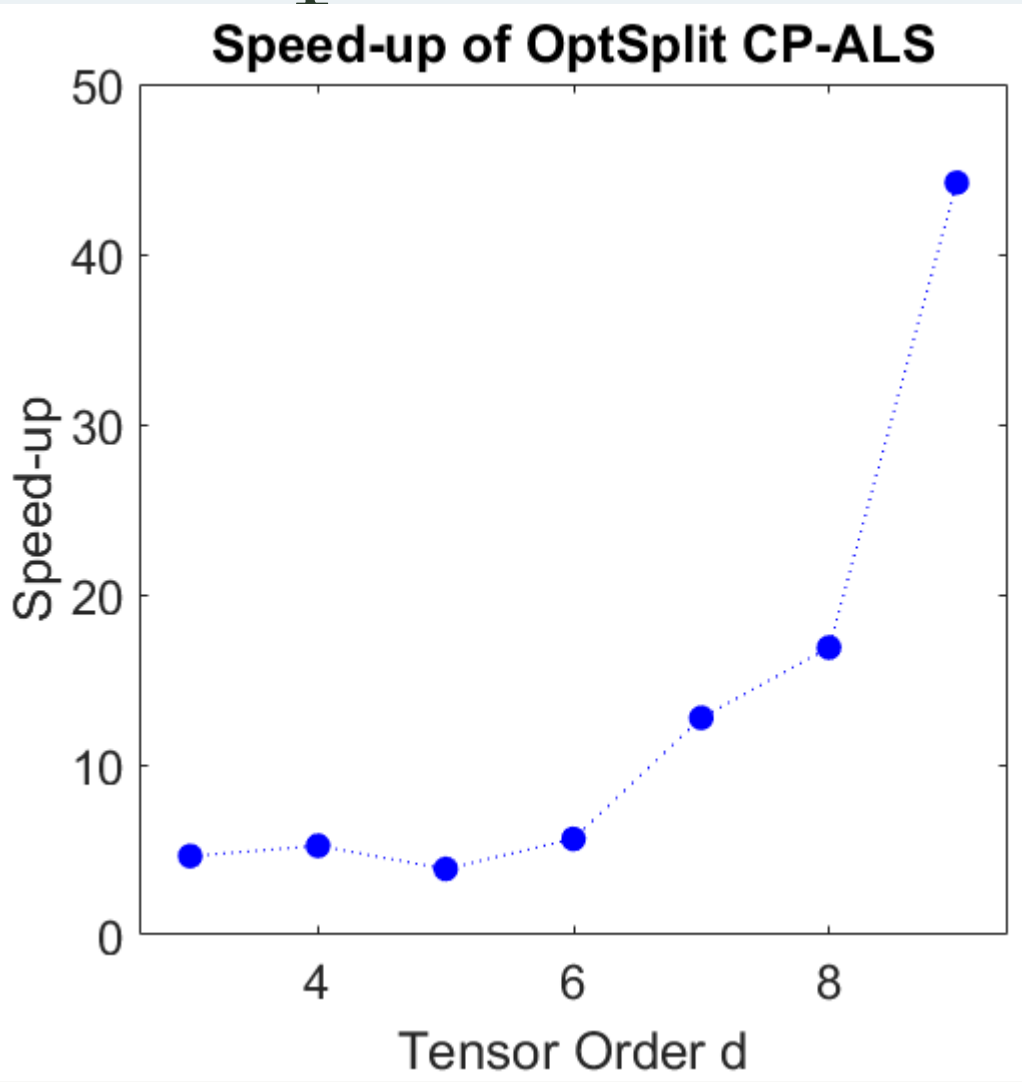
Ex: $d = 4 \;\; \mathrm{n} = 100 \;\; \mathrm{r} = 10$

CP-ALS Algorithm	KRP Construction*	Ex (floats)	Post-KRP Computations*	Ex (FMAs)
Standard	drn^{d-1}	4.00×10^7	drn^d	4.00×10^9
OptSplit	$2rn^{d/2} + drn^{\frac{d}{2}-1}$	2.04×10^5	$2rn^d + drn^{d/2}$	2.00×10^9
DimTree	$2rn^{d/2} + 4rn^{\frac{d}{4}}$	2.04×10^5	$2rn^d + 4rn^{d/2}$	2.00×10^9

**Leading two terms. Assumes perfect splitting.*

Higher Order – Better Speed-up





Tests use $n = 10$ $r = 10$

Run on Dell laptop, Intel Core i7 vPro, with MATLAB R2017a 64-bit.

Splitting KRPs reduces memory and allows computational reuse.

- Optimal splitting minimizes construction cost for Khatri-Rao products.
- Matricized full() uses less memory which is faster.
  - Vectorized: rn^d Matricized: $2rn^{d/2}$
- MTTKRP sequence (all modes) can be done forming two medium KRPs.
  - Standard: drn^{d-1} floats OptSplit: $2rn^{d/2}$ floats
- MTTKRP sequence can be computed using two matrix multiplies.
  - Standard: drn^d FMAs OptSplit: $2rn^d$ FMAs
- Follow-up ops are lower complexity - a simple implementation works well.
- Technique applies to both alternating (CP-ALS) and all-at-once (CP-OPT).

Jed A. Duersch - jaduers@sandia.gov