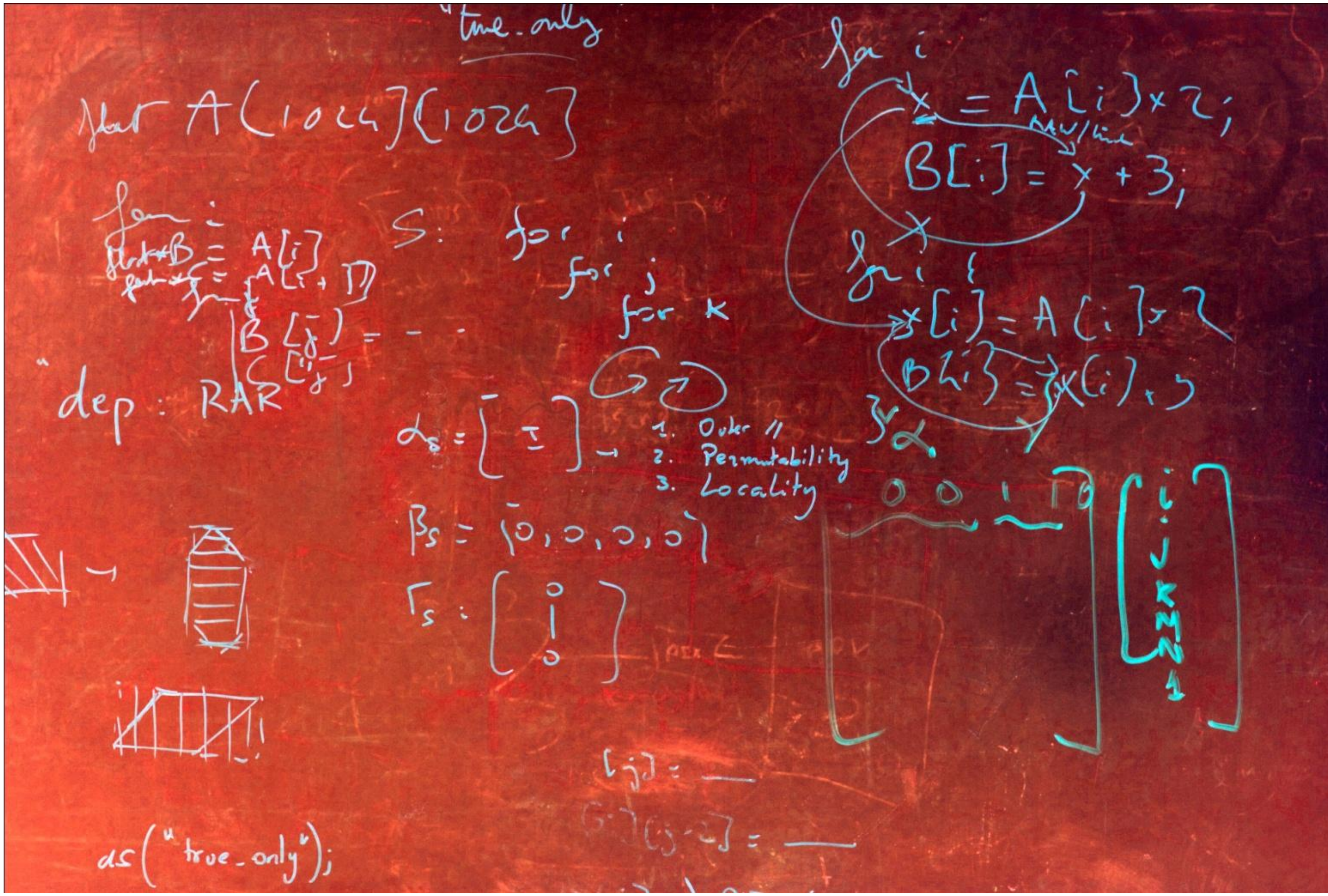
Efficient and Scalable Computations with Sparse Tensors

M. Baskaran, B. Meister, N. Vasilache, R. Lethin



Introduction

ENSIGN (Exascale Non-Stationary Graph Notation)

- Goal
  - **Source-to-source optimization tool for "dynamic" graph analytics**
    - Key highlight: Specify graph analytics in the form of multi-linear (tensor) algebraic formulations
    - Input: High-level specification of graph analytics
    - Output: Optimized code for exascale parallel systems
- Features
  - High-level programming notation for tensor computations
  - **Scalable optimizations for tensor computations**
  - Automatic parallelization and data locality optimizations
    - Inter-operate with Reservoir Labs' auto-parallelizing compiler **R-Stream**

Presentation Roadmap

Motivation

Background

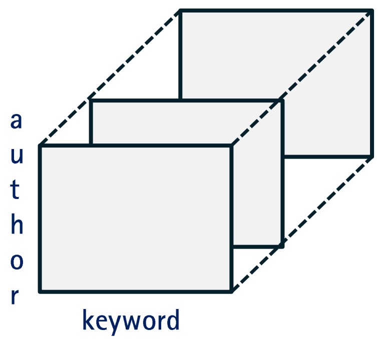
Techniques

Performance Results

Summary & Forward Work

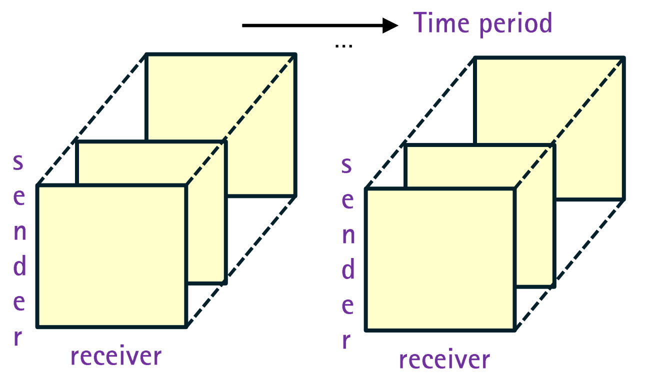
Real-world Data – Representation

Real-world data are multi-dimensional with multiple aspects



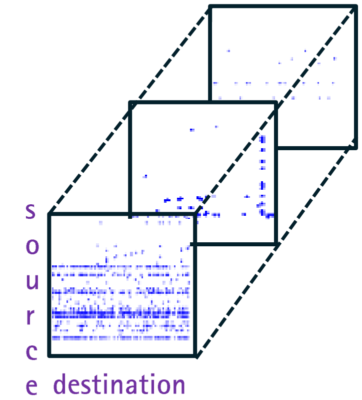
Author x Keyword x Year

Bibiliometric Data



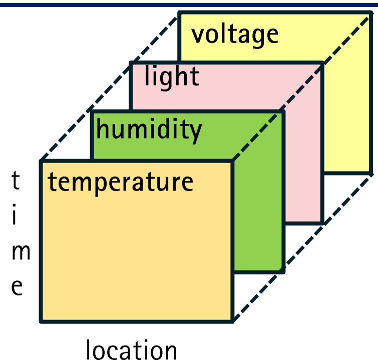
Sender x Receiver x Keyword x Time period

Email Data



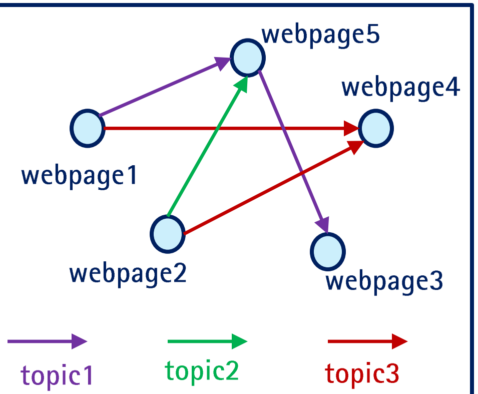
Src x Dest x Time

Network Traffic Data



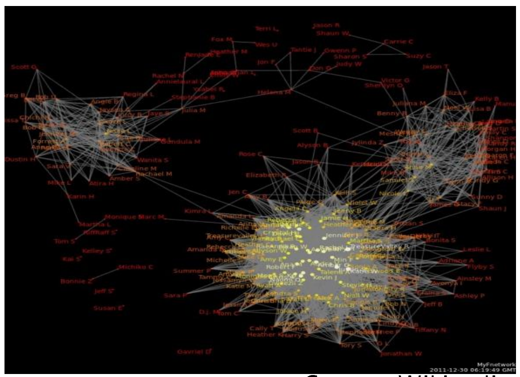
Time x Location x Type

Environmental Sensor Monitoring



Page x Page x Topic

Web graph



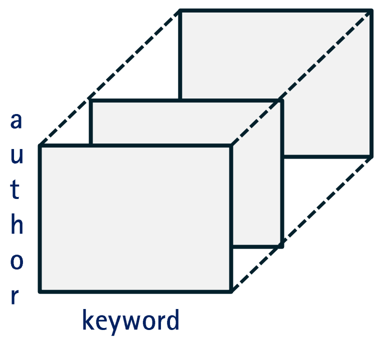
Source: Wikipedia

Person x Person x Relation

Social network graph

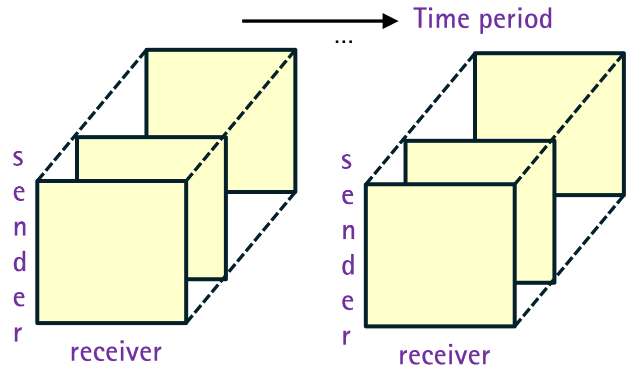
Real-world Data – Representation

Tensors or multi-dimensional arrays are natural fit to represent



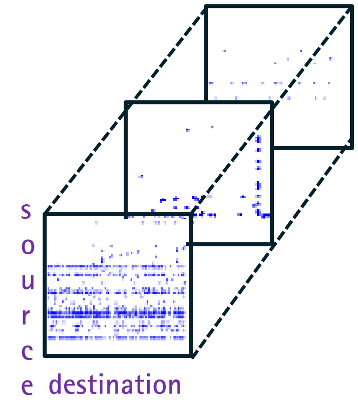
Author x Keyword x Year

Bibiliometric Data



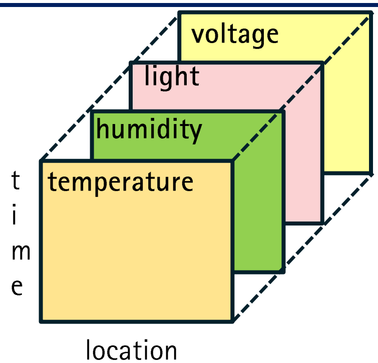
Sender x Receiver x Keyword x Time period

Email Data



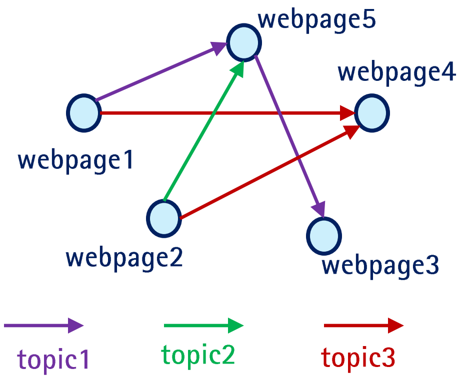
Src x Dest x Time

Network Traffic Data



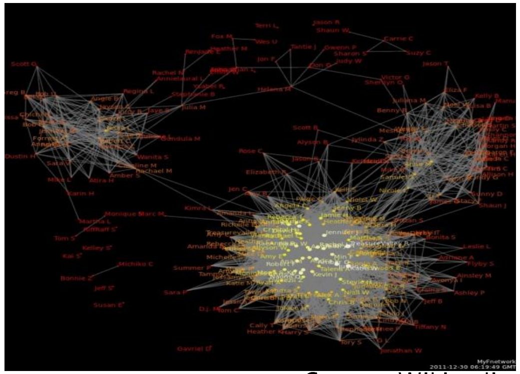
Time x Location x Type

Environmental Sensor Monitoring



Page x Page x Topic

Web graph



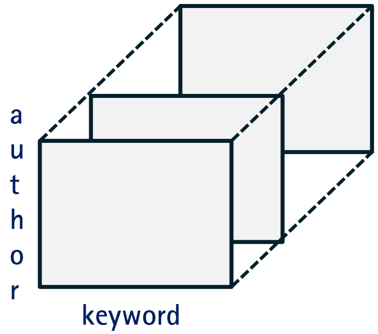
Source: Wikipedia

Person x Person x Relation

Social network graph

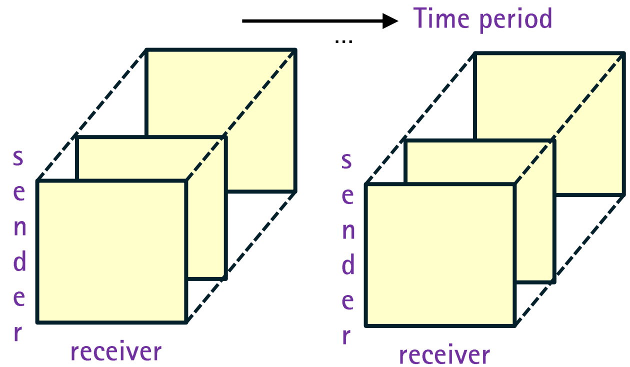
Real-world Data – Representation

Another key characteristic : sparsity



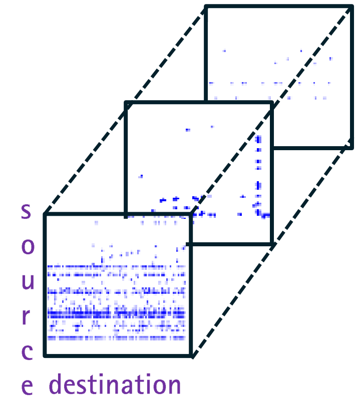
Author x Keyword x Year

Bibiliometric Data



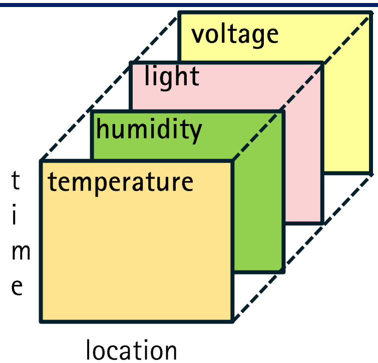
Sender x Receiver x Keyword x Time period

Email Data



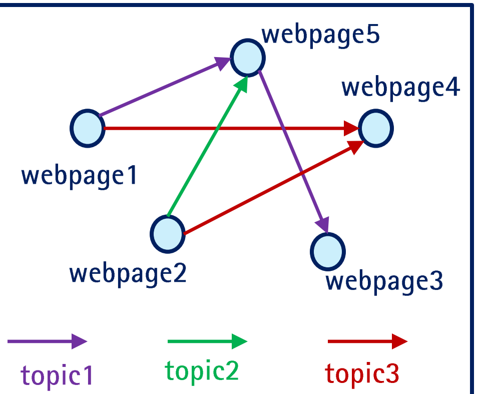
Src x Dest x Time

Network Traffic Data



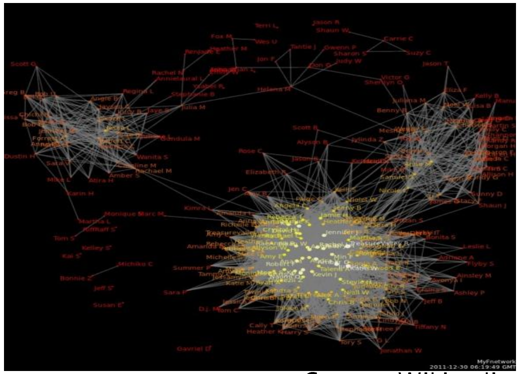
Time x Location x Type

Environmental Sensor Monitoring



Page x Page x Topic

Web graph



Source: Wikipedia

Person x Person x Relation

Social network graph

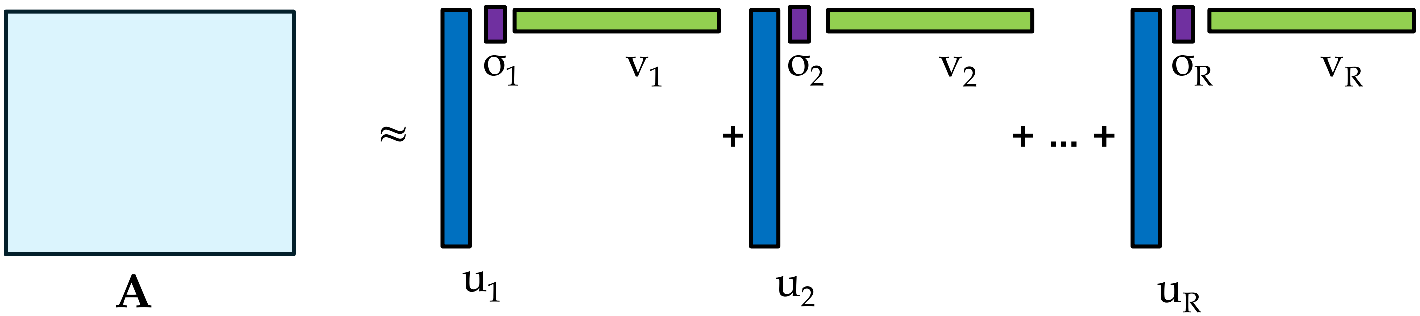
Real-world Data – Analysis

Linear Algebra popular for two-dimensional data and graph analysis

- Linear algebra methods – SVD or Eigen Value Decomposition

$$\mathbf{A} = U\Sigma V^T$$

$$\mathbf{A} \approx \sum_{r=1}^{R} \sigma_r \, u_r \circ v_r$$



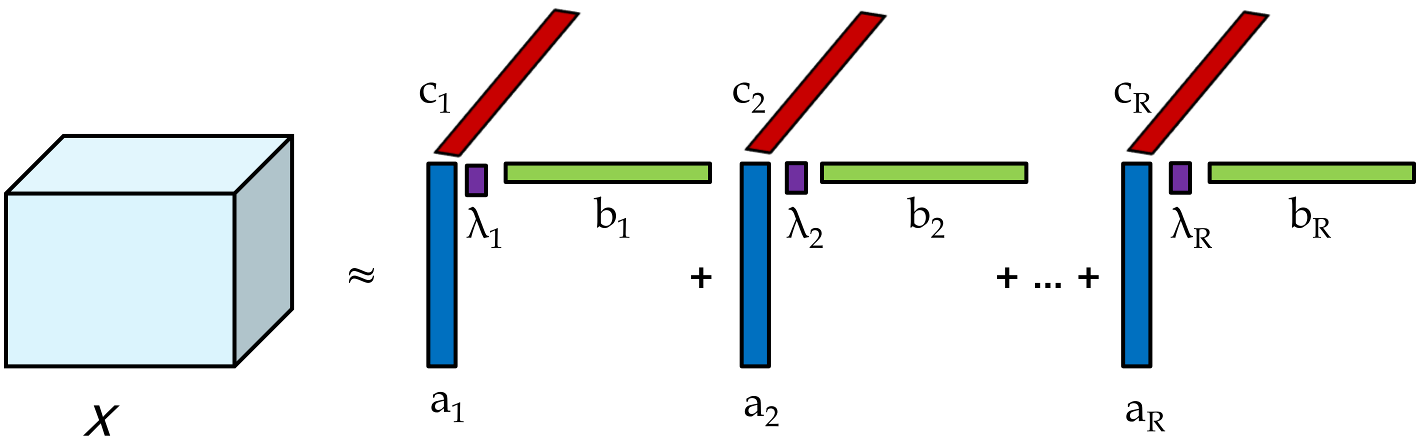
- Popular graph and data analysis problems using linear algebra
  - Page Rank, HITS, Latent Semantic Analysis

Real-world Data – Analysis

Multi-linear Algebra for analyzing higher-dimensional data with multiple aspects and multi-link graphs

- E.g. Higher-order tensor decomposition

$$X \approx \sum_{r=1}^{R} \lambda_r a_r \circ b_r \circ c_r$$



Presentation Roadmap

Motivation

Background

Techniques

Performance Results

Summary & Forward Work

Tensor Operations

- n-Mode vector product
  - Multiplying a tensor by a vector in mode n

$\mathbf{X}$: M x N x K tensor
$\mathbf{v}$: N x 1 vector
$\mathbf{B}$: M x K matrix

$$\mathbf{B} = \mathbf{X} \times_2 \mathbf{v}$$

$$b_{ik} = \sum_{j=1}^{N} x_{ijk} v_j$$

- n-Mode matrix product
  - Multiplying a tensor by a matrix in mode n

$\mathbf{X}$: M x N x K tensor
$\mathbf{A}$: R x N matrix
$\mathbf{Y}$: M x R x K tensor

$$\mathbf{Y} = \mathbf{X} \times_2 \mathbf{A}$$

$$y_{irk} = \sum_{j=1}^{N} x_{ijk} a_{jr}$$

  - Sequence of n-Mode products

$$\mathbf{Z} = \mathbf{X} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$$

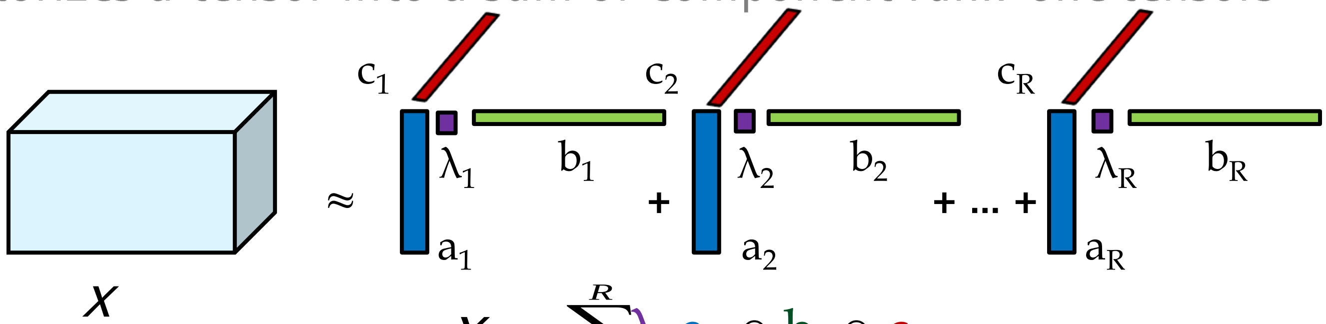
  - Sequence of *all-but-one* n-Mode products

$$\mathbf{Z} = \mathbf{X} \times_2 \mathbf{B} \times_3 \mathbf{C}$$

Tensor Decompositions

CP decomposition

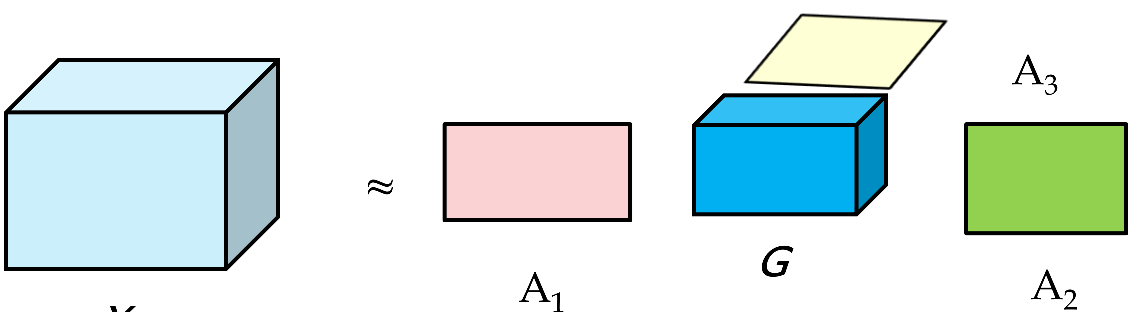
- Factorizes a tensor into a sum of component *rank-one* tensors



$$\mathcal{X} \approx \sum_{r=1}^{R} \lambda_r \, a_r \circ b_r \circ c_r$$

Tucker decomposition

- Factorizes a tensor into a core tensor and a set of factor matrices



$$\mathcal{X} \approx \mathbf{G} \times_1 A_1 \times_2 A_2 \times_3 A_3$$

Presentation Roadmap

Motivation

Tensor Background

Techniques

Performance Results

Summary & Forward Work

Our Approach

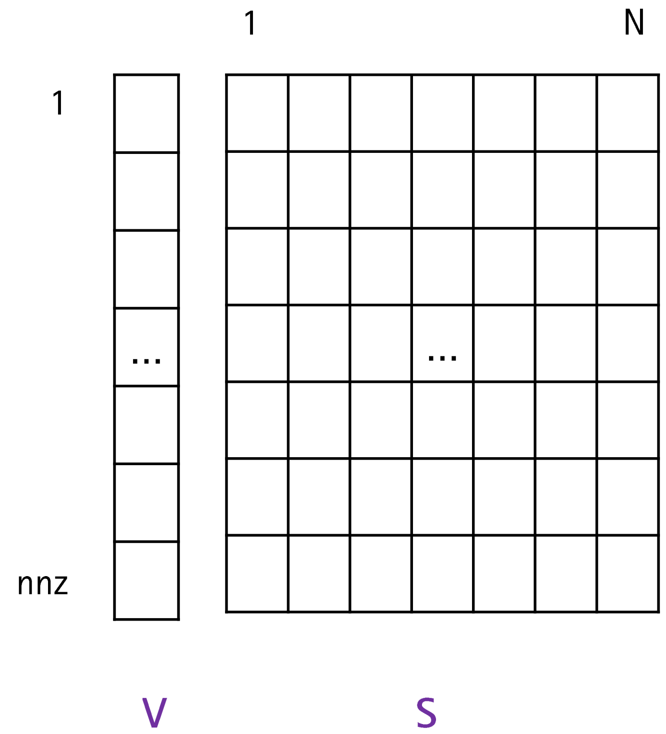
Our approach towards optimizing sparse tensor computations

- Define new efficient sparse tensor formats
  - Efficiently handle the sparseness in input data
- Improve computational efficiency
  - Reduce/Avoid unnecessary computations
  - Improve data reuse
  - Extract maximal parallelism
- Handle large sparse data sets
  - Avoid memory blowup in storing tensors

Sparse Tensor Formats

Coordinate format

- Non-zero values and subscripts stored
- Pros
  - Simple, flexible
  - Computation need not be specialized for different mode orderings
- Cons
  - Simplicity can be adverse
    - Loss of data locality based on the access pattern of the non zeros
  - Inefficient to represent dense sub-tensors within the sparse tensor
    - e.g. intermediate tensors resulting in sequence of n-Mode matrix products



V : vector storing the non-zero values
S : matrix storing the co-ordinates (along N modes) of each non-zero value

Coordinate sparse tensor storage

Sparse Tensor Formats

Our new formats

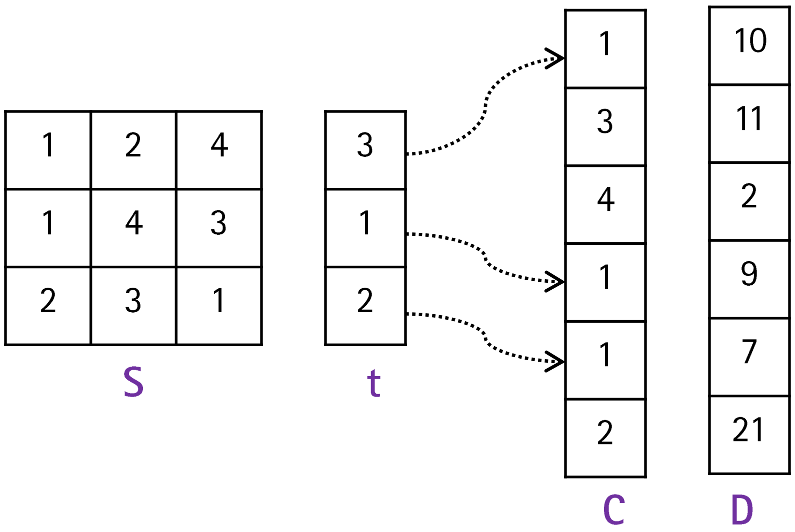
- Motivation behind them
  - Support frequently used mode-specific tensor operations
  - Support to efficiently represent dense sub-tensors within a sparse tensor
- Mode generic format
- Mode specific format

Mode-specific Sparse Tensor Format

Example

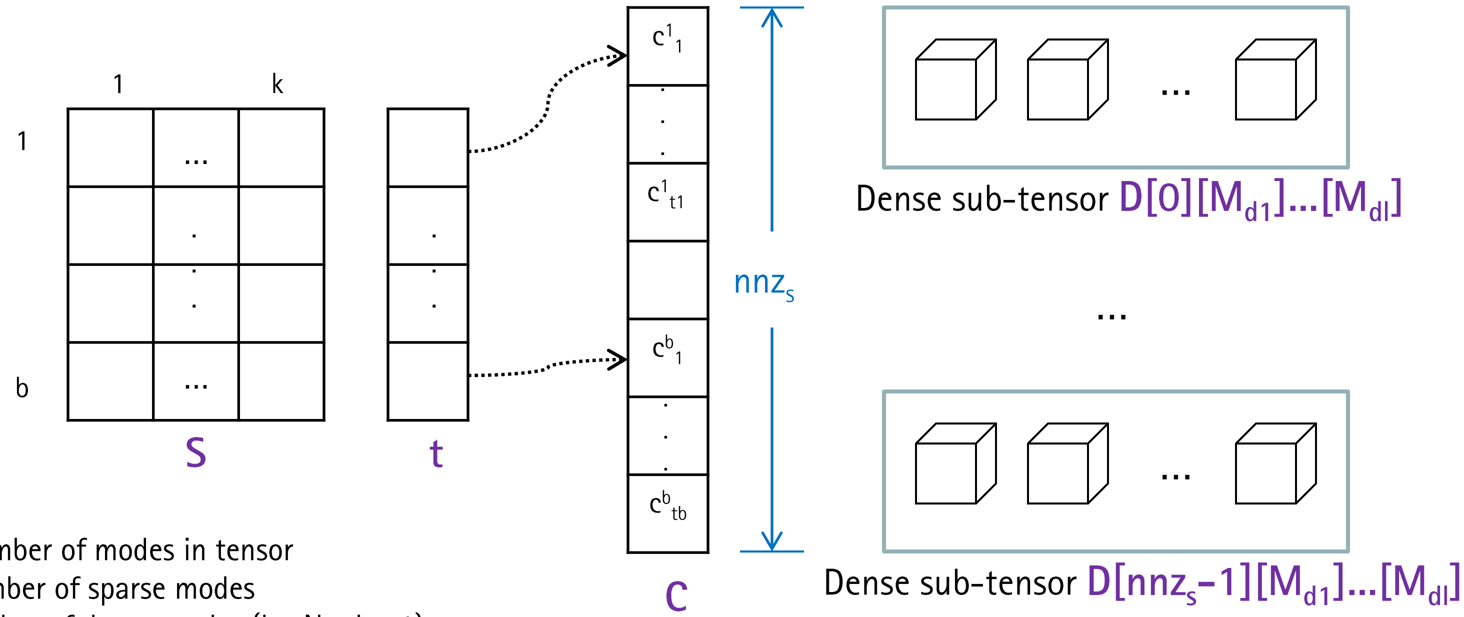
Tensor : X : 4 x 4 x 4 x 4

Non-zero Subscripts	Values
(1,2,1,4)	10
(1,2,3,4)	11
(1,4,1,3)	9
(2,3,1,1)	7
(1,2,4,4)	2
(2,3,2,1)	21



Let us assume a mode-specific storage along mode 3 for e.g.

Mode-specific Sparse Tensor Format



N : number of modes in tensor
k : number of sparse modes
l : number of dense modes ($l = N - k - 1$)
$\{d_1,\ldots,d_l\}$: dense modes
nnz_s : number of dense sub-tensors ($nnz_s = \sum_{i=1}^{b} t[i]$)
M_n : size of tensor along n^{th} mode
b : number of distinct coordinates (say buckets) along the sparse modes

S : b x k matrix storing the coordinates of the k non-candidate sparse modes in each of the b buckets
t : b x 1 vector storing the number of non-zero candidate mode indices in each bucket
C : nnz_s x 1 vector storing the non-zero candidate sparse mode indices
$D[nnz_s][M_{d1}]...[M_{dl}]$: dense sub-tensors

Mode-generic Sparse Tensor Format

Example

Tensor : X : 4 x 4 x 4 x 4

Non-zero Subscripts	Values
(1,2,1,4)	10
(1,2,3,4)	11
(1,4,1,3)	9
(2,3,1,1)	7
(1,2,4,4)	2
(2,3,2,1)	21

A

1	2	1	1
1	1	2	3
2	1	2	1

Tensor : Y: 4 x 4 x 3x 4

Non-zero Subscripts	Values
(1,2,1,4)	23
(1,2,2,4)	38
(1,2,3,4)	44
(1,4,1,3)	9
(1,4,2,3)	9
(1,4,3,3)	18
(2,3,1,1)	49
(2,3,2,1)	28
(2,3,3,1)	35

$$Y = X \times_3 \mathbf{A}$$

Mode-generic Sparse Tensor Format

Example

Tensor : X : 4 x 4 x 4 x 4

Non-zero Subscripts	Values
(1,2,1,4)	10
(1,2,3,4)	11
(1,4,1,3)	9
(2,3,1,1)	7
(1,2,4,4)	2
(2,3,2,1)	21

A

1	2	1	1
1	1	2	3
2	1	2	1

Tensor : Y: 4 x 4 x 3x 4

Non-zero Subscripts	Values
(1,2,1,4)	23
(1,2,2,4)	38
(1,2,3,4)	44
(1,4,1,3)	9
(1,4,2,3)	9
(1,4,3,3)	18
(2,3,1,1)	49
(2,3,2,1)	28
(2,3,3,1)	35

$Y = X \times_3 \mathbf{A}$

Y

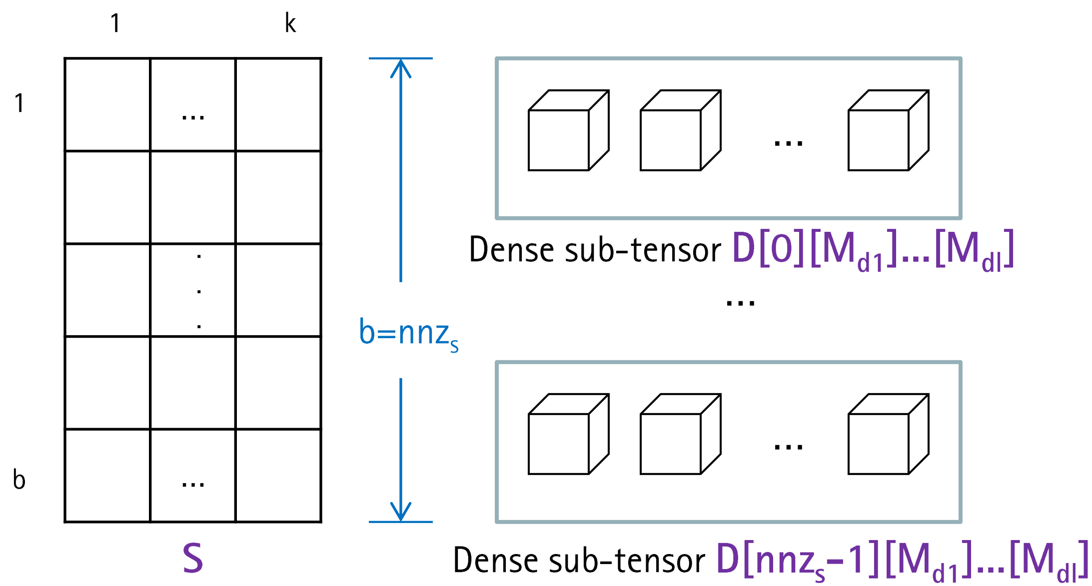
S

1	2	4
1	4	3
2	3	1

D

23	38	44
9	9	18
49	28	35

Mode-generic Sparse Tensor Format



N : number of modes in tensor
k : number of sparse modes
l : number of dense modes ($l = N - k$)
$\{d_1,\ldots,d_l\}$: l-tuple of dense modes
nnz_s : number of dense sub-tensors
M_n : size of tensor along n^{th} mode
b : number of distinct coordinates (say buckets) along the sparse modes ($nnz_s = b$)

S : b x k matrix storing the coordinates of the k sparse modes in each of the b distinct buckets

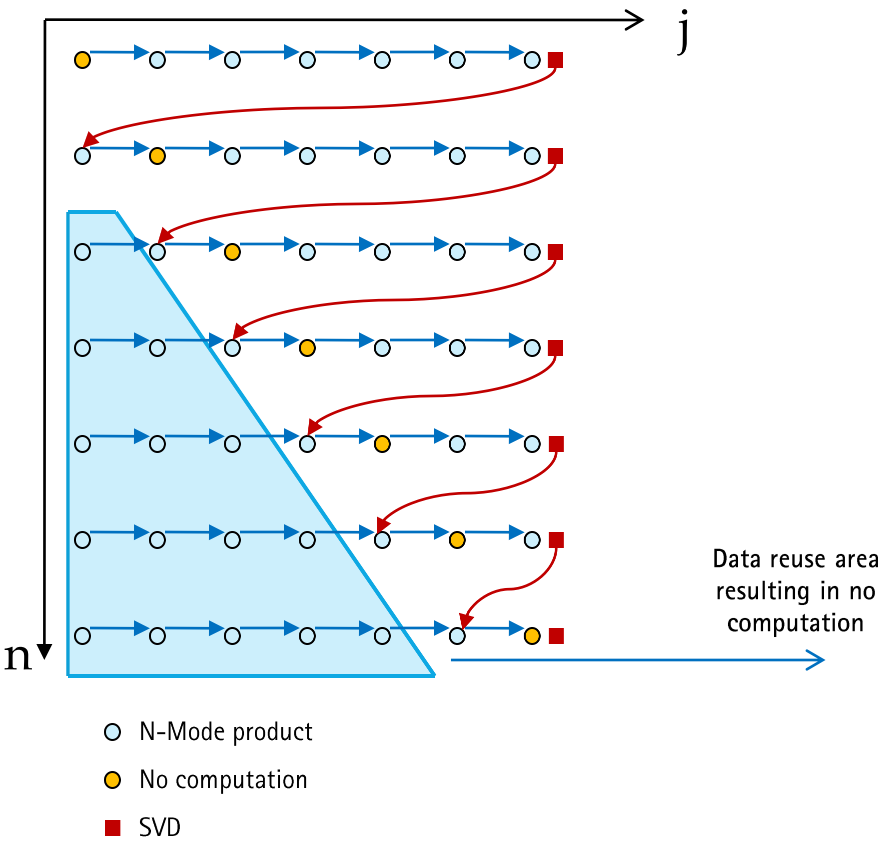
$D[nnz_s][M_{d1}]\ldots[M_{dl}]$: dense sub-tensors

Data Reuse Optimization

Tucker decomposition algorithm (HOOI method)

repeat
  for $n = 1 \ldots N$ do
    $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)T} \cdots \times_{n-1} \mathbf{A}^{(n-1)T} \times_{n+1} \mathbf{A}^{(n+1)T} \cdots \times_N \mathbf{A}^{(N)T}$
    $\mathbf{A}_n = J_n$ leading left singular vectors of $\mathbf{Y}_n$
  end for
  $\mathcal{G} = \mathcal{Y} \times_N \mathbf{A}^{(N)T}$
until convergence

No. of reuses : $\frac{N^2}{2} - \frac{3N}{2} + 1$

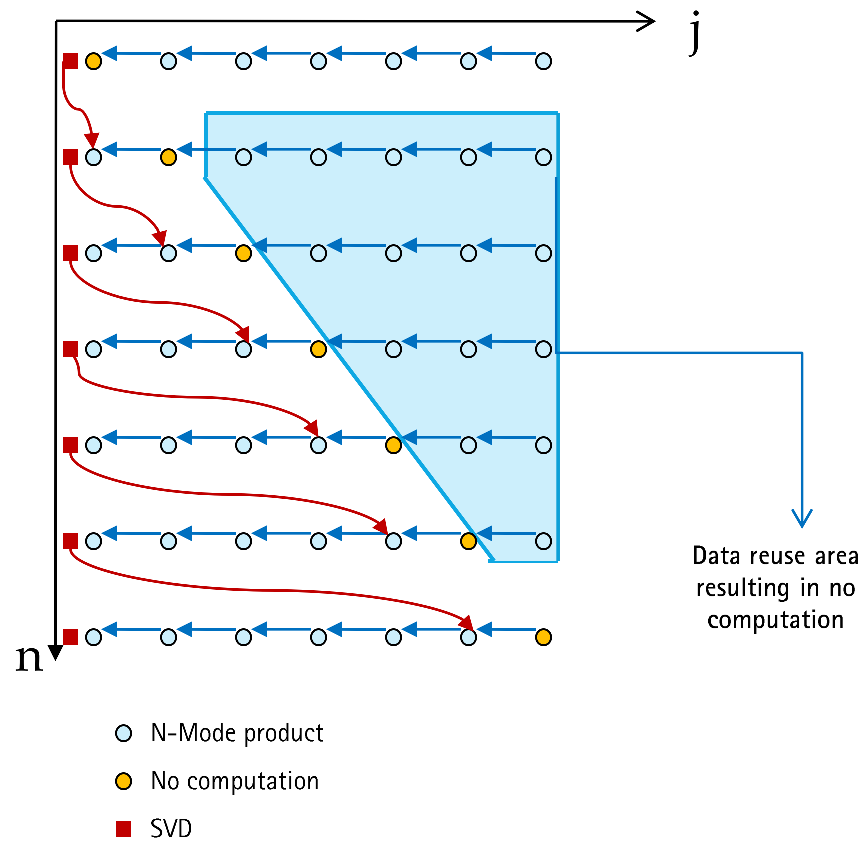


Data Reuse Optimization

Tucker decomposition algorithm
(HOOI method)

repeat
  for $n = 1 \ldots N$ **do**
    $\mathcal{Y} = \mathcal{X} \times_N \mathbf{A}^{(N)T} \cdots \times_{n+1} \mathbf{A}^{(n+1)T} \times_{n-1} \mathbf{A}^{(n-1)T} \cdots \times_1 \mathbf{A}^{(1)T}$
    $\mathbf{A}_n = J_n$ leading left singular vectors of $\mathbf{Y}_n$
  end for
  $\mathcal{G} = \mathcal{Y} \times_N \mathbf{A}^{(N)T}$
until convergence

No. of reuses: $\frac{N^2}{2} - \frac{3N}{2} + 1$

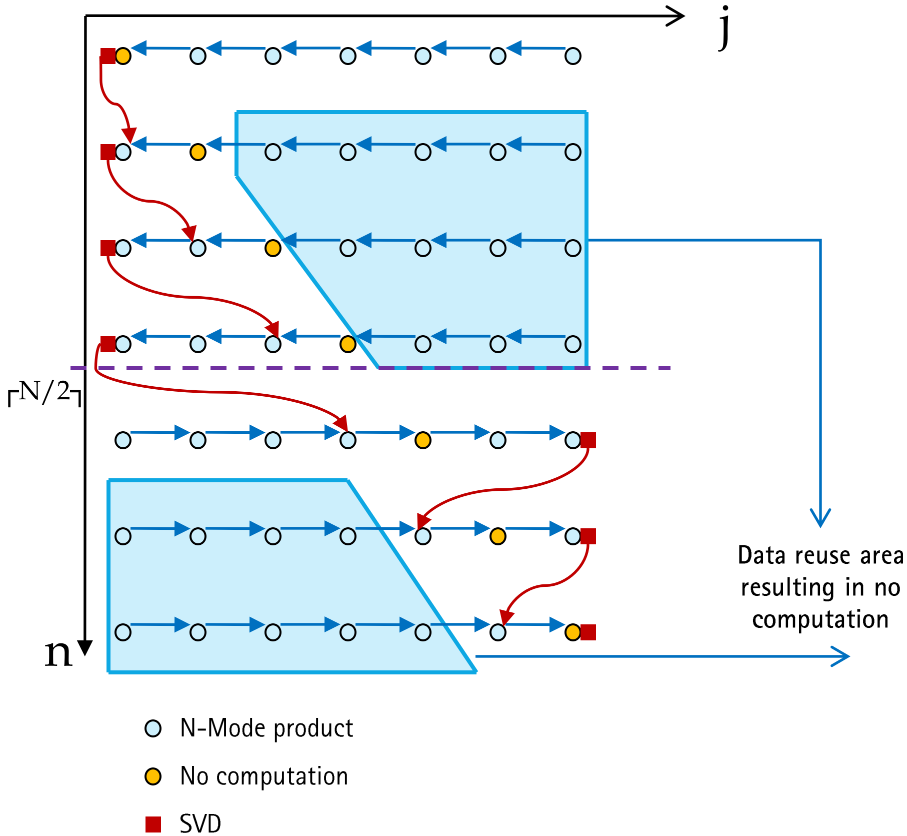


$\mathbf{X} \times_m \mathbf{A} \times_n \mathbf{B} = \mathbf{X} \times_n \mathbf{B} \times_m \mathbf{A} \ (m \neq n)$

Data Reuse Optimization

Tucker decomposition algorithm (HOOI method)

repeat
  for $n = 1 \ldots \lceil \frac{N}{2} \rceil$ do
    $\mathcal{Y} = \mathcal{X} \times_N \mathbf{A}^{(N)T} \cdots \times_{n+1} \mathbf{A}^{(n+1)T} \times_{n-1} \mathbf{A}^{(n-1)T} \cdots \times_1 \mathbf{A}^{(1)T}$
    $\mathbf{A}_n = J_n$ leading left singular vectors of $\mathbf{Y}_n$
  end for
  for $n = \lceil \frac{N}{2} \rceil + 1 \ldots N$ do
    $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)T} \cdots \times_{n-1} \mathbf{A}^{(n-1)T} \times_{n+1} \mathbf{A}^{(n+1)T} \cdots \times_N \mathbf{A}^{(N)T}$
    $\mathbf{A}_n = J_n$ leading left singular vectors of $\mathbf{Y}_n$
  end for
  $\mathcal{G} = \mathcal{Y} \times_N \mathbf{A}^{(N)T}$
until convergence



No. of reuses: $\geq \frac{N^2}{2} - \frac{3N}{2} + \frac{(N-2)^2+1}{4} + 1$

Memory-efficient Scalable Optimization

Memory blowup problem in Tucker decomposition

- Intermediate tensors in computation
  - Pros
    - Reduces redundant computations
  - Cons
    - Requires large storage
    - Memory blowup due to large storage
  - Storage vs computation trade-off

repeat
  for $n = 1 \ldots N$ do
    $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)T} \cdots \times_{n-1} \mathbf{A}^{(n-1)T} \times_{n+1} \mathbf{A}^{(n+1)T} \cdots \times_N \mathbf{A}^{(N)T}$
    $\mathbf{A}_n = J_n$ leading left singular vectors of $\mathbf{Y}_n$
  end for
  $\mathcal{G} = \mathcal{Y} \times_N \mathbf{A}^{(N)T}$
until convergence

- Kolda et al. Memory Efficient Tucker (MET) solution
  - Categorize modes : *elementwise* and *standard* based on available memory (using heuristics)
    - *elementwise*: no intermediates along the modes
    - *standard*: intermediates stored along the modes

Memory-efficient Scalable Optimization

Our approach

- Uses mode-generic sparse formats for intermediate tensors in computation
  - State-of-the-art approach uses dense formats for intermediate tensors
- Optimally categorizes modes as *elementwise* and *standard* based on available memory
  - Optimal order of n-Mode products in a sequence that reduces total computation cost and total memory consumption
- Uses data reuse optimization
  - to reuse intermediate tensors
  - reduce redundant computations

Presentation Roadmap

Motivation

Tensor Background

Techniques

Performance Results

Summary & Forward Work

Performance Results

Benchmarking Tucker decomposition

- Dual socket quad core Intel Xeon E5504 2GHz processor
- Timed *all but one* sequence of tensor matrix products
- Enron data set
  - Number of modes: 4
  - dimensionality of input tensor: 1000 x 1000 x 1100 x 200
  - Non-zero density: 0.0025% (number of non-zeros = 5.5M)

Version	Time (s)
Baseline	175.17
Kolda et al. Approach	21.79
Our Approach (partial data reuse)	9.29
Our Approach (optimal data reuse)	**7.12**

Our sequential version: 3x over existing approach

Time for one iteration; typically 75-100 iterations

Performance Results

Benchmarking Tucker decomposition

- Timed *all but one* sequence of tensor matrix products
- Enron data set
- Timed parallel code (with optimal data reuse)

Processors	Time (s)
1	7.12
2	6.25
4	3.80
8	2.57

Our parallel version: 8.5x over existing approach's sequential version

Performance Results

Benchmarking Tucker decomposition

- Timed *all but one* sequence of tensor matrix products
- Synthetic tensors

Data set	Parameters	
	Size	Non-zeros
Tensor 1	28 x 501 x 24 x 8	26400
Tensor 2	1000 x 1000 x 200 x 12	686400

Version	Tensor 1	Tenosr 2
	Time (s)	Time (s)
Baseline	0.751	23.47
Kolda et al. Approach	0.065	2.04
Our Approach (optimal data reuse)	**0.050**	**1.78**

Presentation Roadmap

Motivation

Tensor Background

Techniques

Performance Results

Summary & Forward Work

Summary & Forward Work

What we do for optimizing sparse tensor computations

- Improve computation speedup
  - Reduce/Avoid unnecessary computations
  - Improve data reuse
  - Extract maximal parallelism
- Efficiently handle the sparseness in input data
  - New sparse formats
- Handle large problem sizes
  - Avoid memory blowup in storing tensors

What we plan to do

- Optimizations for choosing the right sparse tensor formats
- Optimizations based on input sparse tensor structure