Parallelizing and Scaling Tensor Computations*

Muthu Baskaran, Benoit Meister, Richard Lethin

Reservoir Labs

* Patent Pending Technology

Introduction

ENSIGN (Exascale Non-Stationary Graph Notation)

- Goal
  - Optimized tensor toolbox for dynamic graph analytics
    - Produce/provide optimized tensor computations for large-scale parallel systems
- Features
  - High-performance implementation of different variants of tensor decomposition methods
    - Scalable optimizations for tensor computations
  - Automatic parallelization and data locality optimizations
    - Inter-operate with Reservoir Labs' auto-parallelizing compiler R-Stream

Presentation Roadmap

About ENSIGN Tensor Toolbox

Optimization and Parallelization Techniques

Performance Evaluation

Summary & Forward Work

ENSIGN Tensor Toolbox

Available in two versions

- ETTB++v3.5.1
  - Accelerated C++ version of tensor toolbox built on top of **Sandia National Laboratories** C++ Tensor Toolbox v1.0.2
- ETTB v1.0
  - Accelerated C version of tensor toolbox

Toolbox of core algorithms

- CP decomposition variants
  - CP-ALS, CP-APR[1], CP-APR-PDNR[2], INDSCAL
- Tucker decomposition variants
  - HOOI, memory-efficient HOOI[3]
- Methods for "low-rank updates"[4]
- Coupled (Joint) tensor decomposition[5]
- Standardized tensor decomposition[6]

[1][Chi, Kolda 2012], [2][Hansen et al, 2014], [3][Kolda et al, 2008], [4][Ohara 2010], [5][MATLAB CMTF Toolbox, Acar], [6][Brown et al, 2014]

Presentation Roadmap

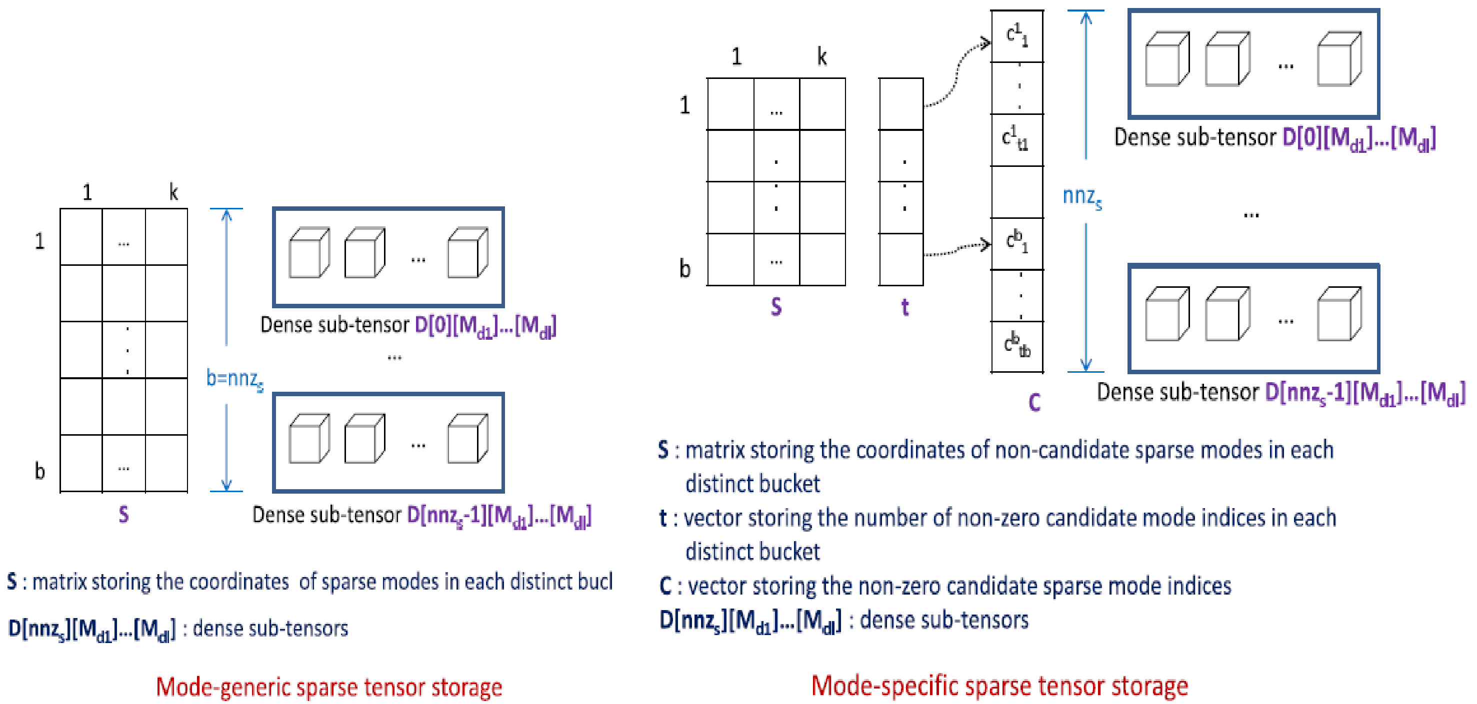
About ENSIGN Tensor Toolbox

Optimization and Parallelization Techniques

Performance Evaluation

Summary & Forward Work

New Data Structures for Scaling the Computations



(a) Mode generic format

(b) Mode specific format

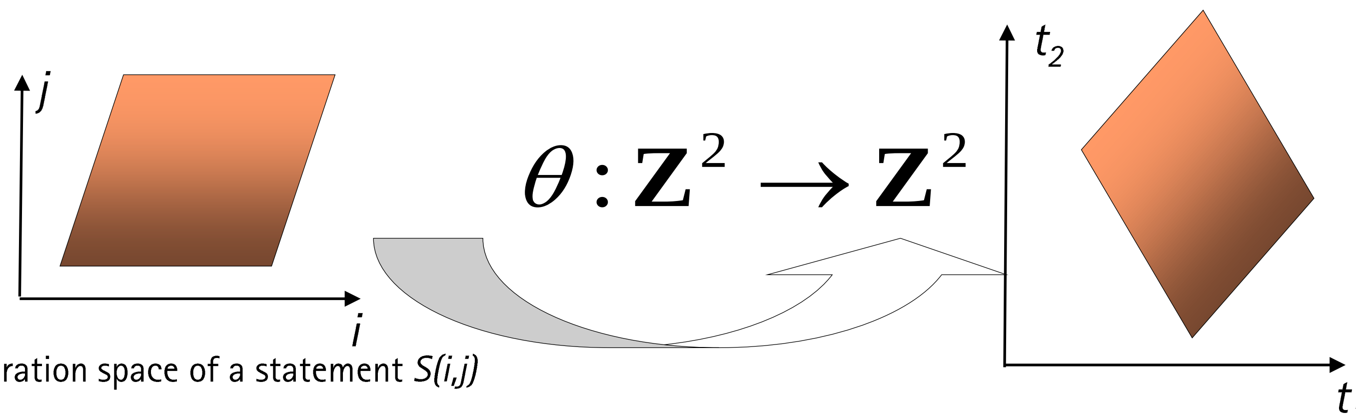
Baskaran et al., "Efficient and Scalable Computations with Sparse Tensors", IEEE HPEC 2012.

Optimizing Dense Computations

R-Stream optimizations

- State-of-the-art polyhedral compiler algorithms
- Optimize dense computations automatically
- Advanced compiler techniques using “polyhedral model” for
  - Parallelization, Locality, Contiguity, Vectorization, …
  - “Affine” transformations of arbitrary loop nests

Affine Schedule ϑ maps iterations to multi-dimensional space-time



Initial Iteration space of a statement *S(i,j)*

Transformed Iteration space of *S(i,j)*

Optimizing Sparse Computations

Challenge: Optimizing “sparse” computations

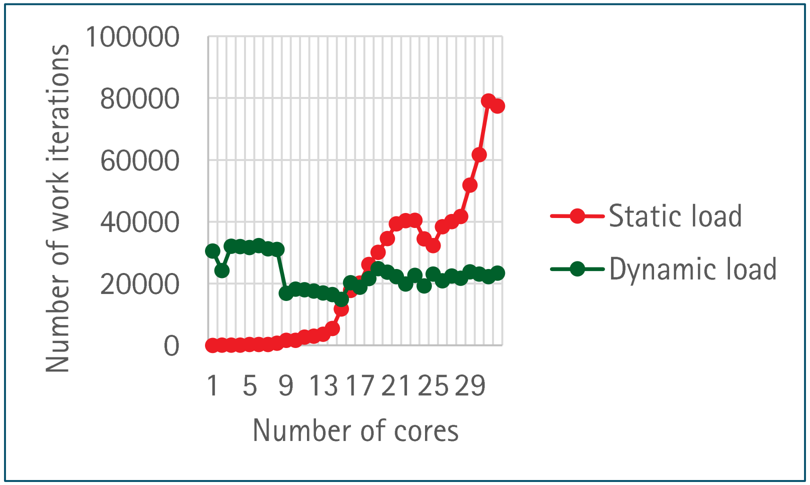
- A “data-driven” scheduling problem
- Need to efficiently handle irregular memory accesses
- Current parallelization efforts (including **R-Stream**) have scope for improvement
  - Parallelism, synchronization, data locality, etc.

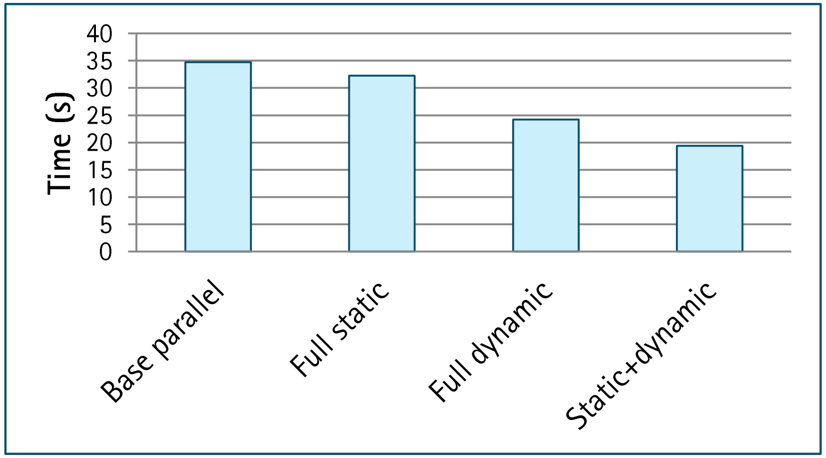
Goals of our improvisation techniques

- Uncover more concurrency
- Reduce synchronization
- Improve data locality
- Achieve load balance
- Reduce scheduling overhead

Mixed Static and Dynamic Runtime Scheduling

- Static scheduling - poor load balance, low scheduling overhead
- Dynamic scheduling - good load balance, high scheduling overhead
- Our Approach – Achieves the pros of both schemes
  - One dynamic scheduling iteration to get a load balanced pattern
  - Static scheduling using the pattern for later iterations
  - good load balance, low scheduling overhead





Baskaran et al., "Low-overhead Load-balanced scheduling for Sparse Tensor Computations," IEEE HPEC 2014.

Improved Data Locality

- Memory-hierarchy aware approach
  - Task distribution across processor cores in the dynamic scheduling iteration governed by
    - Data touched by them
    - Memory in which data resides
  - Over-loaded cores "steal" tasks from "topologically" closer neighbors that are under-loaded
    - NUMA topology in shared memory systems
  - Facilitate data sharing across cores

Baskaran et al., "Low-overhead Load-balanced scheduling for Sparse Tensor Computations," IEEE HPEC 2014.

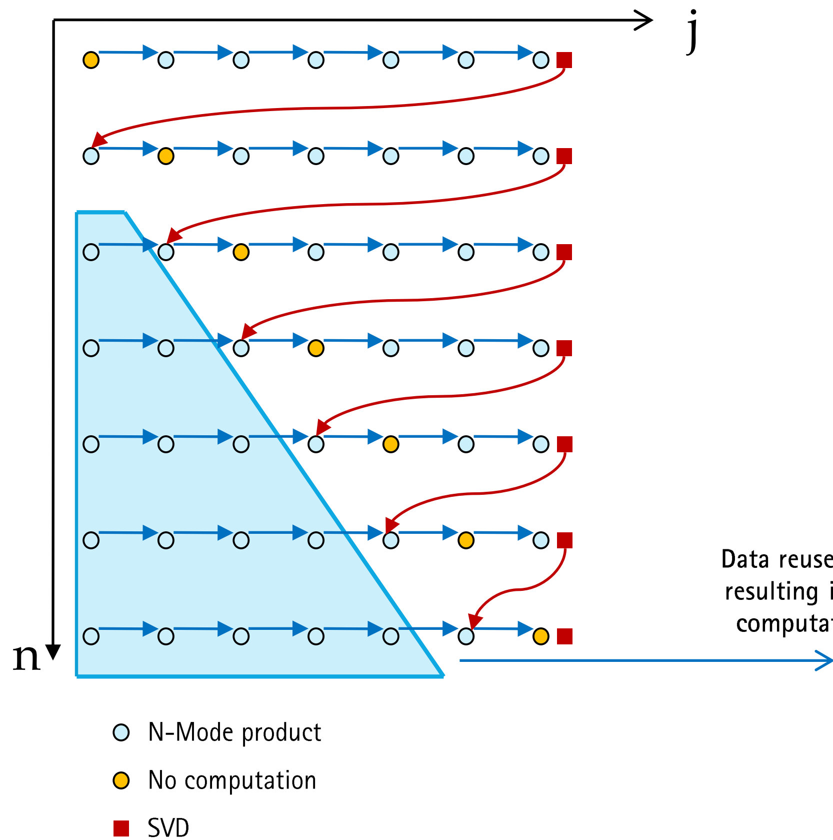
Optimizing Tucker Decomposition

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 : $\frac{N^2}{2} - \frac{3N}{2} + 1$



Baskaran et al., "Efficient and Scalable Computations with Sparse Tensors", IEEE HPEC 2012.

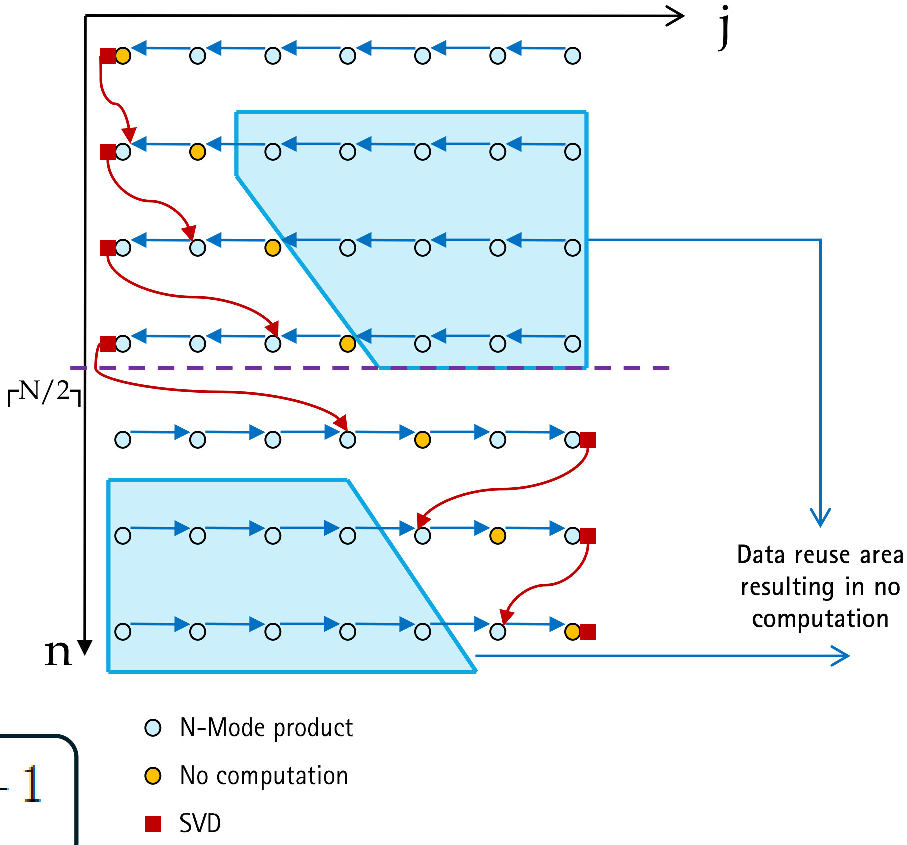
Optimizing Tucker Decomposition

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: $\geq \frac{N^2}{2} - \frac{3N}{2} + \frac{(N-2)^2+1}{4} + 1$



Baskaran et al., "Efficient and Scalable Computations with Sparse Tensors", IEEE HPEC 2012.

Optimizing Tucker Decomposition

Memory-efficient Scalable Optimization

Memory blowup problem in Tucker decomposition

- Intermediate tensors in computation
  - Storage vs computation trade-off
- 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 (similar to Kolda et al. 2008)
  - Optimal order of n-Mode products in a sequence that reduces total computation cost and total memory consumption
- Uses data reuse optimization

Baskaran et al., "Efficient and Scalable Computations with Sparse Tensors", IEEE HPEC 2012.

Presentation Roadmap

About ENSIGN Tensor Toolbox

Optimization and Parallelization Techniques

Performance Evaluation

Summary & Forward Work

Performance Evaluation

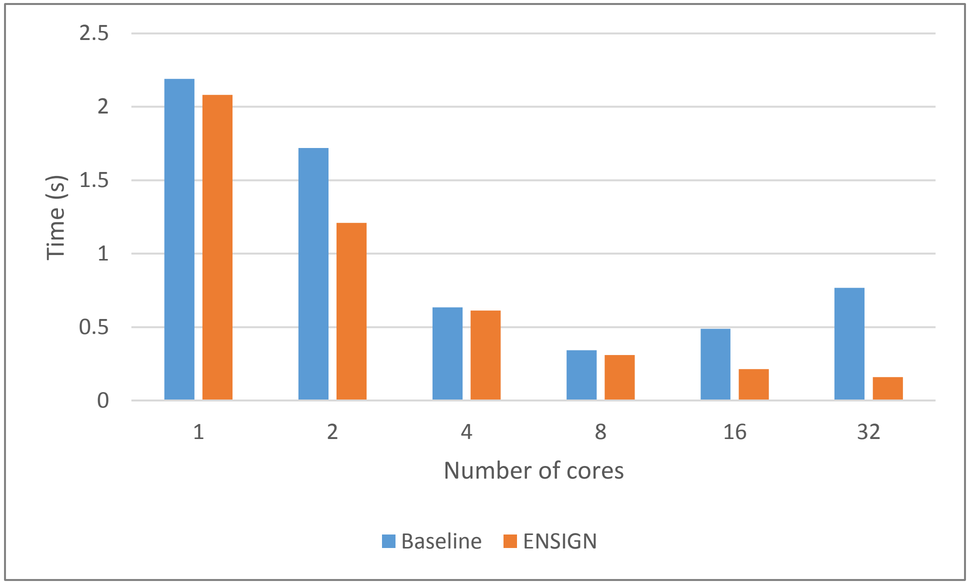
Benchmarked different methods on different sized datasets

- Intel Xeon E5-4620 2.2 GHz (Quad socket 8-core)

Reservoir Labs Tensorstation™

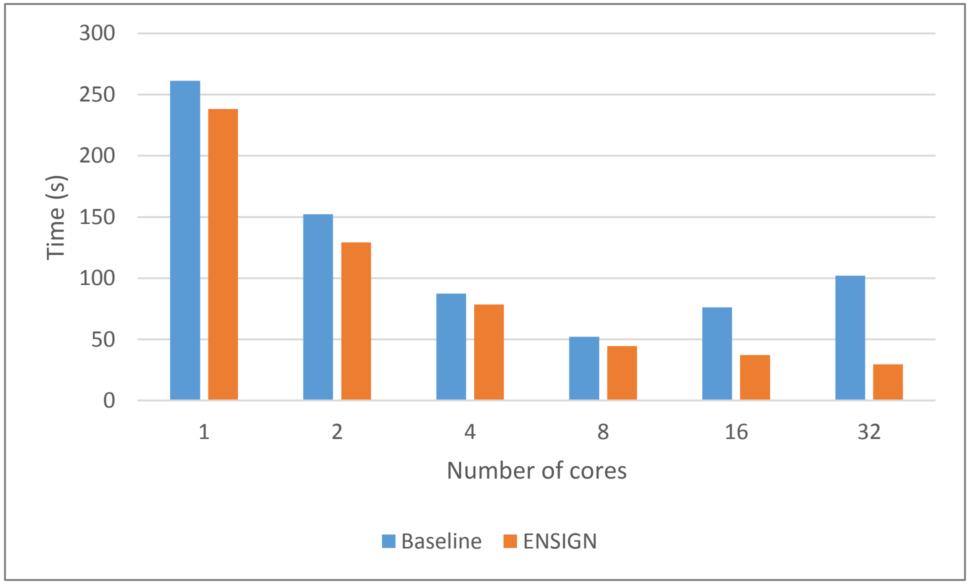
Performance Evaluation
CP-ALS evaluation

Tensor	Size	Non-zeros	#iterations timed
Facebook	63891 x 63891 x 1591	737934	50



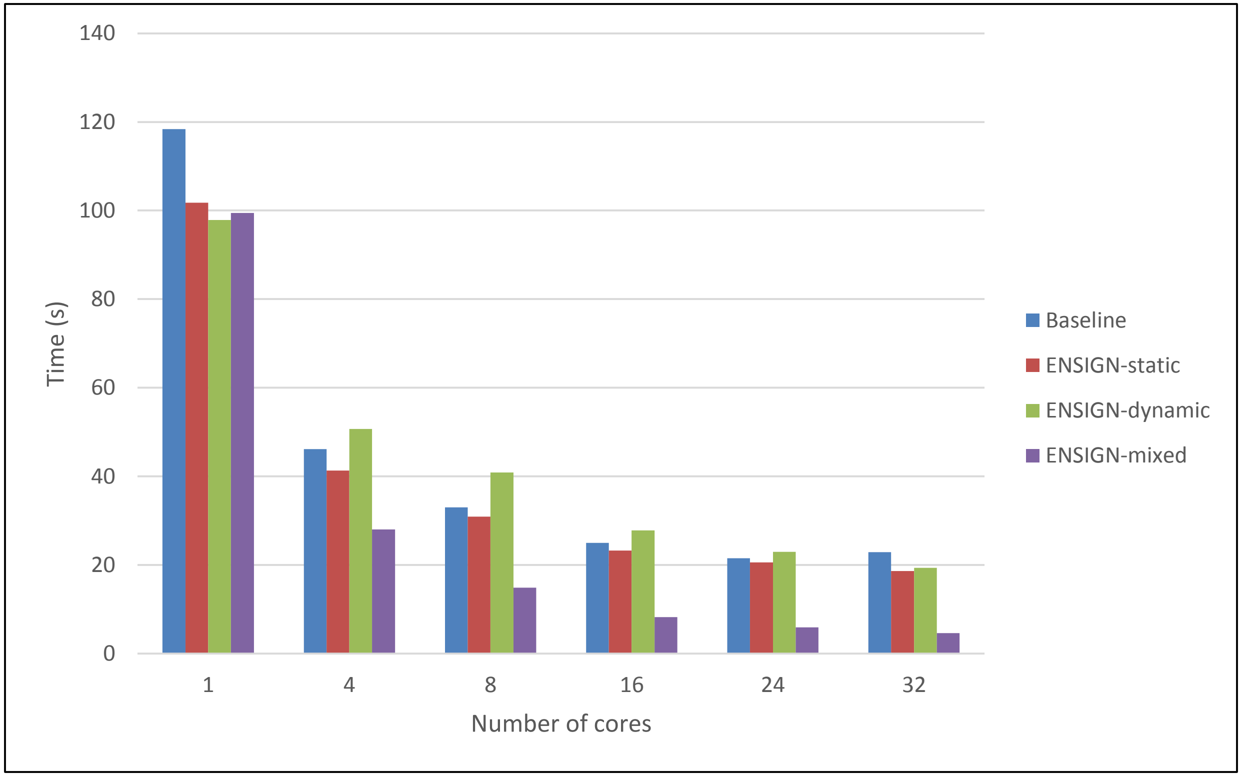
Performance Evaluation
CP-ALS evaluation

Tensor	Size	Non-zeros	#iterations timed
Cyber	565872 x 795 x 13868 x 21862	4865458	50



Performance Evaluation
CP-APR evaluation

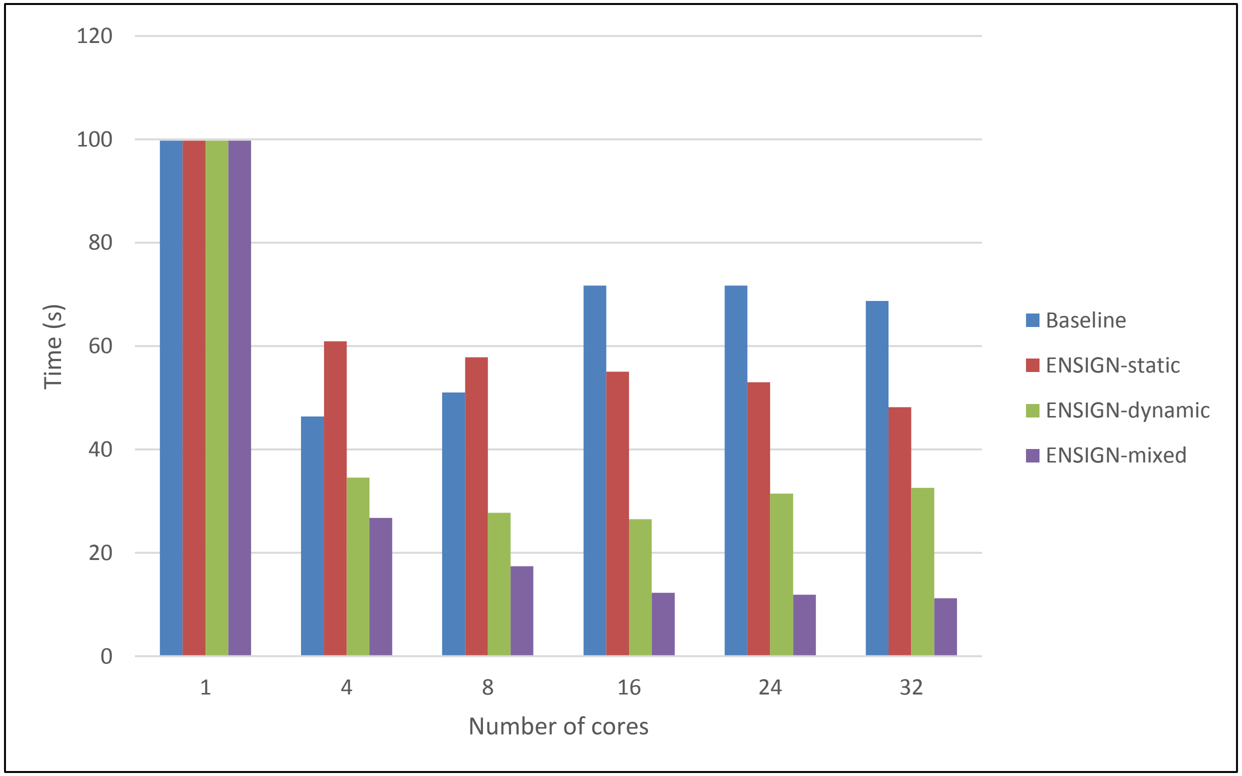
Tensor	Size	Non-zeros	#iterations timed
Facebook	63891 x 63891 x 1591	737934	190



Performance Evaluation

CP-APR evaluation

Tensor	Size	Non-zeros	#iterations timed
Cyber	14811811 x1899 x1899 x 3 x 6067	2085108	50



Performance Evaluation
Tucker evaluation

- Timed *all but one* sequence of tensor matrix products
- Data set
  - Number of modes: 4
  - dimensionality of input tensor: 1000 x 1000 x 1100 x 200
  - 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 Evaluation
Tucker evaluation

- Timed parallel code (with optimal data reuse)

Number of cores	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

Presentation Roadmap

About ENSIGN Tensor Toolbox

Optimization and Parallelization Techniques

Performance Evaluation

Summary & Forward Work

Summary of ENSIGN Techniques

Performance:

- Developed techniques to effectively parallelize and scale large sparse and dense tensor computations
  - New efficient sparse formats
  - Extract maximal parallelism
  - Extract data locality & data reuse
    - Reduce data movement
  - Reduce/Avoid unnecessary computations

Capability:

- Software released to customers
- Demonstrated on real-world problems

Ongoing and Forward Work

More focus on applying ENSIGN on real-world problems

- Genomics
- Cyber security

Enhancing usability of the tool

- Graphical User Interface
- Visualization of decompositions

Distributed-memory versions of tensor methods