Fast and Scalable Distributed Tensor Decompositions

Muthu Baskaran, Tom Henretty, James Ezick

Reservoir Labs

Presentation Outline

Context

- Tensor Decompositions
- ENSIGN

Problem Overview

- Challenges in Scalable Decompositions
- Data and Computation Distribution

Approach for Distributed Decompositions

- Distributed Sparse Tensor Data Structures
- Data Distribution Strategies
- Communication Minimization

Results

Conclusion

Tensor Analysis

Tensors

- Provide a natural representation for multi-dimensional data
- Suitable for a variety of data sources (cyber, genomics, GEOINT, …)

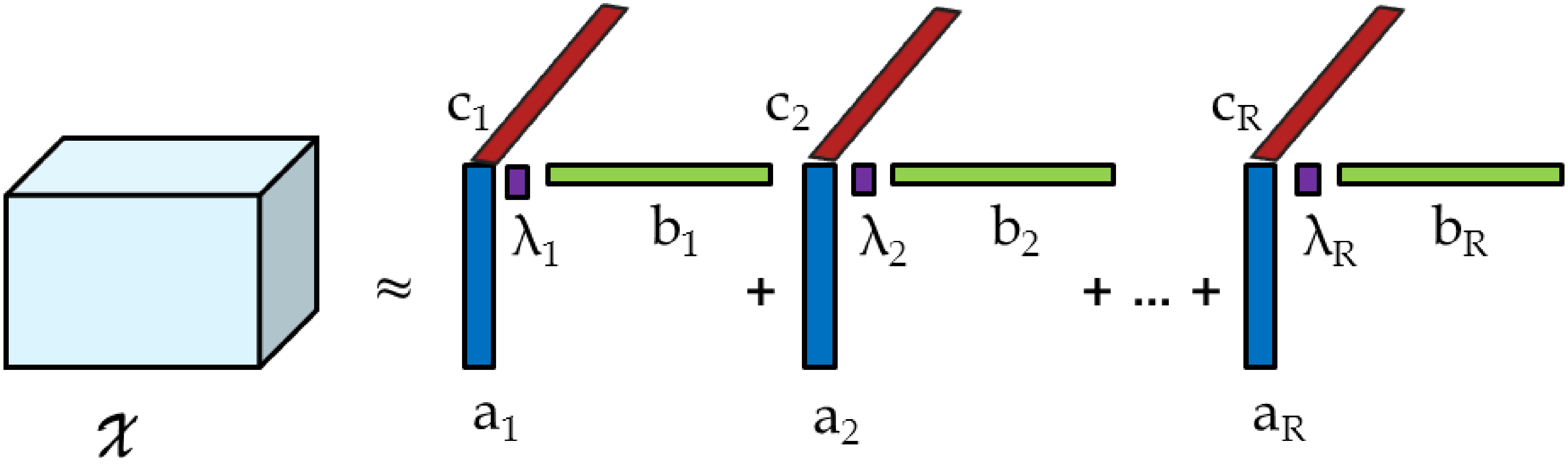
Tensor Analysis Generalizes Matrix Analysis

- "Graph Analysis in the Language of Linear Algebra"
- Becomes … "Multi-link Graph Analysis in the Language of Multi-linear (Tensor) Algebra"
- Semantics are higher dimensions and "first class" with links
- Gives more complete and "contextual" insights into data

This talk

- Show how do this faster and more efficiently on advanced HPC systems to meet modern application demands

Tensor Decompositions



CP Tensor Decomposition

Tensor is decomposed into a non-unique weighted sum of a pre-defined number of rank-1 components

- Break multidimensional data into distinct components
- Components reveal patterns and latent structure in the dataset

Exascale NonStationary Graph Notation (ENSIGN)

Driving Towards a Practical High-performance Data Analytics Tool

Class	Differentiating Specifics	Benefit to Analyst
Modeling (Capability)	First-order decomposition methods Second-order decomposition methods Joint tensor decompositions Multiple data distribution models Normalized decompositions Streaming decompositions … more coming	Breadth of models enabled Framework for graph fusion Platform for anomaly detection Sparsity-maximizing approaches Efficient update with arrival of new data Discovery of new behaviors through new components
Performance	Optimized sparse tensor data structures Mixed static/dynamic optimization Memory-efficiency optimizations Algorithmic improvements Shared memory parallelism Distributed memory parallelism Cloud-based optimizations	Extend the range, scale, and scope of analysis Analyze tensors of billion-scale and beyond Enable large rank decompositions Enable large number of mode decompositions Leverage HPC Systems Quick time-to-solution
Usability	GUI & CLI Python bindings C bindings QGIS support Virtual machine distributions Documented, Tested, Supported	Interactive large scale exploration In standard environments (e.g., Jupyter notebooks) Integration with existing corporate data lakes/pipelines Visualization Reliable install and operation Training, Someone to Call

Scalable Decompositions for HPC Systems

Challenge	Approach
Load-balanced parallel execution	• Light-weight load distribution (at the beginning of the decomposition)
Communication minimization	• Selective tensor and factor matrix distribution to minimize communication volume and frequency
Reduced memory footprint	• Selective re-computation of intermediate data (vs. storing large footprint intermediate data)
Minimal computations	• Efficient sparse tensor data structures to facilitate memory- and operation-efficient computations
Data locality	• Fusion of computations to increase thread-local operations with improved locality

HPEC 2017: Memory-efficient Parallel Tensor Decompositions [Best Paper Award]

Scalable Decompositions for HPC Systems

Challenge	Approach
Load-balanced parallel execution	• Light-weight load distribution (at the beginning of the decomposition)
Communication minimization	• Selective tensor and factor matrix distribution to minimize communication volume and frequency
Reduced memory footprint	• Selective re-computation of intermediate data (vs. storing large footprint intermediate data)
Minimal computations	• Efficient sparse tensor data structures to facilitate memory- and operation-efficient computations
Data locality	• Fusion of computations to increase thread-local operations with improved locality

HPEC 2019: Fast and Scalable Distributed Tensor Decompositions

Tensor Decomposition Method for Cyber Analysis

CP-APR Algorithm

Input: $\mathcal{X}$, $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    repeat
      Compute:
      $\Phi = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n}\mathbf{A}^{(m)})^T))(\odot_{m \neq n}\mathbf{A}^{(m)})$
      Compute inner convergence
      Compute: $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \Phi$
    until convergence
  end for
  Compute outer convergence
until convergence
Output: $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$

Chi, E., Kolda, T., On Tensors, Sparsity, and Nonnegative Factorizations, SIAM Journal on Matrix Analysis and Applications 33.4 (2012): 1272-1299.

"CP-APR" Method

- Models sparse count data
  - Poisson distribution
- Uses alternating Poisson regression (APR) for non-negative CP model
- Proven to be extremely suited for cyber data (which is sparse count data)

Tensor Decomposition Method for Cyber Analysis

CP-APR Algorithm

Input: $\mathcal{X}$, $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$

repeat → **Outer Optimization loop**

  for $n = 1 \ldots N$ **do** → **Loop over all modes**

    repeat → **Inner Optimization loop**

      Compute: $\Phi = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n}\mathbf{A}^{(m)})^T))(\odot_{m \neq n}\mathbf{A}^{(m)})$ → **MTTKRP+**

      Compute inner convergence

      Compute: $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \Phi$

    until convergence

  end for

  Compute outer convergence

until convergence

Output: $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$

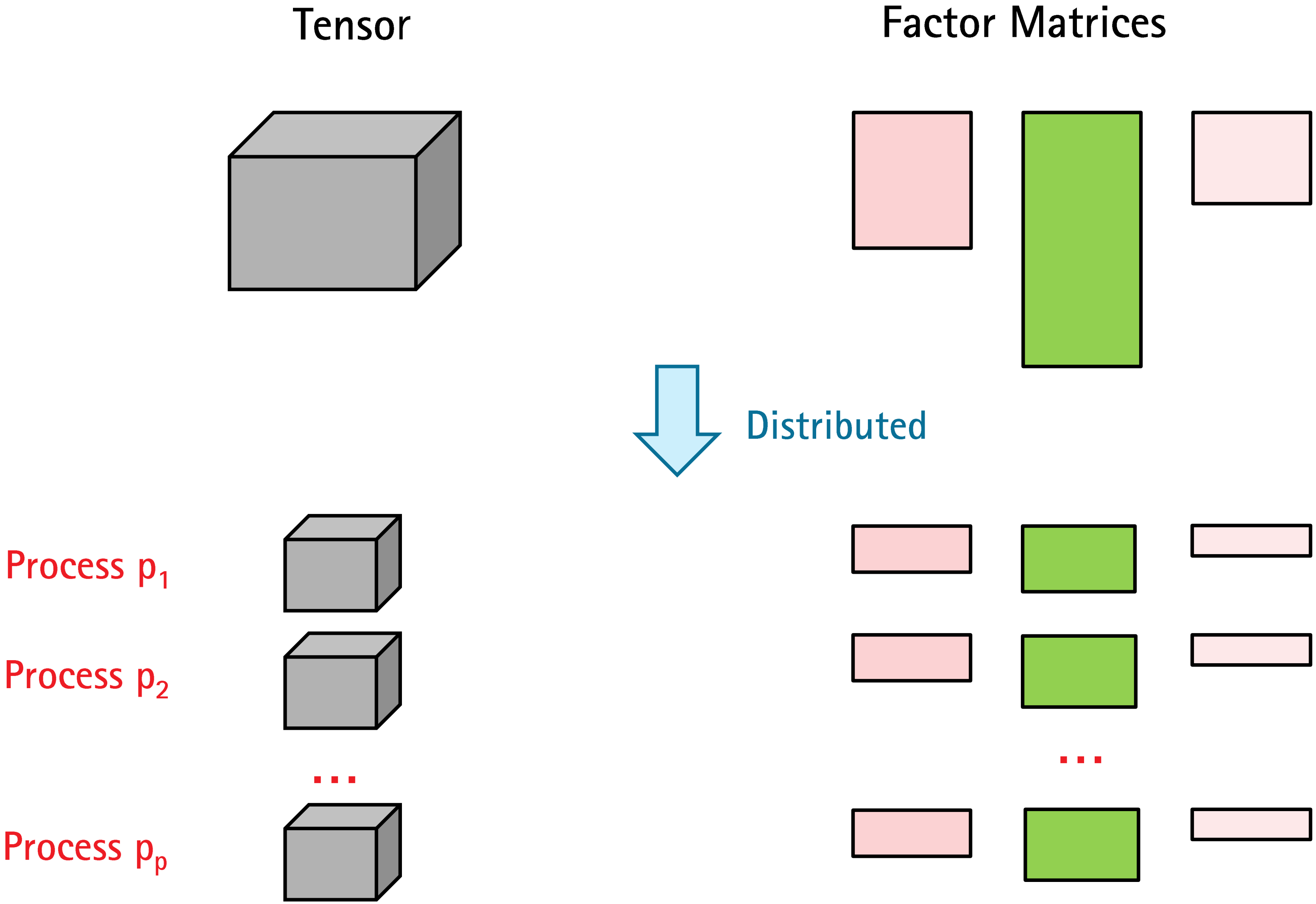
Explicitly storing the result of this computation (sparse Khatri-Rao Product) leaves a huge memory footprint $O(PR)$

P: Number of non-zeros in tensor

R: Rank of decomposition

HPEC 2017: "Memory-efficient Parallel Tensor Decompositions"
Muthu Baskaran, Tom Henretty, Benoit Pradelle, M. Harper Langston, David Bruns-Smith, James Ezick, Richard Lethin (Reservoir Labs)

Data Distribution

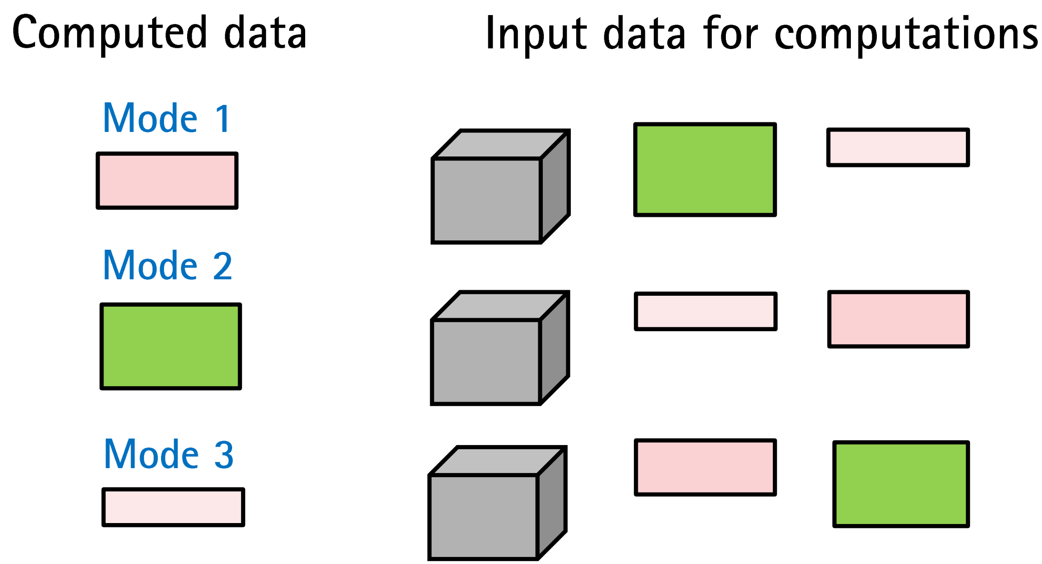


Computation Pattern

At each process

CP-APR Algorithm

Input: $\mathfrak{X}$, $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    repeat
      Compute: $\Phi = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n} \mathbf{A}^{(m)})^T))(\odot_{m \neq n} \mathbf{A}^{(m)})$
      Compute inner convergence
      Compute: $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \Phi$
    until convergence
  end for
  Compute outer convergence
until convergence
Output: $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$

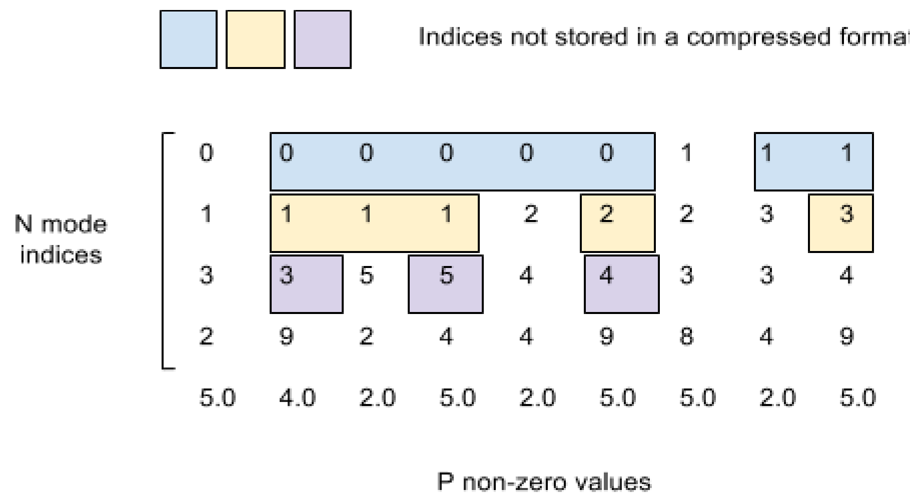


- Rows of computed factor matrix
  - May be LOCAL ("owned" by the process – has updated values)
  - May be REMOTE (partial computed values to be sent to "owner")
- Rows of input factor matrix
  - May be LOCAL ("owned" by the process – has updated values)
  - May be REMOTE (updated values gathered from "owner")

ENSIGN Sparse Tensor Data Structures

Highlights

- Hierarchical compressed sparse tensor storage
- Mode-generic and mode-specific formats



Key differentiators

- Applies to all tensor decomposition methods
- Supports a spectrum of tensors within the formats
  - From extremely sparse to partially dense to fully dense tensors
- Enables computation and memory reduction (from compression)
- Enables improved parallelism (from data structure arrangement)

Sparse Tensor Data Structure Selection

Selection of distributed sparse tensor format

- Some modes are chosen as candidates for mode-specific format
  - Choice made based on size of mode (usually "larger" modes are biased towards mode-specific format)
- If m (where $m \leq n$) modes chosen as mode-specific format candidates
  - We have m+1 distributed copies of the input tensor: m mode-specific tensors and 1 mode-generic tensor

Data Distribution Strategies

Three strategies for distributing factor matrices

- Distributed
  - Factor matrices distributed across processes
  - Each factor matrix row has a unique "owner" process
- Replicated
  - Factor matrices replicated across processes
  - Usually applied for "smaller" modes
- Partitioned
  - Factor matrices distributed across processes
  - Each factor matrix row has a unique "owner" process
  - Sparse sub-tensor contributing to the output of "owned" rows is entirely local to the process
  - Usually applied for "very large" modes for reducing communications

Computations and Communications in Distributed Decompositions

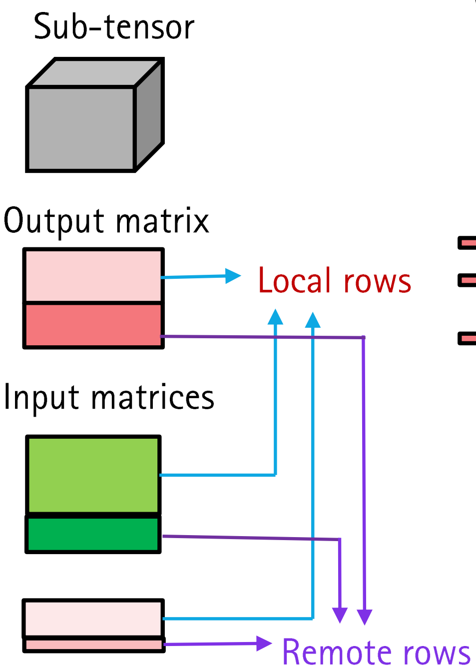
Replicated mode	Distributed mode	Partitioned mode

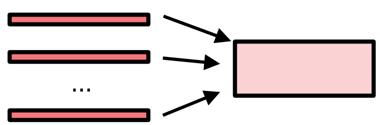
repeat
  for n= 1…N
    repeat
      Compute Ø using X, A's
      [Ø gather: Allreduce]
      Compute inner convergence
      Update A(n) with Ø
    until convergence
  Compute outer convergence
until convergence
	repeat
	  for n= 1…N
	    repeat
	      Compute Ø using X, A's
	      [Ø gather: rank-wise Reduce]
	      Compute inner convergence
	      Update A(n) with Ø
	      [A(n) gather: rank-wise Gather]
	    until convergence
	  Compute outer convergence
	until convergence
		repeat
		  for n= 1…N
		    repeat
		      Compute Ø using X, A's
		      Compute inner convergence
		      Update A(n) with Ø
		    until convergence
		    [A(n) gather: rank-wise Gather]*
		  Compute outer convergence
		until convergence
		
		* if no. of partitioned modes > 1

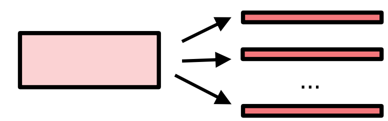
Computations and Communications in Distributed Decompositions

At each process [Distributed mode]

Compute output → Rank-wise Reduce (local rows updated with remote partial results) → More computations (updates) on local → Rank-wise Gather: Send updated local rows to all remote uses (= Receive updated remote rows) for next computation





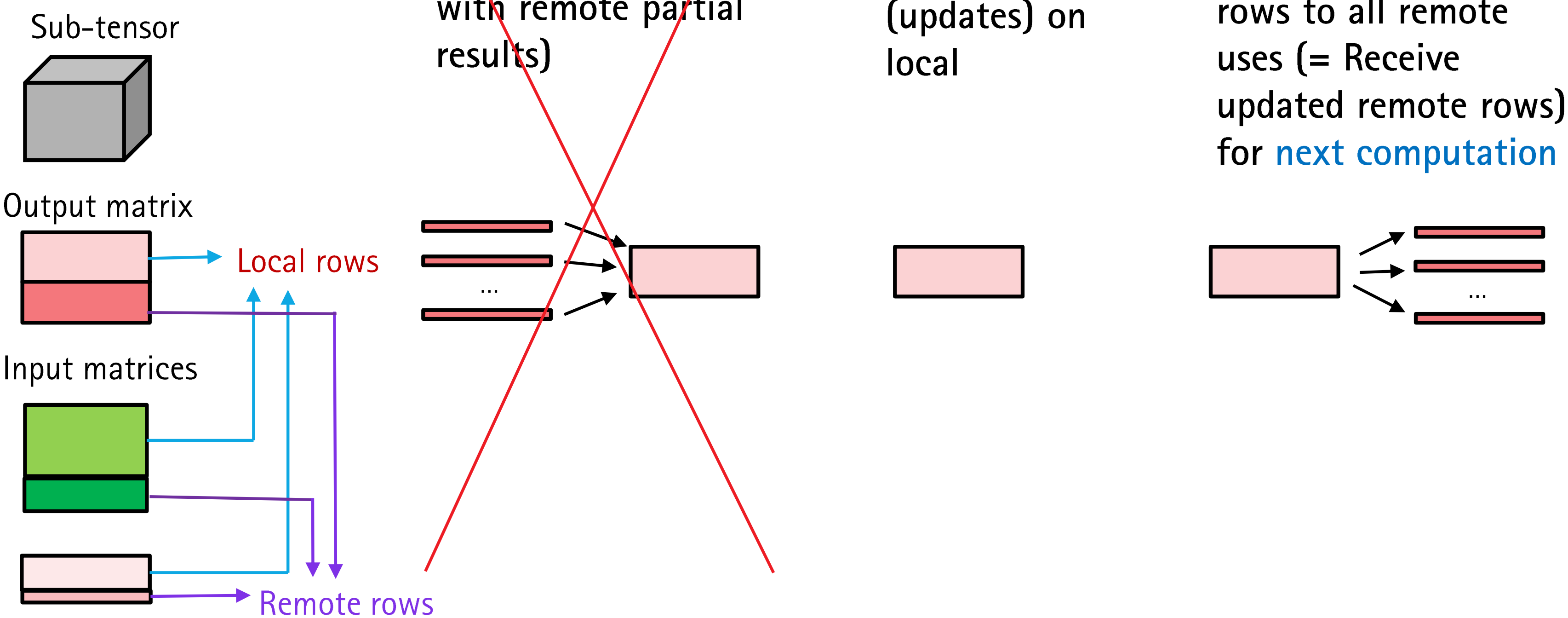


- Portions of tensor and/or factor matrices contributing to the output of “owned” or local rows present in remote processes results in communication

Computations and Communications in Distributed Decompositions

At each process [Partitioned mode]

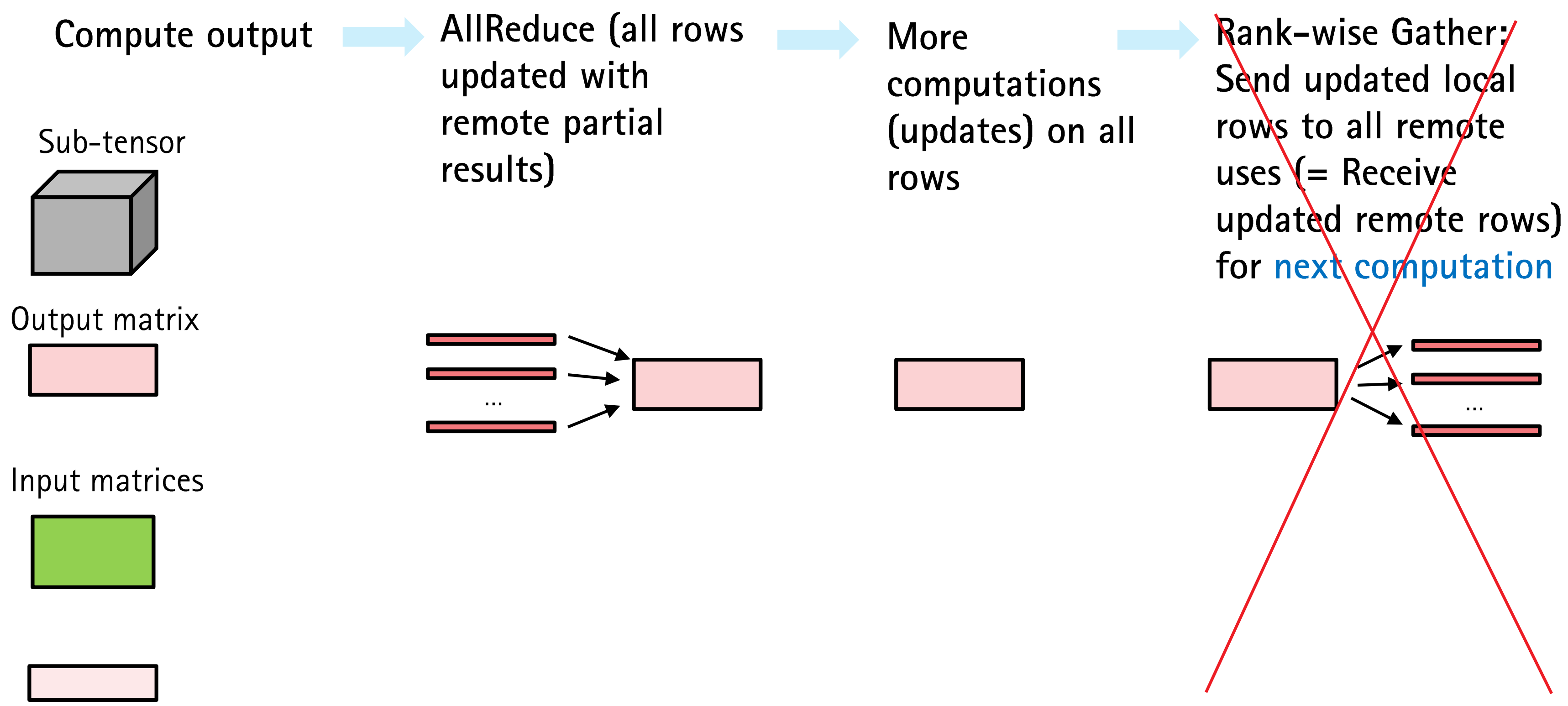
Compute output → ~~Rank-wise Reduce (local rows updated with remote partial results)~~ → More computations (updates) on local → Rank-wise Gather: Send updated local rows to all remote uses (= Receive updated remote rows) for next computation



- Sparse sub-tensor contributing to the output of "owned" rows is entirely local to the process => Implication: no remote partial results for "mode-level" iteration
- If only one partitioned mode => no communication due to that mode

Computations and Communications in Distributed Decompositions

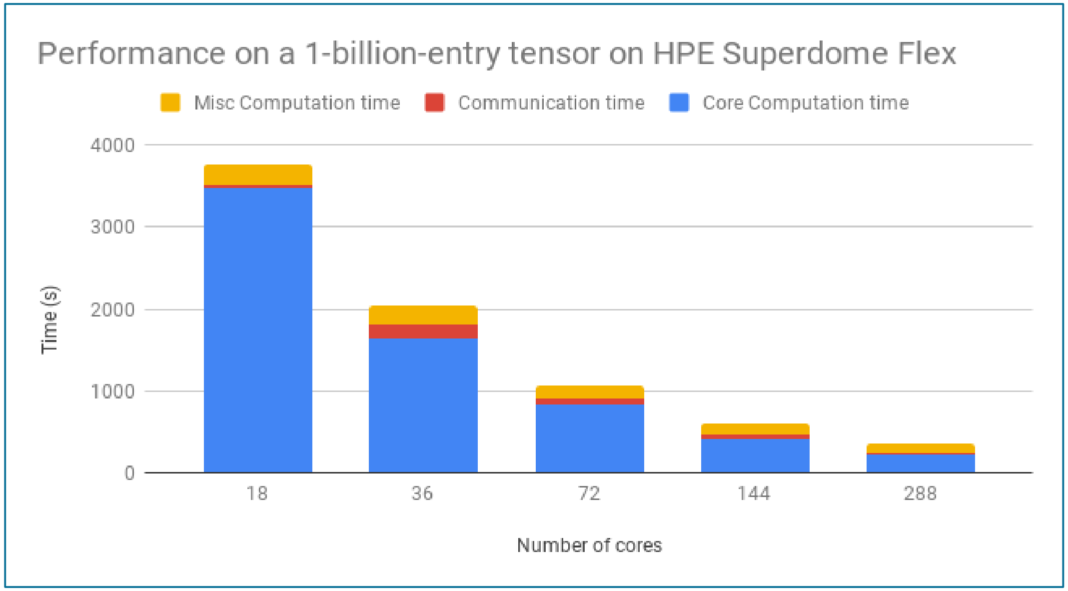
At each process [Replicated mode]



- Portions of tensor contributing to the output of "owned" or local rows present in remote processes results in communication

Scaled-up Results with HPE Superdome Flex

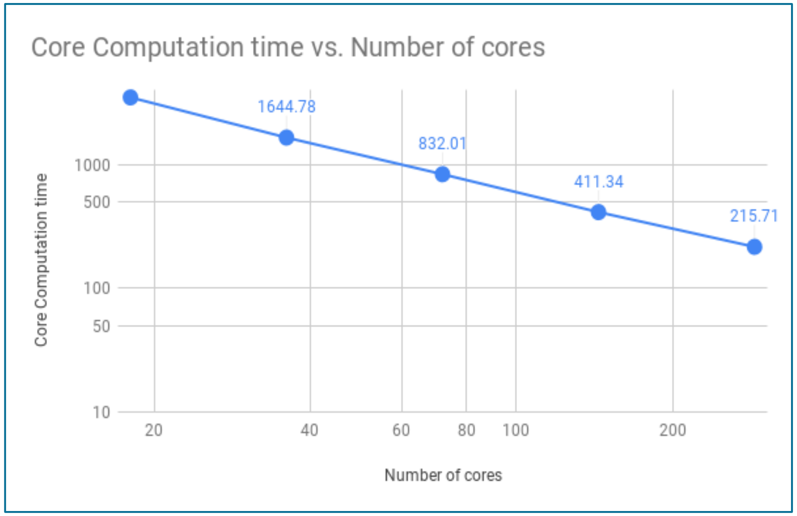
Overall scaling of performance

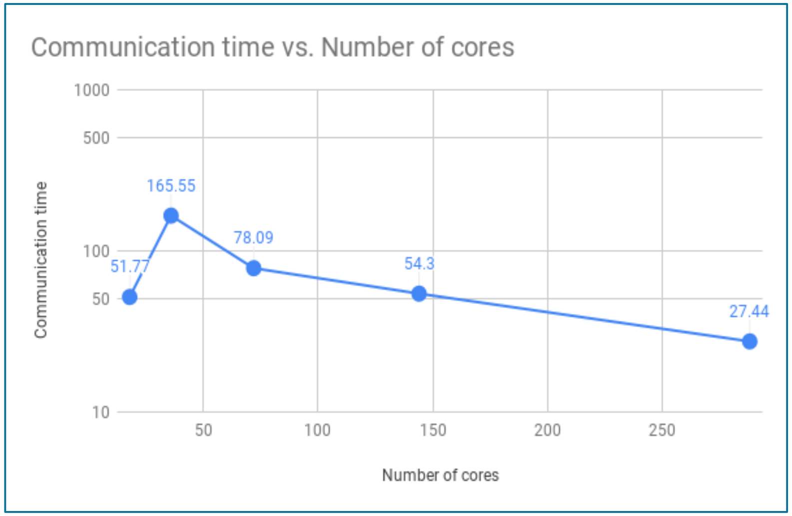


Low communication bottleneck

Near-ideal scaling of computations

Dip in communication performance initially before scaling

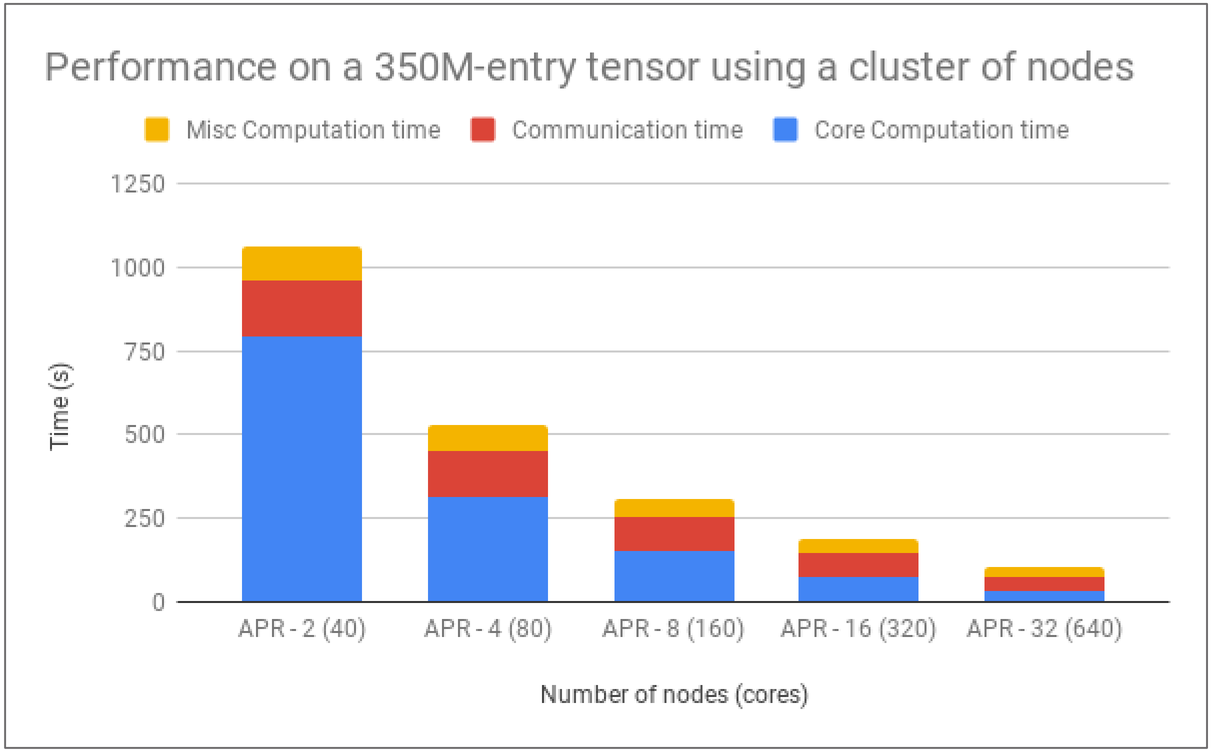




Scaled-up Results with Cluster of Nodes

Overall scaling of performance

Computation and communication scale



Summary & Forward Work

What we did

- Developed techniques for enabling tensor analysis to meet modern application needs
- Implemented efficient distributed tensor decomposition methods in ENSIGN Tensor Toolbox
- Showed scalable results on HPE Superdome Flex server and a distributed cluster of Intel nodes

What is in progress and what we plan to do

- Adapting these techniques to GPU-based implementations of tensor decompositions
- Extending these techniques to more tensor decomposition methods and more application areas

How to get ENSIGN

Contact Reservoir Labs

- Use the URL: https://www.reservoir.com/company/contact/
- or email support@reservoir.com