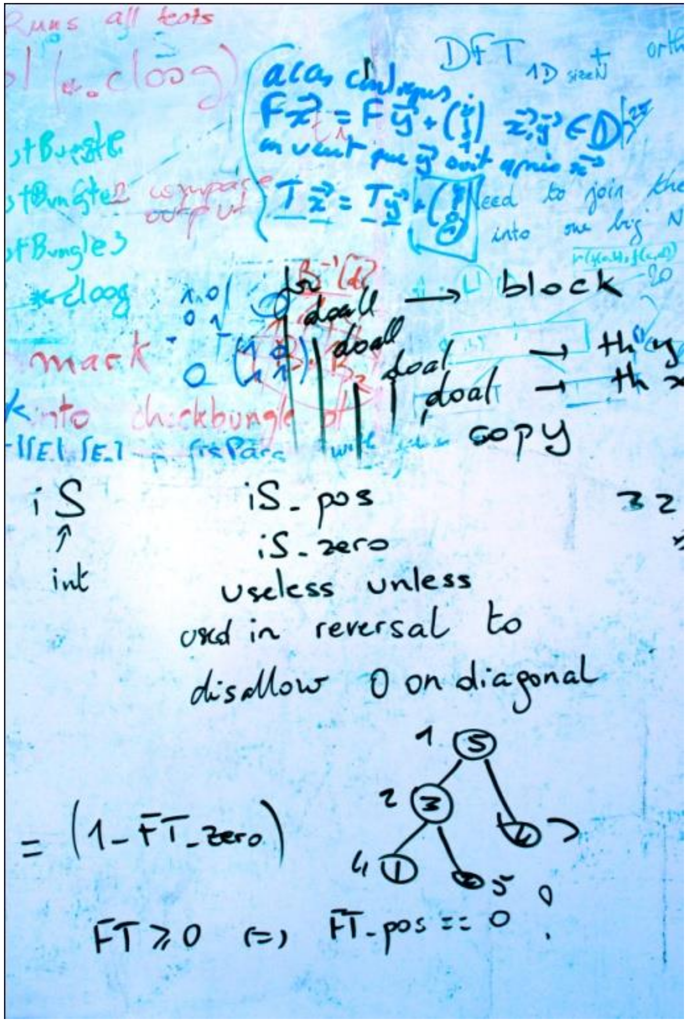
ENSIGN: High-performance Data Analytics Tool

Scaling and Deepening Tensor Decompositions and Applications using ENSIGN

Muthu Baskaran
James Ezick
Aditya Gudibanda
Thomas Henretty
M. Harper Langston
Pierre-David Letourneau
Richard Lethin
Benoit Meister
Matt Robillard

Reservoir Labs, Inc.
New York, NY

1 March 2019



www.reservoir.com

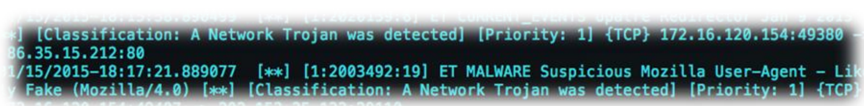
Exascale NonStationary Graph Notation (ENSIGN)

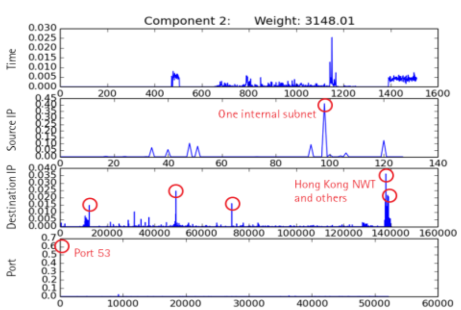
Driving Towards a Practical High-performance Data Analytics Tool

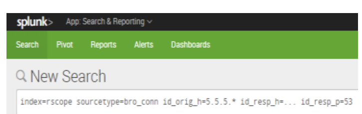
Class	Differentiating Specifics	Benefit to Analyst
Performanc e	Optimized sparse tensor data structures Mixed static/dynamic optimization Memory-efficiency optimizations Shared memory parallelism Distributed memory parallelism Communication optimizations Cloud-based optimizations	Extend the range, scale, 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
Modeling (Capability)	First-order decomposition methods Second-order decomposition methods Algorithmic improvements to 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
Usability	GUI & CLI Python bindings C bindings QGIS support Virtual machine distributions Documented, Tested, Supported	Tools to drive application workflow 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

ENSIGN Application Areas

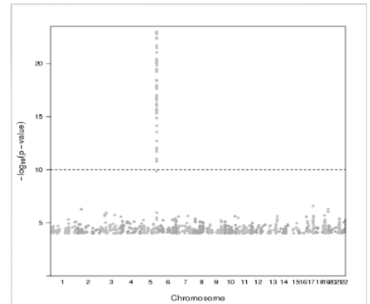
Cyber Security

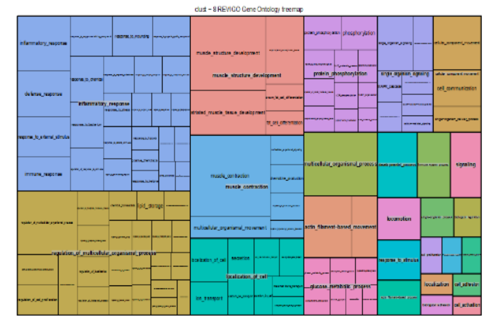




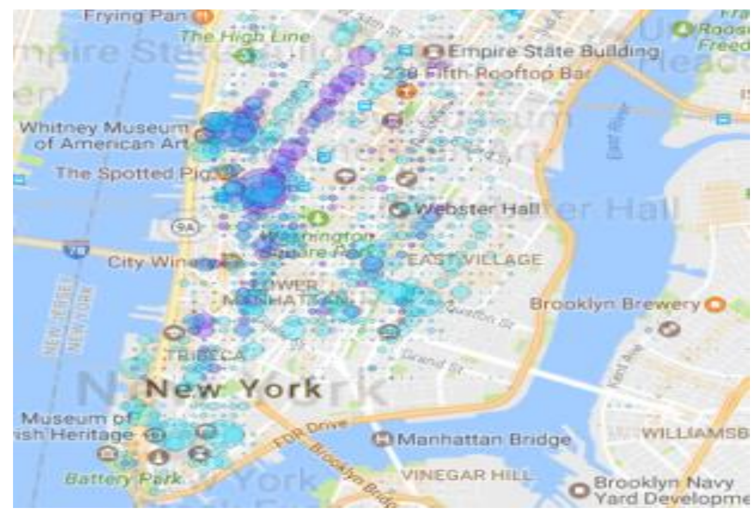


Bioinformatics





GEOINT



PERFORMANCE

ENSIGN Data Structures

Highlights

- 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)

*Baskaran, M., Meister, B., Vasilache, N., & Lethin, R. (2012). Efficient and scalable computations with sparse tensors. In *High Performance Extreme Computing (HPEC)*. (https://www.reservoir.com/publication/efficient-scalable-computations-sparse-tensors/)

Performance Optimizations

Highlights

- Distributed-memory (MPI) optimizations
- Shared-memory (OpenMP) optimizations*
- Cloud-based (Spark) optimizations**
- Memory- and operation-efficient tensor operations
  - Building blocks for newer capabilities

*Baskaran, M., Henretty, T., Pradelle, B., Langston, M. H., Bruns-Smith, D., Ezick, J., & Lethin, R. (2017). Memory-efficient parallel tensor decompositions. In IEEE High Performance Extreme Computing Conference (HPEC). [Best paper award]

**Gudibanda, A., Henretty, T., Baskaran, M., Ezick, J. and Lethin, R. (2018), All-at-once Decomposition of Coupled Billion-scale Tensors in Apache Spark. In IEEE High Performance Extreme Computing (HPEC) Conference.

CP Decomposition Methods

CP-APR Algorithm

initialize $\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \dots N$ **do**
    $\boxed{\mathbf{\Pi} = (\odot_{m \neq n} \mathbf{A}^{(m)})^T}$ Sparse Khatri-Rao Product
    repeat
      $\boxed{\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)} \mathbf{\Pi})) \mathbf{\Pi}^T}$ "MTTKRP+"
      $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \mathbf{\Phi}$
    until convergence
  end for
until convergence

CP-ALS Algorithm

initialize $\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \dots N$ **do**
    $\mathbf{V} = *_{m \neq n} \mathbf{A}^{(m)T} \mathbf{A}^{(m)}$
    $\boxed{\mathbf{U} = \mathbf{X}_{(n)} (\odot_{m \neq n} \mathbf{A}^{(m)})}$ MTTKRP
    $\mathbf{A}^{(n)} = \mathbf{U} \mathbf{V}^{\dagger}$
  end for
until convergence

CP-ALS-NN Algorithm

initialize $\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \dots N$ **do**
    $\mathbf{V} = *_{m \neq n} \mathbf{A}^{(m)T} \mathbf{A}^{(m)}$
    $\boxed{\mathbf{U} = \mathbf{X}_{(n)} (\odot_{m \neq n} \mathbf{A}^{(m)})}$ MTTKRP
    $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \frac{\mathbf{U}}{\mathbf{A}^{(n)} \mathbf{V}}$
  end for
until convergence

Need for Memory-efficiency in CP-APR

CP-APR Algorithm

```
initialize A^(1) ... A^(N)
repeat
  for n = 1 ... N do
    Π = (⊙_{m≠n} A^(m))^T
    repeat
      Φ = (X_(n) ⊘ (A^(n) Π)) Π^T
      A^(n) = A^(n) * Φ
    until convergence
  end for
until convergence
```

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

Rematerialization of Sparse Khatri-Rao Product

CP-APR Algorithm

initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    $\boxed{\mathbf{\Pi} = (\cancel{\odot_{m \neq n}} \mathbf{A}^{(m)})^T}$
    repeat
      $\boxed{\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}\mathbf{\Pi}))\mathbf{\Pi}^T}$
      $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \mathbf{\Phi}$
    until convergence
  end for
until convergence

CP-APR Modified Algorithm

initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    repeat
      $\boxed{\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n}\mathbf{A}^{(m)})))(\odot_{m \neq n}\mathbf{A}^{(m)})^T}$
      $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \mathbf{\Phi}$
    until convergence
  end for
until convergence

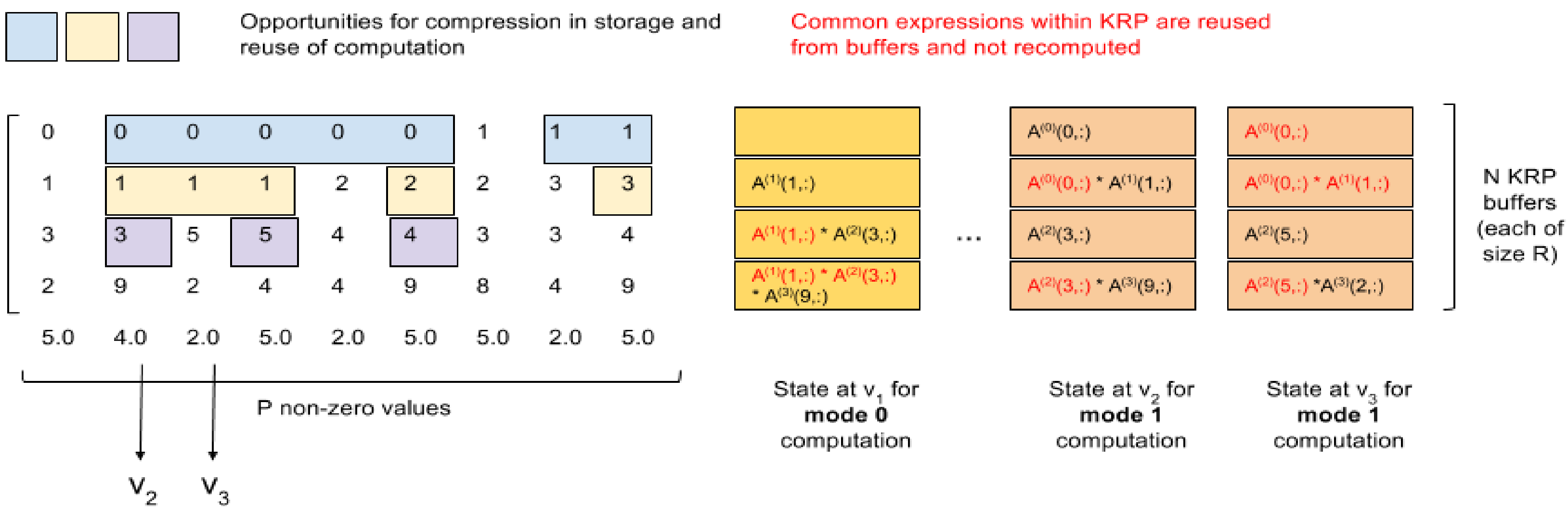
Memory footprint is reduced but number of operations is increased

Memory- and Operation-efficient CP-APR

CP-APR Modified Algorithm

initialize $\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \dots N$ **do**
    repeat
      $\boxed{\Phi = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n}\mathbf{A}^{(m)})))(\odot_{m \neq n}\mathbf{A}^{(m)})^T}$
      $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \Phi$
    until convergence
  end for
until convergence

Make this computation memory- and operation-efficient



Memory- and Operation-efficient CP -- Generalized

CP-APR Modified Algorithm

initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    repeat
      $\boldsymbol{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n} \mathbf{A}^{(m)}))) (\odot_{m \neq n} \mathbf{A}^{(m)})^T$
      $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \boldsymbol{\Phi}$
    until convergence
  end for
until convergence

= 1

MTTKRP+

CP-ALS Algorithm

initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    $\mathbf{V} = *_{m \neq n} \mathbf{A}^{(m)T} \mathbf{A}^{(m)}$
    $\mathbf{U} = \mathbf{X}_{(n)}(\odot_{m \neq n} \mathbf{A}^{(m)})$
    $\mathbf{A}^{(n)} = \mathbf{U}\mathbf{V}^{\dagger}$
  end for
until convergence

MTTKRP

CP-ALS-NN Algorithm

initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
repeat
  for $n = 1 \ldots N$ **do**
    $\mathbf{V} = *_{m \neq n} \mathbf{A}^{(m)T} \mathbf{A}^{(m)}$
    $\mathbf{U} = \mathbf{X}_{(n)}(\odot_{m \neq n} \mathbf{A}^{(m)})$
    $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \frac{\mathbf{U}}{\mathbf{A}^{(n)}\mathbf{V}}$
  end for
until convergence

MTTKRP

Increasing Thread-local Computations

CP-APR Modified Algorithm

1: initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
2: **repeat**
3:   **for** $n = 1 \ldots N$ **do**
4:     **repeat**
5:       $\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n} \mathbf{A}^{(m)})))(\odot_{m \neq n} \mathbf{A}^{(m)})^T$
6:       $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \mathbf{\Phi}$
7:     **until** convergence
8:   **end for**
9: **until** convergence

Fuse these computations

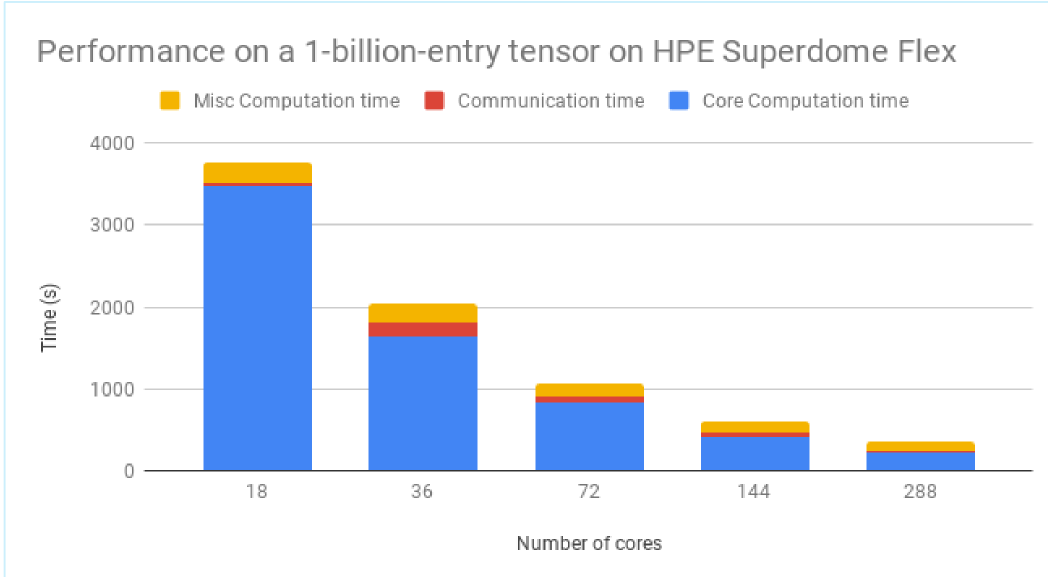
```
#pragma omp parallel
 … // Compute Φ

#pragma omp parallel
 … // Compute A^(n)

#pragma omp parallel
 … // convergence check
```

→

```
#pragma omp parallel
 … // Compute Φ
 … // Compute A^(n)
 … // convergence check
```

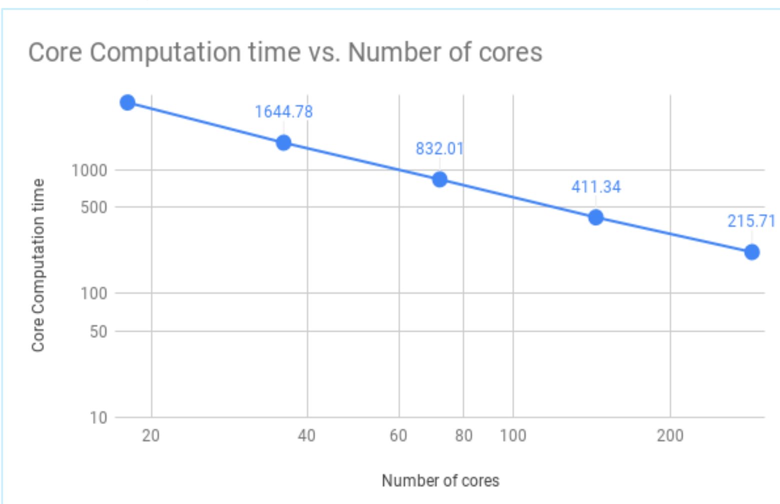
Scaled-up Results with HPE Superdome Flex

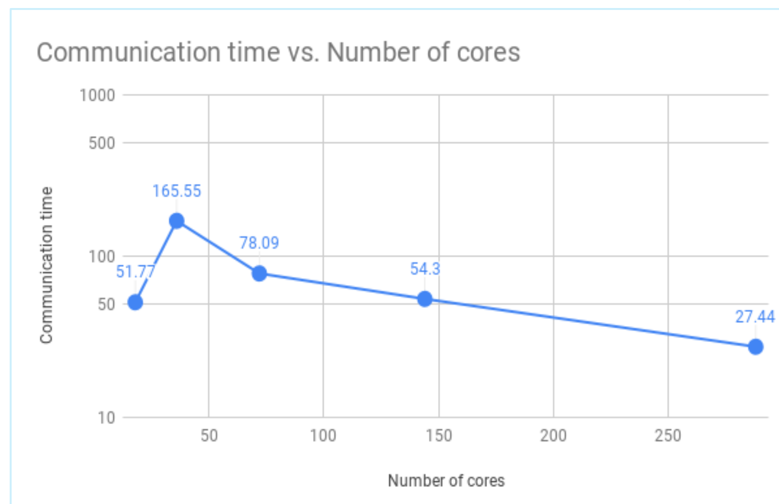
Overall scaling of performance



Near-ideal scaling of computations

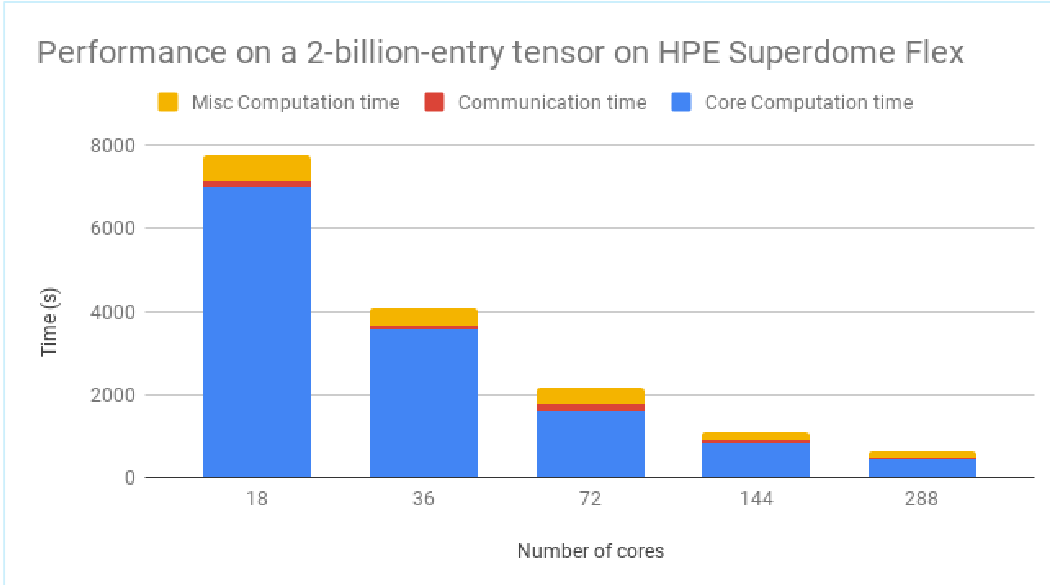
Dip in communication performance initially before scaling





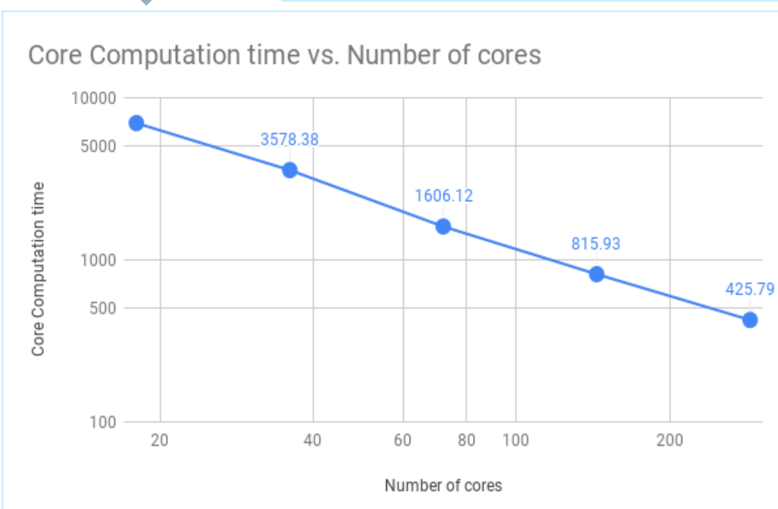
Scaled-up Results with HPE Superdome Flex

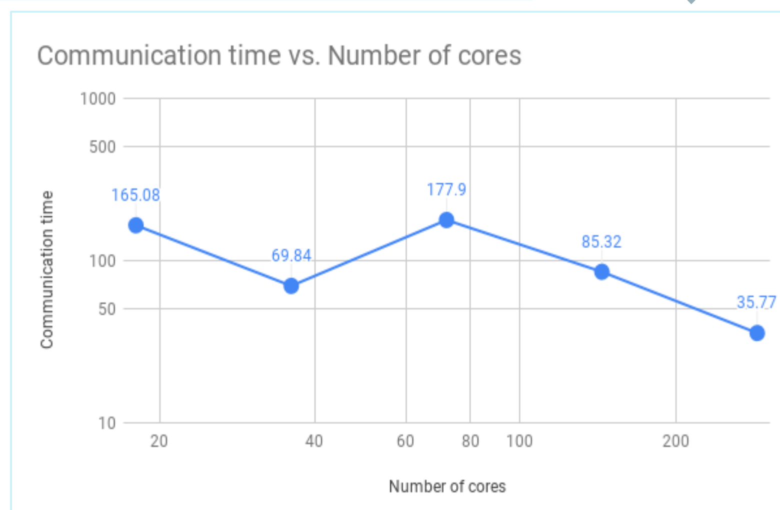
Overall scaling of performance



Near-ideal scaling of computations

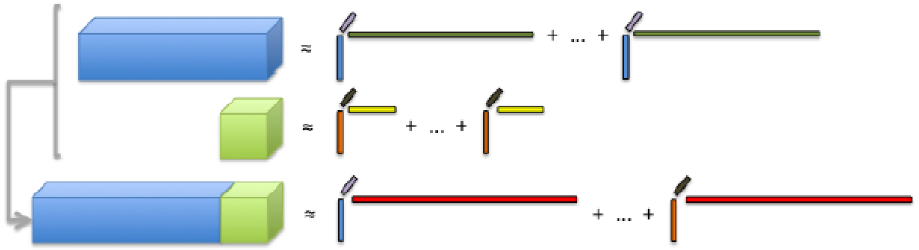
Dip in communication performance initially before scaling





MODELING (CAPABILITY)

Generalized CP Streaming Framework



Algorithm Streaming CP update

Input: $[[\mathbf{A}_{old}^{(n)}]]$, $\boldsymbol{\mathcal{X}}_{new}$, $K_{new} > 0$, $0 < \nu_{sim} \leq 1$, $\tau > 0$, $\tilde{K}$

Compute: $[[\mathbf{A}_{new}^{(n)}]]$ (rank-K_{new} decomp. of $\boldsymbol{\mathcal{X}}_{new}$)

$[[\mathbf{A}^{(n)}]],\ \tilde{\mathbf{A}}_{new}^{(N+1)} \leftarrow \mathbf{MERGE}\left([[\mathbf{A}_{old}^{(n)}]], [[\mathbf{A}_{new}^{(n)}]], \nu_{sim}\right)$

$\mathbf{A}^{(N+1)} \leftarrow \mathbf{UPDATE}\left([[\mathbf{A}^{(n)}]], \tilde{\mathbf{A}}_{new}^{(N+1)}\right)$

$\{C_1, C_2, C_3\} \leftarrow \mathbf{CLASSIFY}\left([[\mathbf{A}^{(n)}]], K, K_{old}, \tau\right)$

$[[\mathbf{A}^{(n)}]], S_{trunc} \leftarrow \mathbf{TRUNCATE}\left([[\mathbf{A}^{(n)}]], K, \tilde{K}\right)$

Output: $[[\mathbf{A}^{(n)}]]$, $\{C_1, C_2, C_3\}$, S_{trunc}

Highlights/Differentiators

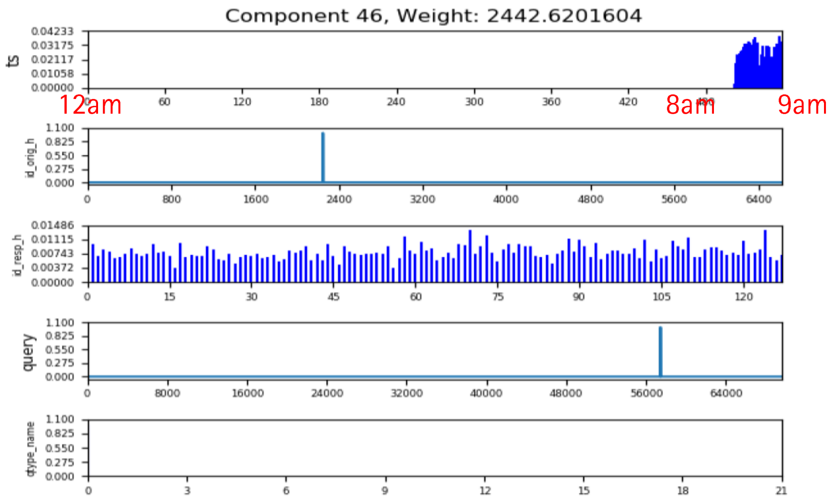
- Low-cost computations (of the order of size of streaming data streams)
- Extraction of "new information" entirely present in the new data streams
- Unified framework across different CP decompositions

Letourneau, P.D., Baskaran, M., Henretty, T., Ezick, J. and Lethin, R. (2018) Computationally Efficient CP Tensor Decomposition Update Framework for Emerging Component Discovery in Streaming Data. In High Performance Extreme Computing (HPEC) Conference. [Best Paper Award].

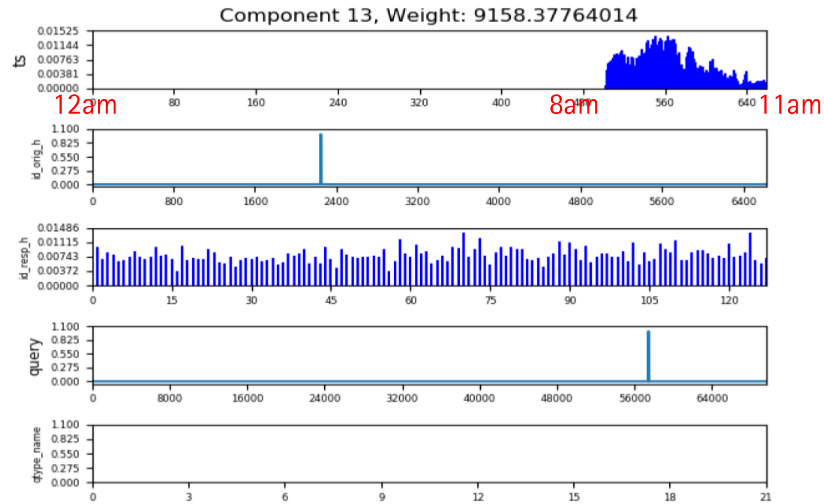
Real-world Cyber Application

… Evolution of the attack seen with streaming decompositions

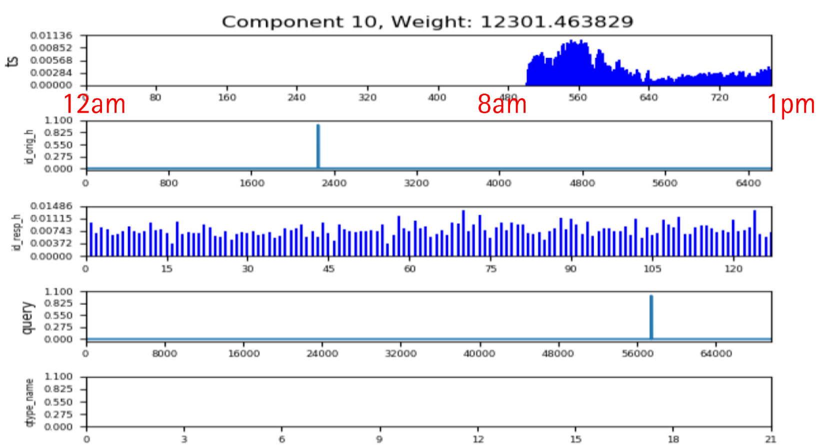
State of the activity at 9am



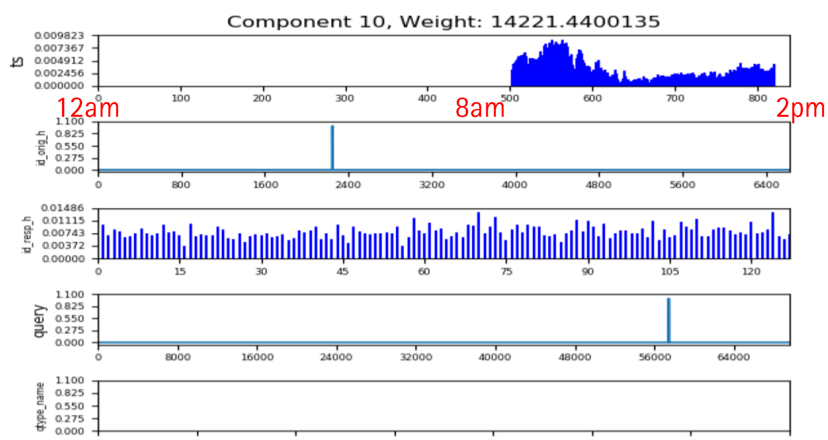
State of the activity at 11am



State of the activity at 1pm



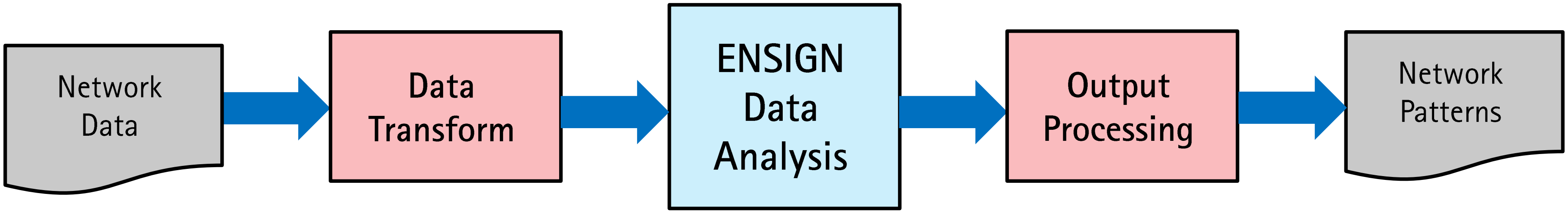
State of the activity at 2pm

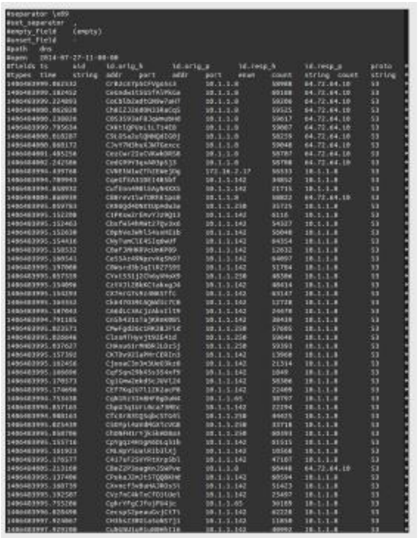


USABILITY

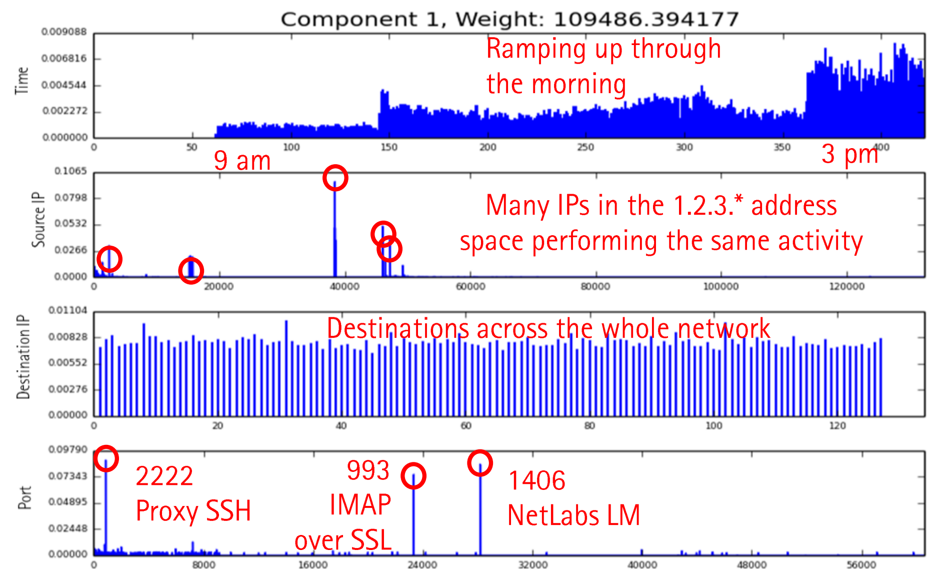
Tools for Driving Application Workflows

CLI and GUI tools to drive multi-stage application workflows





Network Flow Logs



References (Reservoir Labs, Part 1/3)

Baskaran, M. M., Henretty, T., Ezick, J., Lethin, R., & Bruns-Smith, D. (2017). Enhancing Network Visibility and Security through Tensor Analysis. (To Appear) In *Future Generation Computer Systems*.

Letourneau, P.D., Baskaran, M., Henretty, T., Ezick, J. and Lethin, R. (2018) Computationally Efficient CP Tensor Decomposition Update Framework for Emerging Component Discovery in Streaming Data. In High Performance Extreme Computing (HPEC) Conference. IEEE. [Best Paper Award]. (https://www.reservoir.com/publication/cp-tensor-decomposition-update-framework/)

Gudibanda, A., Henretty, T., Baskaran, M., Ezick, J. and Lethin, R. (2018), All-at-once Decomposition of Coupled Billion-scale Tensors in Apache Spark. In *High Performance Extreme Computing (HPEC) Conference*. IEEE. (https://www.reservoir.com/publication/coupled-tensor-decomposition-apache-spark/)

Henretty, T. S., Langston, M. H., Baskaran, M., Ezick, J., & Lethin, R. (2018). Topic modeling for analysis of big data tensor decompositions. In *Disruptive Technologies in Information Sciences*. International Society for Optics and Photonics. (https://www.reservoir.com/publication/topic-modeling-for-analysis-of-big-data-tensor-decompositions/)

Baskaran, M. M., Henretty, T., Ezick, J., Lethin, R., & Bruns-Smith, D. (2017). Enhancing Network Visibility and Security through Tensor Analysis. In *4th International Workshop on Innovating the Network for Data Intensive Science (INDIS) held in conjunction with SC17*. (https://www.reservoir.com/publication/enhancing-network-visibility-security-tensor-analysis/)

References (Reservoir Labs, Part 2/3)

Baskaran, M., Henretty, T., Pradelle, B., Langston, M. H., Bruns-Smith, D., Ezick, J., & Lethin, R. (2017). Memory-efficient parallel tensor decompositions. In *High Performance Extreme Computing Conference (HPEC).* IEEE. [Best paper award] (https://www.reservoir.com/publication/memory-efficient-parallel-tensor-decompositions/)

Henretty, T., Baskaran, M., Ezick, J., Bruns-Smith, D., & Simon, T. A. (2017). A quantitative and qualitative analysis of tensor decompositions on spatiotemporal data. In *High Performance Extreme Computing Conference (HPEC).* IEEE. (https://www.reservoir.com/publication/quantitative-qualitative-analysis-tensor-decompositions-spatiotemporal-data/)

Baskaran, M., Langston, M. H., Ramananandro, T., Bruns-Smith, D., Henretty, T., Ezick, J., & Lethin, R. (2016). Accelerated low-rank updates to tensor decompositions. In *High Performance Extreme Computing Conference (HPEC)*. IEEE. (https://www.reservoir.com/publication/accelerated-low-rank-updates-tensor-decompositions/)

Bruns-Smith, D., Baskaran, M. M., Ezick, J., Henretty, T., & Lethin, R. (2016). Cyber security through multidimensional data decompositions. In *2016 Cybersecurity Symposium (CYBERSEC)*. IEEE. (https://www.reservoir.com/publication/cyber-security-multidimensional-data-decompositions/)

Cai, J., Baskaran, M., Meister, B., & Lethin, R. (2015). Optimization of symmetric tensor computations. In *High Performance Extreme Computing Conference (HPEC).*. IEEE. (https://www.reservoir.com/publication/optimization-symmetric-tensor-computations/)

References (Reservoir Labs, Part 3/3)

Baskaran, M., Meister, B., & Lethin, R. (2014). Low-overhead load-balanced scheduling for sparse tensor computations. In *High Performance Extreme Computing Conference (HPEC)*. IEEE. (https://www.reservoir.com/publication/low-overhead-load-balanced-scheduling-sparse-tensor-computations/)

Baskaran, M. M., Meister, B., & Lethin, R. (2014). Parallelizing and optimizing sparse tensor computations. In *Proceedings of the 28th ACM international conference on Supercomputing*. ACM. (https://www.reservoir.com/publication/parallelizing-optimizing-sparse-tensor-computations/)

Baskaran, M., Meister, B., Vasilache, N., & Lethin, R. (2012). Efficient and scalable computations with sparse tensors. In *High Performance Extreme Computing (HPEC)*. IEEE. (https://www.reservoir.com/publication/efficient-scalable-computations-sparse-tensors/)

MORE SLIDES

ENSIGN on HPE Superdome Flex

ENSIGN

- Highly-optimized MPI version of tensor analysis methods

HPE Superdome Flex

- 4 chassis
- 16 sockets (4 sockets per chassis)
- 288 cores (18 cores per socket)
- 4 (chassis) * 48 (DDR4 per chassis) * 64 GB = 12 TB

Significant improvement compared to prior result on a distributed cluster

- "communication : computation time ratio" improved upto10x
  - Reduction in communication latency
  - Communication performance scaled in addition to computation performance

Python Bindings & Jupyter Notebook

jupyter ENSIGN-Jupyter Last Checkpoint: 4 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Python [conda root]

CellToolbar

In [2]:
```python
import ensign.cp_decomp as cpd
import ensign.sptensor as spt

# Parameters
rank = '100'
sptensor_file = 'tensor_data.txt'

# Load tensor & decompose
the_tensor = spt.read_sptensor_file(sptensor_file)
als_decomp = cpd.cp_als(the_tensor, rank, "als_save_dir")
apr_decomp = cpd.cp_apr(the_tensor, rank, "apr_save_dir")
pdnr_decomp = cpd.cp_apr_pdnr(the_tensor, rank, "pdnr_save_dir")
```

In [3]:
```python
als_weights = pd.Series(als_decomp.weights)
apr_weights = pd.Series(apr_decomp.weights)
pdnr_weights = pd.Series(pdnr_decomp.weights)
```

In [4]:
```python
ax = als_weights.plot(color='orange', logy=True, label='als', legend=True)
ax = apr_weights.plot(color='magenta', logy=True, label='apr', ax=ax, legend=True)
pdnr_weights.plot(color='blue', logy=True, label='pdnr', ax=ax, legend=True)
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29bd28fe90>