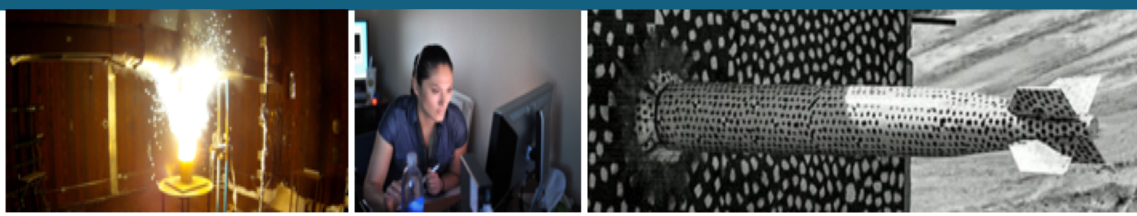
SAND2019-2243 C

Performance portable parallel sparse CP-APR tensor decompositions



SIAM CSE19, 03/01/2019

PRESENTED BY

Keita Teranishi

Christopher Forster (NVIDIA), Richard Barrett, Daniel Dunlavy, and Tamara Kolda

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

HPDA Tensor Project

Develop production quality library software to perform CP factorization for **Poisson Regression Problems** for HPC platforms

Tensor Tool Box (http://www.tensortoolbox.org)
- Matlab only!

Support several HPC platforms
- Node parallelism (Multicore, Manycore and GPUs)

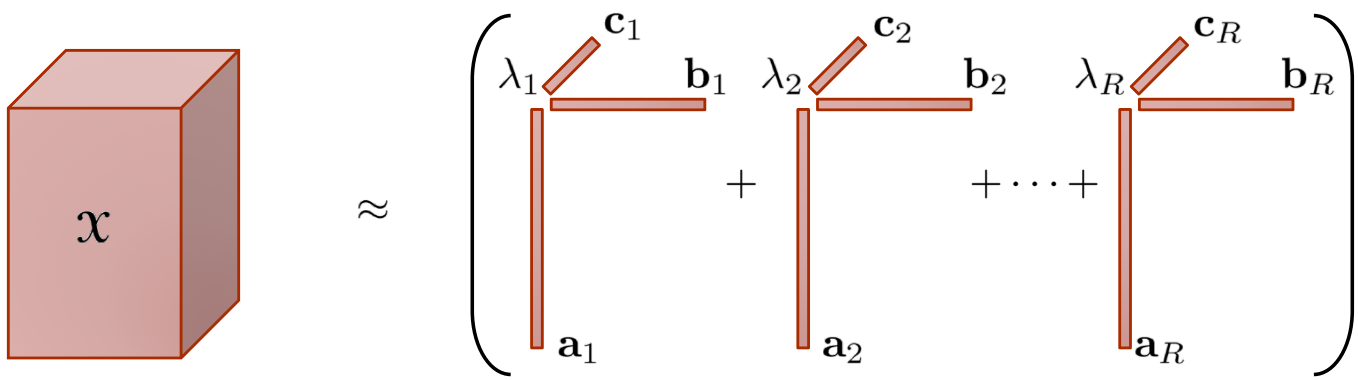
Major Questions
- Software Design
- Performance Tuning

This talk
- We are interested in two major variants
  - **Multiplicative Updates**
  - Projected Damped Newton for Row-subproblems

CP Tensor Decomposition

CANDECOMP/PARAFAC (CP) Model

Express the important feature of data using a small number of vector outer products



Model: $\mathcal{M} = \sum_r \lambda_r \, \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$

$$x_{ijk} \approx m_{ijk} = \sum_r \lambda_r \, a_{ir} \, b_{jr} \, c_{kr}$$

Key references: Hitchcock (1927), Harshman (1970), Carroll and Chang (1970)

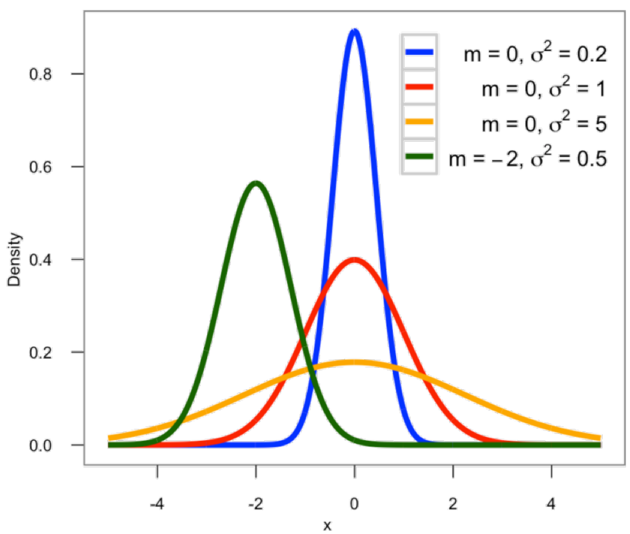
Poisson for Sparse Count Data

Gaussian (typical)

The random variable x is a continuous real-valued number.

$$x \sim N(m, \sigma^2)$$

$$P(X = x) = \frac{\exp(-\frac{(x-m)^2}{2\sigma^2})}{\sqrt{2\pi\sigma^2}}$$

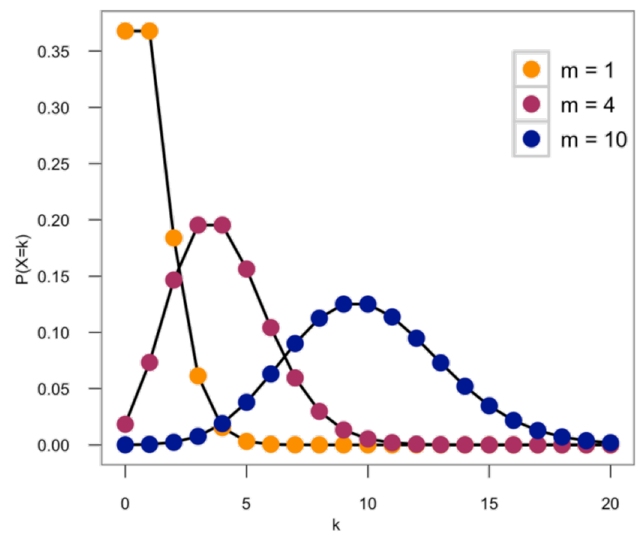


Poisson

The random variable x is a discrete nonnegative integer.

$$x \sim \text{Poisson}(m)$$

$$P(X = x) = \frac{\exp(-m)m^x}{x!}$$



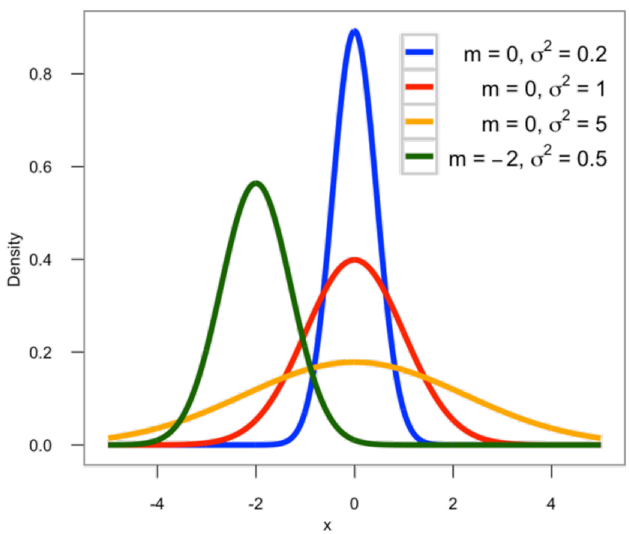
Poisson for Sparse Count Data

Gaussian (typical)

The random variable x is a continuous real-valued number.

$$x \sim N(m, \sigma^2)$$

$$P(X = x) = \frac{\exp(-\frac{(x-m)^2}{2\sigma^2})}{\sqrt{2\pi\sigma^2}}$$

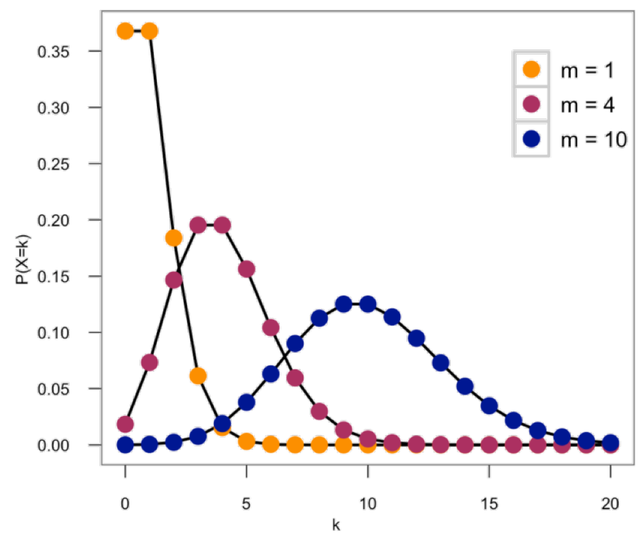


Poisson

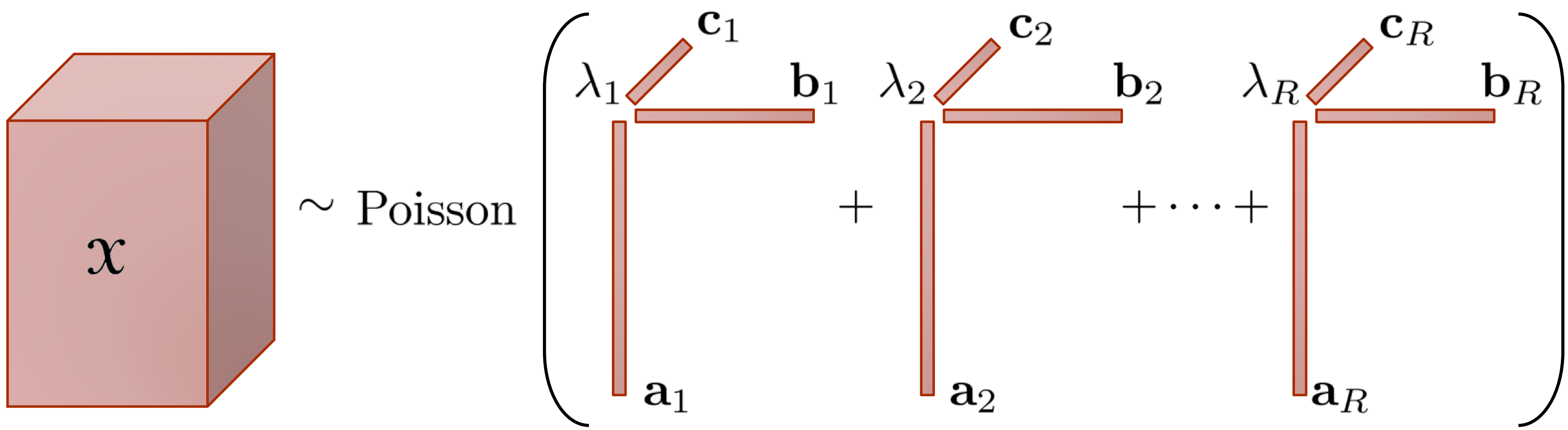
The random variable x is a discrete nonnegative integer.

$$x \sim \text{Poisson}(m)$$

$$P(X = x) = \frac{\exp(-m) m^x}{x!}$$



Sparse Poisson Tensor Factorization



<u>Model</u>: Poisson distribution (nonnegative factorization)

$$x_{ijk} \sim \text{Poisson}(m_{ijk}) \text{ where } m_{ijk} = \sum_r \lambda_r \, a_{ir} \, b_{jr} \, c_{kr}$$

- Nonconvex problem!
  - Assume R is given
- Minimization problem with constraint
  - The decomposed vectors must be non-negative
- Alternating Poisson Regression (Chi and Kolda, 2011)
  - Assume (d-1) factor matrices are known and solve for the remaining one

Alternating Poisson Regression (CP-APR)

Repeat until converged...

1. $\bar{\mathbf{A}} \leftarrow \arg\min_{\bar{\mathbf{A}} \geq 0} \sum_{ijk} m_{ijk} - x_{ijk} \log m_{ijk} \text{ s.t. } \mathcal{M} = \sum_r \bar{\mathbf{a}}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$
2. $\boldsymbol{\lambda} \leftarrow \mathbf{e}^\mathsf{T} \bar{\mathbf{A}}$; $\mathbf{A} \leftarrow \bar{\mathbf{A}} \cdot \operatorname{diag}(1/\boldsymbol{\lambda})$

Fix **B,C**; solve for **A**

3. $\bar{\mathbf{B}} \leftarrow \arg\min_{\bar{\mathbf{B}} \geq 0} \sum_{ijk} m_{ijk} - x_{ijk} \log m_{ijk} \text{ s.t. } \mathcal{M} = \sum_r \mathbf{a}_r \circ \bar{\mathbf{b}}_r \circ \mathbf{c}_r$
4. $\boldsymbol{\lambda} \leftarrow \mathbf{e}^\mathsf{T} \bar{\mathbf{B}}$; $\mathbf{B} \leftarrow \bar{\mathbf{B}} \cdot \operatorname{diag}(1/\boldsymbol{\lambda})$

Fix **A,C**; solve for **B**

5. $\bar{\mathbf{C}} \leftarrow \arg\min_{\bar{\mathbf{C}} \geq 0} \sum_{ijk} m_{ijk} - x_{ijk} \log m_{ijk} \text{ s.t. } \mathcal{M} = \sum_r \mathbf{a}_r \circ \mathbf{b}_r \circ \bar{\mathbf{c}}_r$
6. $\boldsymbol{\lambda} \leftarrow \mathbf{e}^\mathsf{T} \bar{\mathbf{C}}$; $\mathbf{C} \leftarrow \bar{\mathbf{C}} \cdot \operatorname{diag}(1/\boldsymbol{\lambda})$

Fix **A,B**; solve for **C**

Convergence Theory

Theorem: The CP-APR algorithm will **converge to a constrained stationary point** if the subproblems are strictly convex and solved exactly at each iteration. (Chi and Kolda, 2011)

Accuracy is High For Very Sparse Data

Data: 1000 x 800 x 600 Tensor with R=10 Components
CP-APR: Max Iterations = 200, Max Inner Iterations = 30 (10 per mode), Tol = 1e-4 (KKT)
CP-ALS: Max Iterations = 200, Tol = 1e-8 (change in fit)

Nonzeros	Poisson Regression FMS	Gaussian Regression FMS
480,000 (.100%)	0.99	0.57
240,000 (.050%)	0.81	0.49
48,000 (.010%)	0.77	0.47
24,000 (.005%)	0.74	0.46

CP-APR

Algorithm 1: CPAPR, Alternating Block Framework

CPAPR $(\mathcal{X}, \mathcal{M})$;

Input : Sparse N-mode Tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \ldots I_N$ and the number of components R

Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \ldots A^{(N)}]$

Initialize

repeat

  for $n = 1, \ldots, N$ **do**

    Let $\Pi^{(n)} = (A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \ldots A^{(1)})^T$

    Compute $\bar{A}^{(n)}$ that minimize $f(\bar{A}^{(n)})$ s.t. $\bar{A}^{(n)} \geq 0$

    $A^{(n)} \leftarrow \bar{A}^{(n)}$

  end

until *all mode subproblems converged*;

Minimization problem is expressed as:

$$\min_{\bar{A}^{(n)}>0} f(\bar{A}^{(n)}) = e^T[\bar{A}^{(n)}\Pi^{(n)} - X_{(n)} * \log(\bar{A}^{(n)}\Pi^{(n)})]e$$

CP-APR

$\odot$ is called Khatori-Rao product
(Column wise Kronecker product)

$$C = [C_1|C_2|C_3]$$
$$D = [D_1|D_2|D_3]$$
$$C \odot D = [C_1 \otimes D_1 |\ C_2 \otimes D_2 | C_3 \otimes D_3]$$

Algorithm 1: CPAPR

<u>CPAPR</u> $(\mathcal{X}, \mathcal{M})$;

Input : Sparse N-mode Tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \dots I_N$ and the number of components R

Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \dots A^{(N)}]$

Initialize

repeat

  for $n = 1, \dots, N$ **do**

    Let $\Pi^{(n)} = (A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \dots A^{(1)})^T$

    Compute $\bar{A}^{(n)}$ that minimize $f(\bar{A}^{(n)})$ s.t. $\bar{A}^{(n)} \geq 0$

    $A^{(n)} \leftarrow \bar{A}^{(n)}$

  end

until *all mode subproblems converged*;

Minimization problem is expressed as:

$$\min_{\bar{A}^{(n)} > 0} f(\bar{A}^{(n)}) = e^T [\bar{A}^{(n)} \Pi^{(n)} - X_{(n)} * \log(\bar{A}^{(n)} \Pi^{(n)})] e$$

CP-APR

$\odot$ is called Khatori-Rao product
(Column wise Kronecker product)

$$C = [C_1 | C_2 | C_3]$$
$$D = [D_1 | D_2 | D_3]$$
$$C \odot D = [C_1 \otimes D_1 | \; C_2 \otimes D_2 | C_3 \otimes D_3]$$

Algorithm 1: CPAPR

<u>CPAPR</u> $(\mathcal{X}, \mathcal{M})$;

Input : Spars

number of components R

Π is expressed in COO sparse format (indices and values).

Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \ldots A^{(N)}]$

Initialize

repeat

  for $n = 1, \ldots, N$ **do**

    Let $\Pi^{(n)} = (A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \ldots A^{(1)})^T$

    Compute $\bar{A}^{(n)}$ that minimize $f(\bar{A}^{(n)})$ s.t. $\bar{A}^{(n)} \geq 0$

    $A^{(n)} \leftarrow \bar{A}^{(n)}$

  end

until *all mode subproblems converged*;

Minimization problem is expressed as:

$$\min_{\bar{A}^{(n)} > 0} f(\bar{A}^{(n)}) = e^T [\bar{A}^{(n)} \Pi^{(n)} - X_{(n)} * \log(\bar{A}^{(n)} \Pi^{(n)})] e$$

CP-APR

$\odot$ is called Khatori-Rao product
(Column wise Kronecker product)

$$C = [C_1|C_2|C_3]$$
$$D = [D_1|D_2|D_3]$$
$$C \odot D = [C_1 \otimes D_1 |\ C_2 \otimes D_2 | C_3 \otimes D_3]$$

Algorithm 1: CPAPR

CPAPR $(\mathcal{X}, \mathcal{M})$;

Input : Spars

number of components R

Output

Initial

repea

fo

en

until

Π is expressed in COO sparse format (indices and values).

- 2 major approaches
  - **Multiplicative Updates** like Lee & Seung (2000) for matrices, but extended by **E. C. Chi and T. G. Kolda.** ***On Tensors, Sparsity, and Nonnegative Factorizations*, SIAM Journal on Matrix Analysis and Applications 33(4):1272-1299, December 2012.**
  - **Newton and Quasi-Newton method for Row-subpblems** by **S. Hansen, T. Plantenga and T. G. Kolda.** ***Newton-Based Optimization for Kullback-Leibler Nonnegative Tensor Factorizations*, to appear in Optimization Methods and Software, 2015.**

Minimiza

Key Elements of MU and PDNR methods

Multiplicative Update (MU)

Key computations

- Khatri-Rao Product $\Pi^{(n)}$
- Multiplicative Update Modifier (10+ iterations)

Key features

- Factor matrix is updated all at once
- Exploits the convexity of row subproblems for global convergence

Projected Damped Newton for Row-subproblems (PDNR)

Key computations

- Khatri-Rao Product $\Pi^{(n)}$
- Constrained Non-linear Newton-based optimization for each row

Key features

- Factor matrix can be updated by rows
- Exploits the convexity of row-subproblems

CP-APR-MU

Algorithm 1: CP-APR-MU, Multiplicative Update

CP-APR-MU $(\mathcal{X}, \mathcal{M})$;

Input : Sparse N-mode Tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \ldots I_N$ and the number of components R

Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \ldots A^{(N)}]$

Initialize

repeat

  for $n = 1, \ldots, N$ **do**

    $B \leftarrow (A^{(n)} + S)\Lambda$ (S is used to remove inadmissible zeros)

    Let $\Pi^{(n)} = (A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \ldots A^{(1)})^T$

    for $i = 1, \ldots, 10$ **do**

      $\Phi^{(n)} \leftarrow (X_{(n)} \oslash \max(B\Pi^{(n)}, \epsilon))(\Pi^{(n)})^T$

      $B \leftarrow B * \Phi^{(n)}$

    end

    $\lambda = e^T B$

    $A^{(n)} \leftarrow B\Lambda^{-1}$, where $\Lambda = \text{diag}(\lambda)$

  end

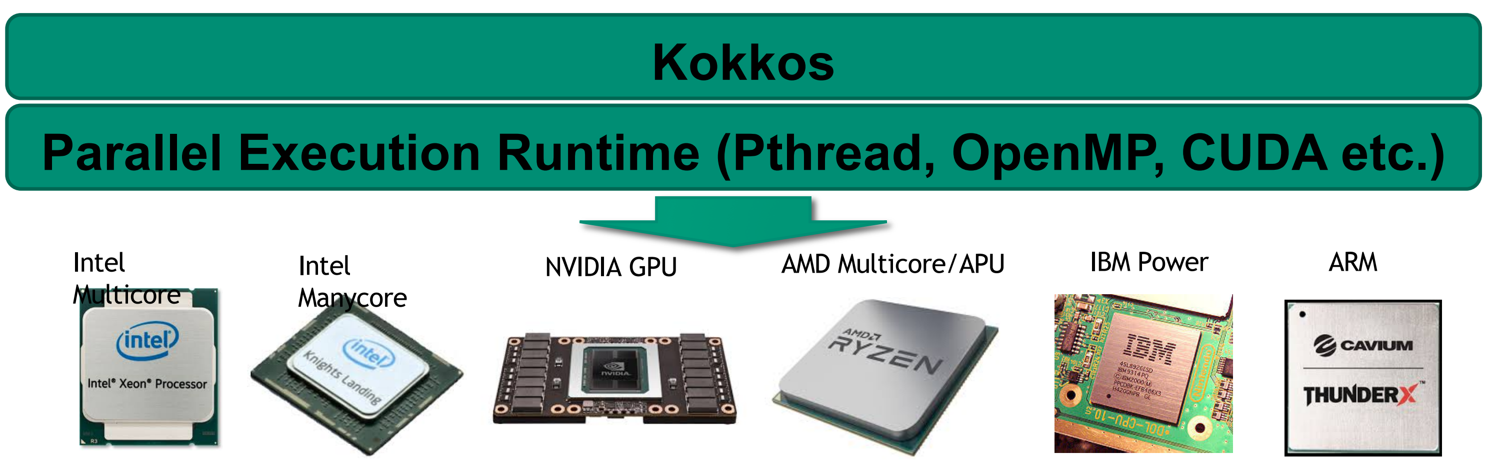
until *all mode subproblems converged*;

Key Computations

Focus on on-node parallelism for multiple architectures

- Multiple choices for programming
  - OpenMP, OpenACC, CUDA, Pthread …
  - Manage different low-level hardware features (cache, device memory, NUMA…)

- Our Solution: **Use Kokkos for productivity and performance portability**
  - Abstraction of parallel loops
  - Abstraction Data layout (row-major, column major, programmable memory)
  - Same code to support multiple architectures



What is Kokkos?

Templated C++ Library by Sandia National Labs (Edwards, et al)

- Serve as substrate layer of sparse matrix and vector kernels
- Support any machine precisions
  - Float, Double, etc

Kokkos::View() accommodates performance-aware multidimensional array data objects

- Light-weight C++ class to accommodate abstractions for platform specific features (host, device, GPU's shared memory, data access pattern, etc.)

Parallelizing loops using C++ language standard

- **Lambda**
- Functors

Extensive support of atomics

Parallel Programing with Kokkos

Serial

```
for (size_t i = 0; i < N; ++i)
{
  /* loop body */
}
```

OpenMP

```
#pragma omp parallel for
for (size_t i = 0; i < N; ++i)
{
  /* loop body */
}
```

Kokkos

```
parallel_for (( N, [=], (const size_t i)
{
  /* loop body */
});
```

Kokkos information courtesy of Carter Edwards

Provide parallel loop operations using C++ language features

Conceptually, the usage is no more difficult than OpenMP. The annotations just go in different places.

Support for task parallel computing is ongoing (Task Parallel Kokkos and UINTHA)

Parallel CP-APR-MU

Algorithm 1: CP-APR-MU in source

CP-APR-MU X, M, R;
Input : Sparse N-mode Tensor X of size $I_1 \times I_2 \times \dots I_N$ and the number of components R
Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \dots A^{(N)}]$
initializeBuffer(X, R)
$\mathcal{E} \leftarrow$ computeIndexMap(X)
repeat
  for $n = 1, \dots, N$ **do**
    $M \leftarrow$ offset(M, n) (Remove inadmissible zeros)
    $M \leftarrow$ distribute(M, n) (Scale the elements of A^n by λ)
    $\Pi^{(n)} \leftarrow$ computePi$(M, \mathcal{E}^{(n)})$
    for $i = 1, \dots, 10$ **do**
      $\Phi_i^{(n)} \leftarrow$ computePhi$(A_i^{(n)}, \Pi^{(n)}, \mathcal{E}^{(n)})$
      $A_{i+1}^{(n)} \leftarrow A_i^{(n)} \Phi_i^{(n)}$
    end
    $M \leftarrow$ normalize(M, A, n)
  end
until *all mode subproblems converged*;

Data Parallel

Notes on Data Structure and implementation

Use Kokkos::View for all data strcutures

Sparse Tensor

- Similar to the Coordinate (COO) Format in Sparse Matrix representation

Atomics

- Expensive for CPUs and Manycore
- Efficient for the latest GPUs

INDICES (_indices)													
	Mode-1	1	1	1	1	2	2	4	1	3	3	5	4
	Mode-2	3	4	1	3	5	2	4	2	3	5	1	4
	Mode-3	5	1	2	2	2	3	3	1	5	4	1	4
	Nonzero Entries (_data)	0.2	0.5	0.1	0.7	0.9	2.1	0.4	0.5	0.1	0.4	0.6	0.1

Nested Parallelism

- Kokkos provides abstraction for multiple platforms (team, thread, vector) to map parallel program execution to:
  - SM
  - Warp

Notes on Implementation of CP-APR-MU

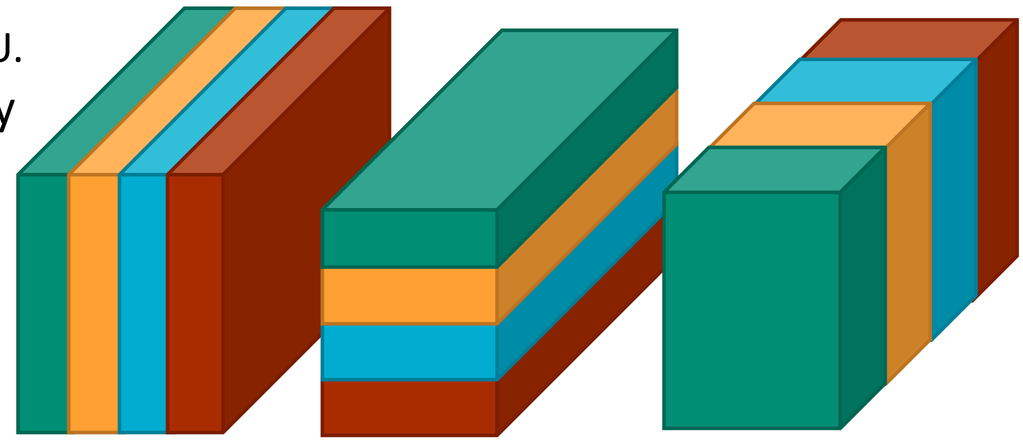
Modifier Computation is the major part of CP-APR-MU.

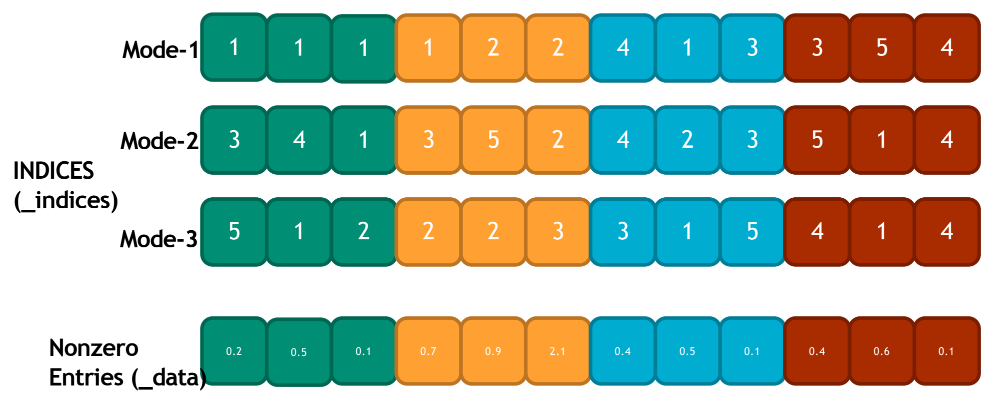
- **Two ways** to parallelize, which affects the way to access the output factor matrices

  1. **Partition with respect to the mode**
     - **No atomics** to access the output vectors by partition
     - **Extra indexing** is required to access nonzero entries by partition **(reordering)**

  2. **Partition COO sparse tensor storage format**
     - **No extra indexing** is required
     - Need efficient hardware supported **atomics**
       - The output vector elements are updated by multiple threads concurrently
     - Large outermost loop irrespective of the mode sizes

- Recent work by Smith and Karypis, and Li and Vuduc suggest more efficient data format than COO





Performance Test

Strong Scalability

- Problem size is fixed

Random Tensor

- 3K x 4K x 5K, 10M nonzero entries
- **100 outer iterations**

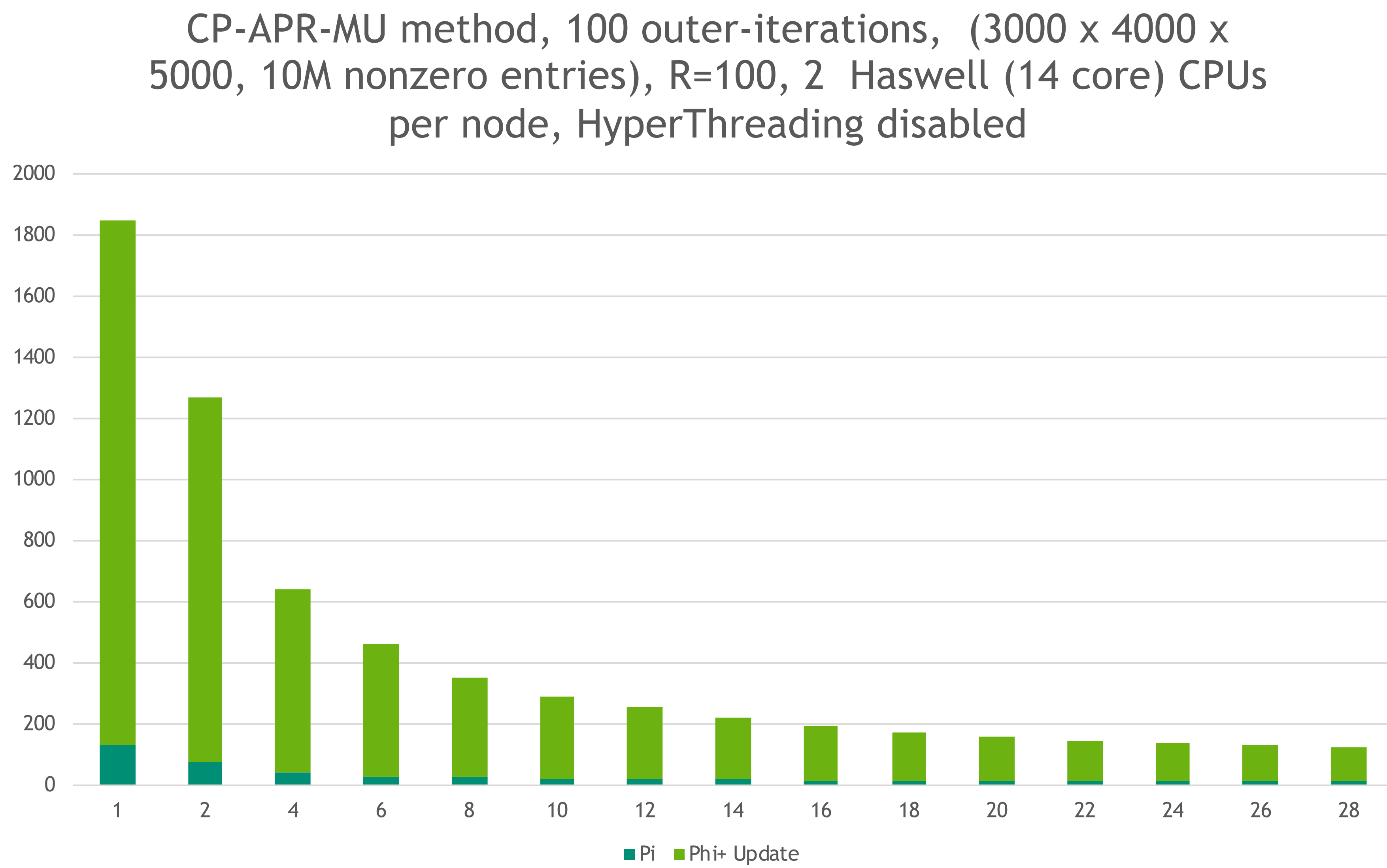
Realistic Problems

- Count Data (Non-negative)
- Available at http://frostt.io/
- **10 outer iterations**

Data	Dimensions	Nonzeros	Rank (*)
LBNL	2K x 4K x 2K x 4K x **866K**	1.7M	10
NELL-2	12K x 9K x 29K	77M	10
NELL-1	3M x 2M x 25M	144M	10
Delicious	500K x **17M** x 3M x **1K**	140M	10

(*) if not indicated.

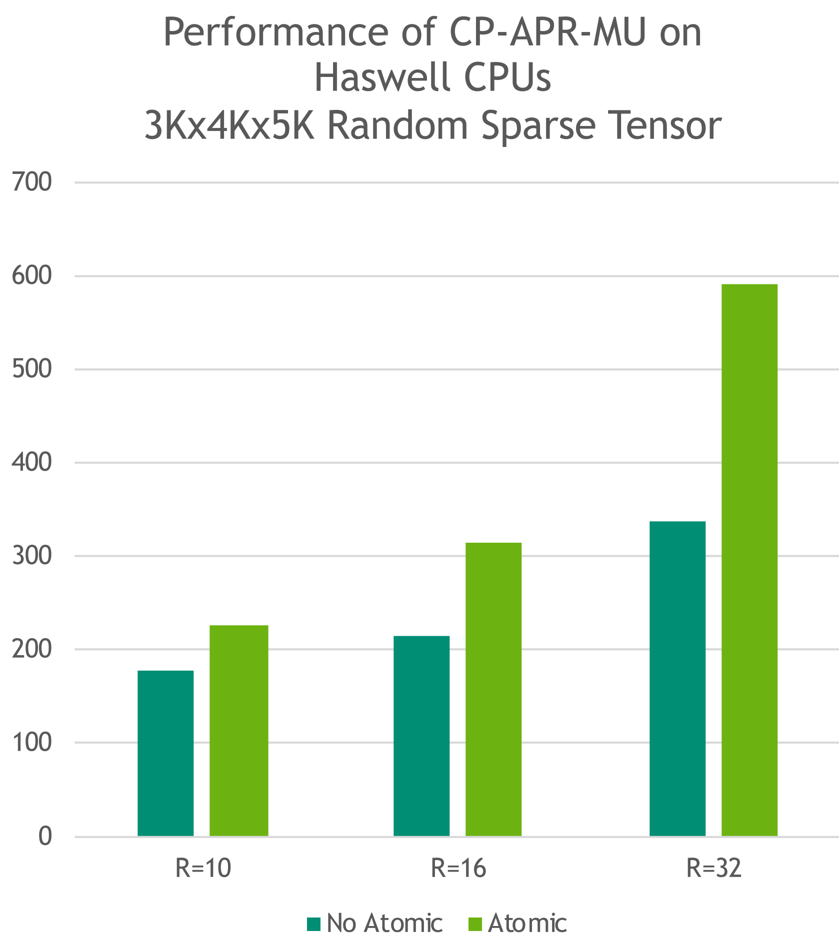
Scalability of CPAPR-MU on CPU (Random)

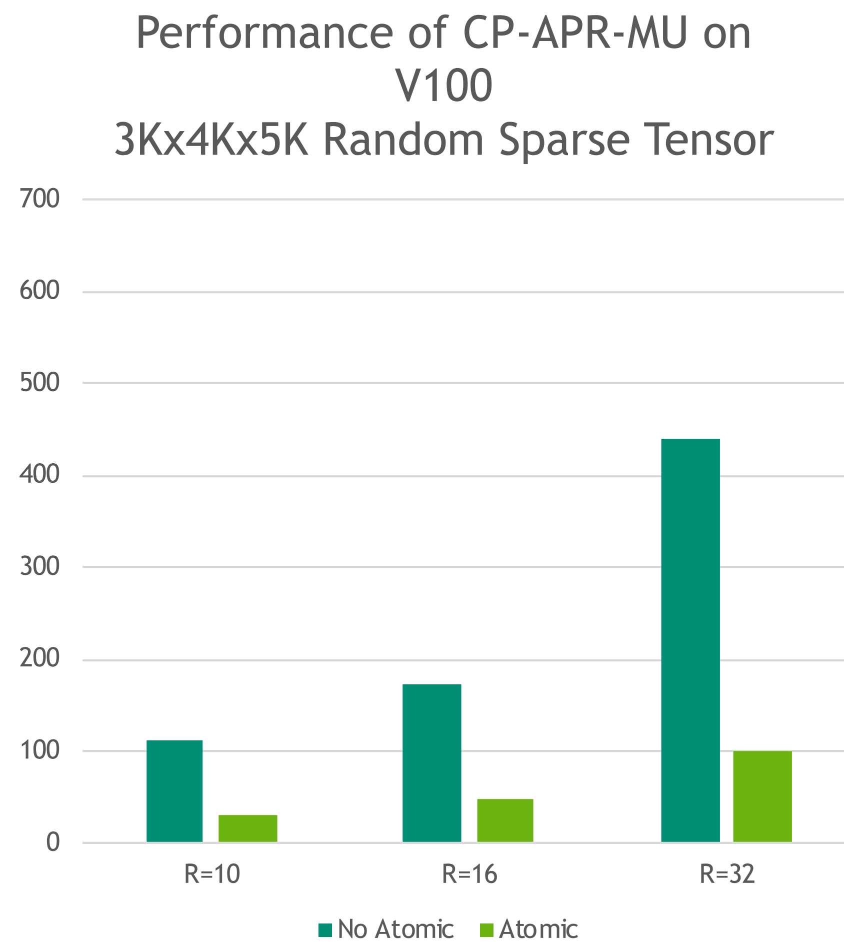


CP-APR-MU: Performance on GPUs (10 inner, 10 outer iterations, 10 components)

Data	Haswell CPU 1-core		2 Haswell CPUs 14-cores		2 Haswell CPUs 28-cores		Intel KNL (Cache Mode) 68-core CPU		NVIDIA P100 GPU		NVIDIA V100 GPU	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
Random	185	1	22	8.4	13	14.11	8.4	22.01	4.47	41.31	3.01	61.53
LBNL	39	1	19	2.05	13	3.0	33	1.18	2.99	13.04	2.09	18.66
NELL-2	1157	1	137	8.44	87	13.29	100	11.02	47.17	24.52	28.80	40.17
NELL-1	3365	1	397	16.62	258	20.9	257	10.86	OOM		OOM	
Delicious	4170	1	2183	1.91	1872	2.23	3463	1.41	OOM		OOM	

Performance Comparison: Atomic vs Non-Atomic

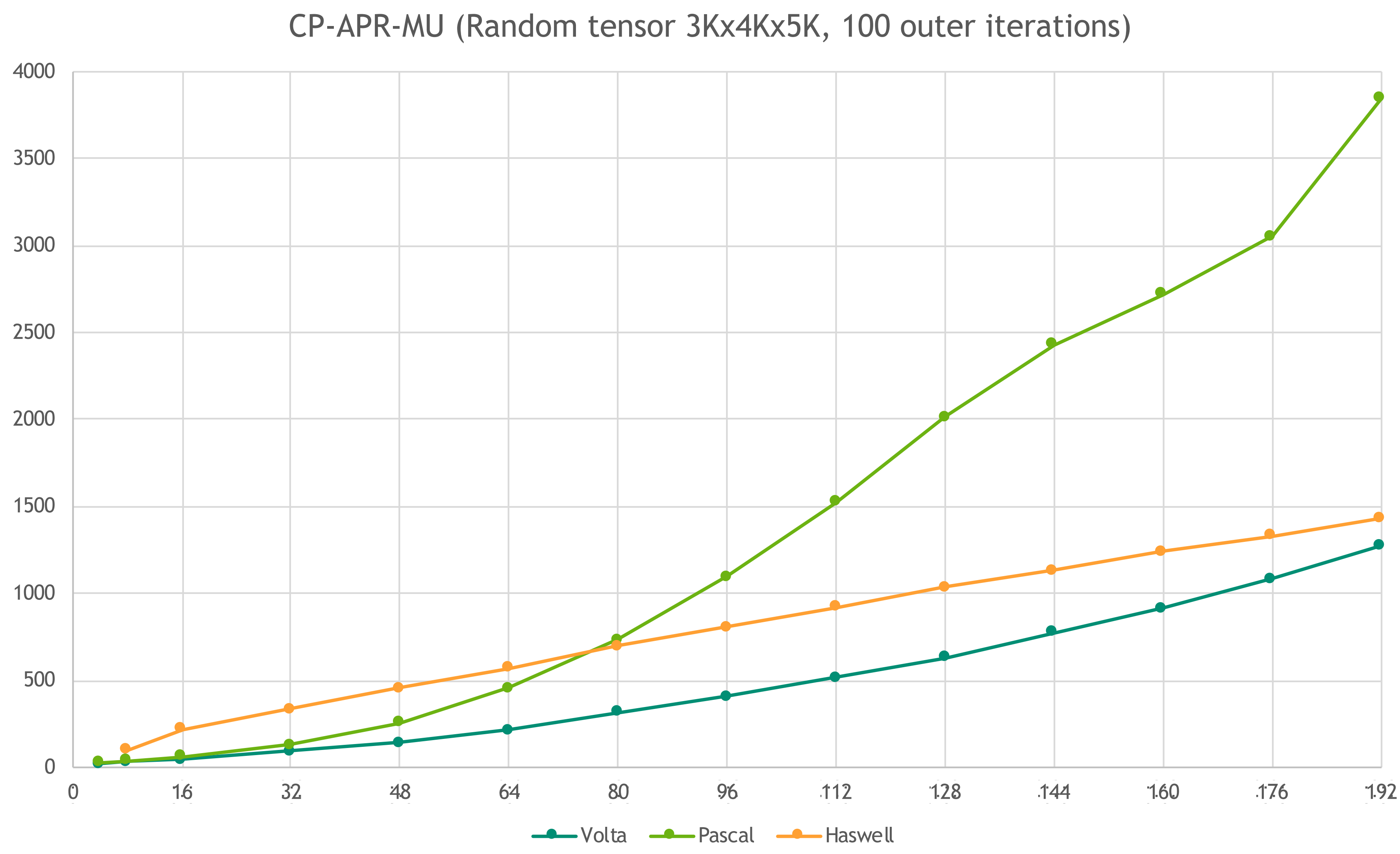




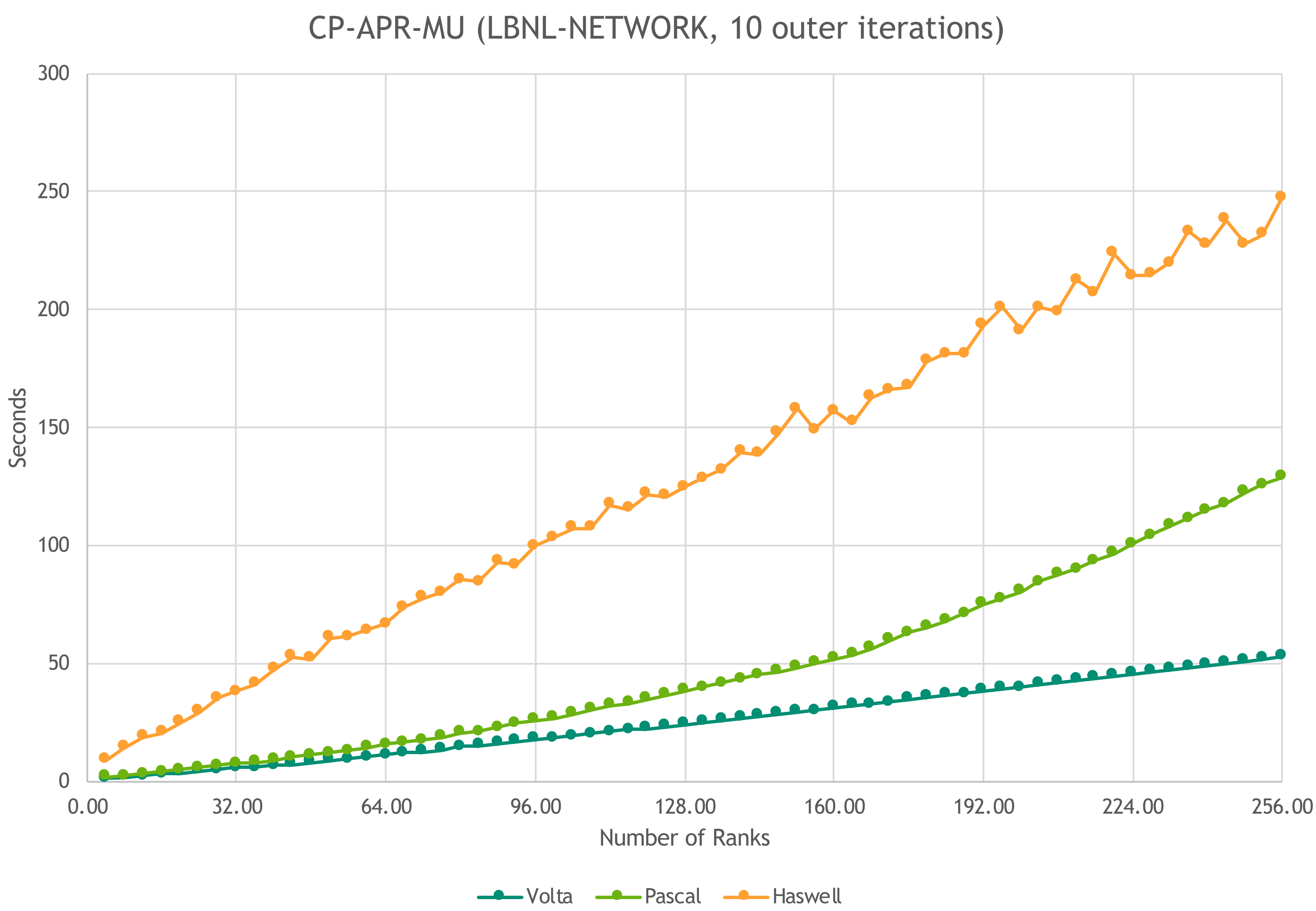
Intel CPUs: Software-based atomic operations

NVIDIA GPUs: Hardware-based atomic operations

Performance of CPU-APR-MU with respect to different rank size



Performance of CP-APR-MU (LBNL-Network) with respect to different rank sizes



Conclusion

Development of Portable on-node Parallel CP-APR Solvers

- Data parallelism for MU method
- Multiple Architecture Support using Kokkos
- Performance on CPU, Manycore and GPUs
- Two different work partitioning
  - CPU: Row-wise in each mode
  - GPU: Partition COO format
- Benefit from GPU atomics
  - Better capability wit latest GPUs

Future Work

- Better GPU support for PDNR and PQNR
- Performance tuning to handle irregular nonzero distributions and disparity in mode sizes