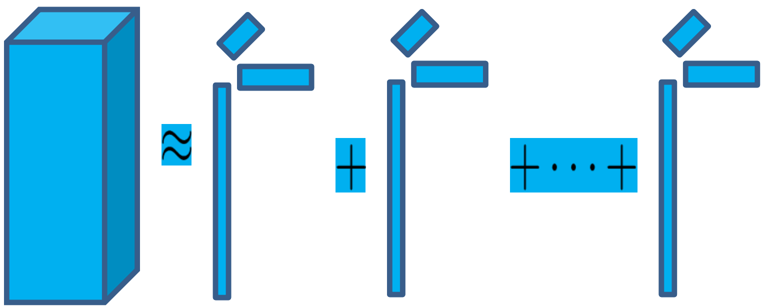
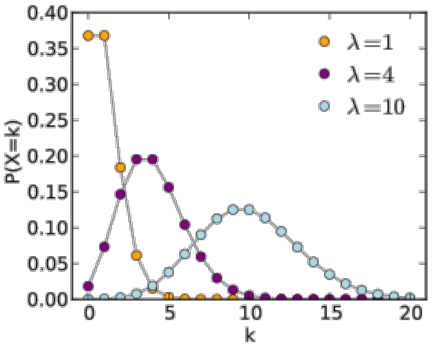
Exceptional service in the national interest

Sandia National Laboratories





Portability and Scalability of Sparse Tensor Decompositions on CPU/MIC/GPU Architectures

Christopher Forster, Keita Teranishi, Greg Mackey, Daniel Dunlavy and Tamara Kolda

SAND2017-6575 C

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

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

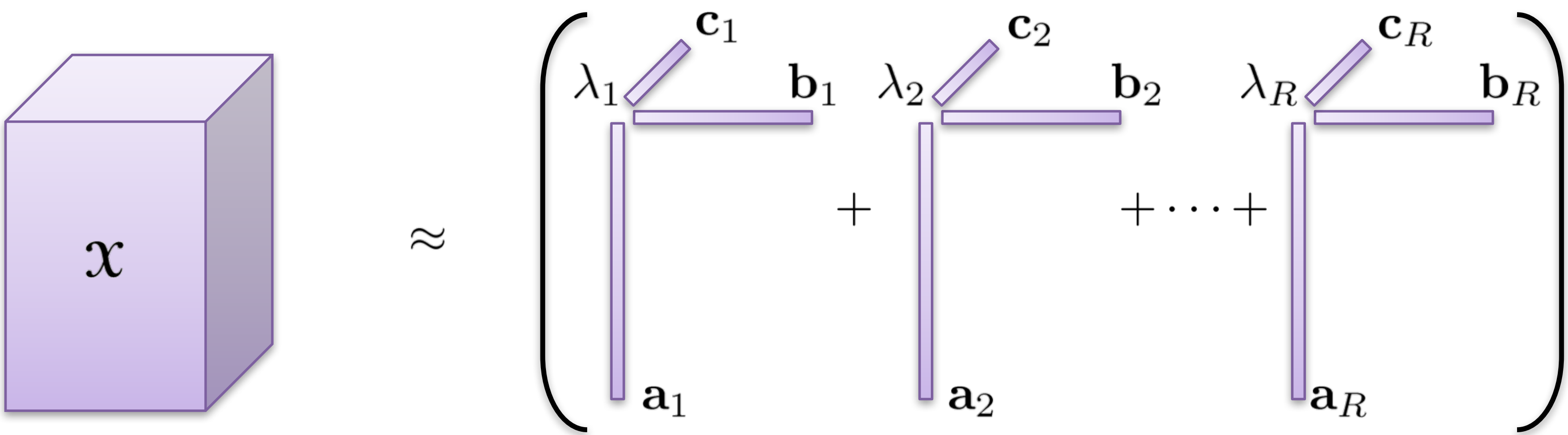
Sparse Tensor Decomposition

- Develop production quality library software to perform CP factorization with Alternating Poisson Regression on HPC platforms
  - SparTen
- 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



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

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

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

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

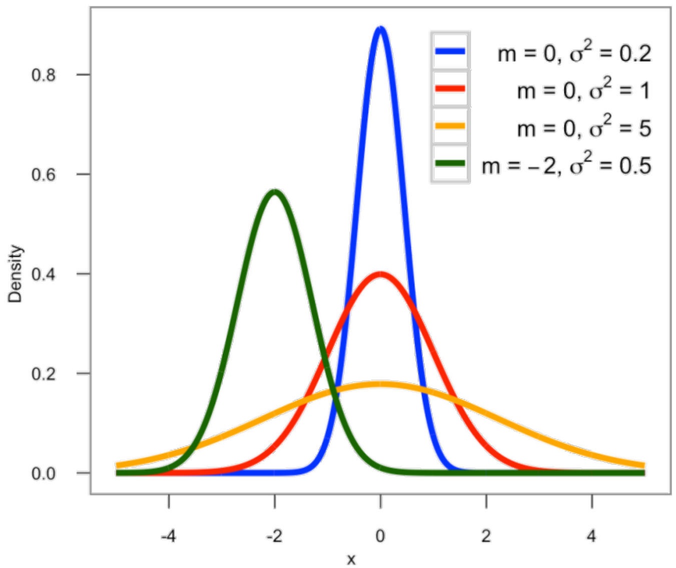
Poisson for Sparse Count Data

Gaussian (typical)

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

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

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

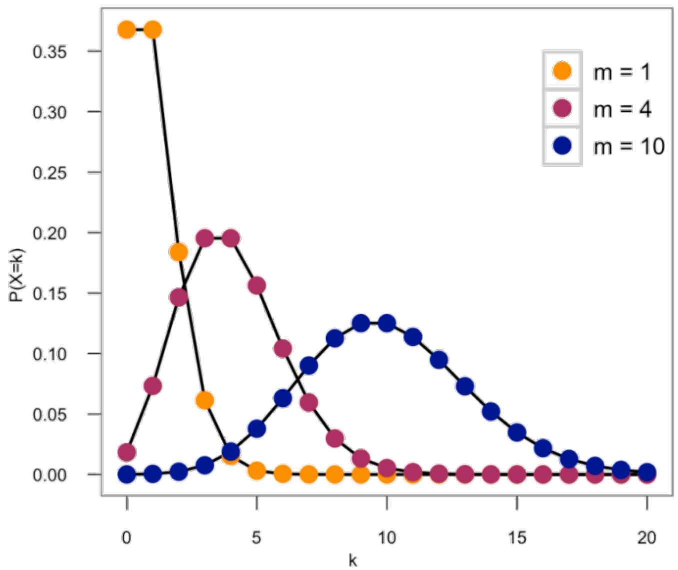


Poisson

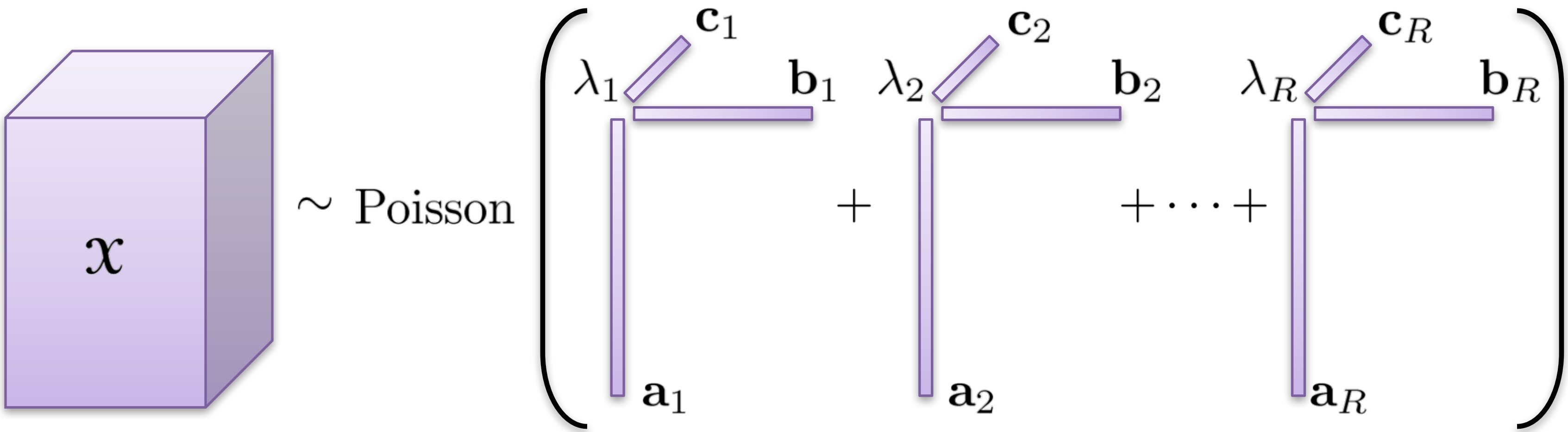
The random variable x is a discrete nonnegative integer.

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

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



Sparse Poisson Tensor Factorization



Model: 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

New Method:
Alternating Poisson Regression (CP-APR)

Repeat until converged…

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

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

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

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

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

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)

CP-APR

Algorithm 1: CPAPR, Alternating Block Framework

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

Input : Sparse N-mode Tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \dots I_N$ 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 $\Pi^{(n)} = (A^{(N)} \odot \dots \odot A^{(n+1)} \odot A^{(n-1)} \odot \dots A^{(1)})^T$

    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

Algorithm 1: CPAPR, Alternating Block Framework

1 CPAPR $(\mathcal{X}, \mathcal{M})$;
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 $\mathcal{M} = [\lambda; A^{(1)} \dots A^{(N)}]$
2 **Initialize**
3 **repeat**
4 f
5
6
7
8
9 ur

Minin
mi

- 2 major approaches
  - **Multiplicative Updates** like Lee & Seung (2000) for matrices, but extended by Chi and Kolda (2011) for tensors
  - **Newton and Quasi-Newton method for Row-subpblems** by Hansen, Plantenga and Kolda (2014)

Key Elements of MU and PDNR methods

Sandia National Laboratories

Multiplicative Update (MU)

- Key computations
  - Khatri-Rao Product $\Pi^{(n)}$
  - 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

Sandia National Laboratories

Algorithm 1: CP-APR-MU, Multiplicative Update

<u>CP-APR-MU</u> $(\mathcal{X}, \mathcal{M})$;

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 $\Lambda = \mathrm{diag}(\lambda)$

    end

until *all mode subproblems converged*;

Key Computations

CP-APR-PDNR

Algorithm 1: CPAPR-PDNR algorithm

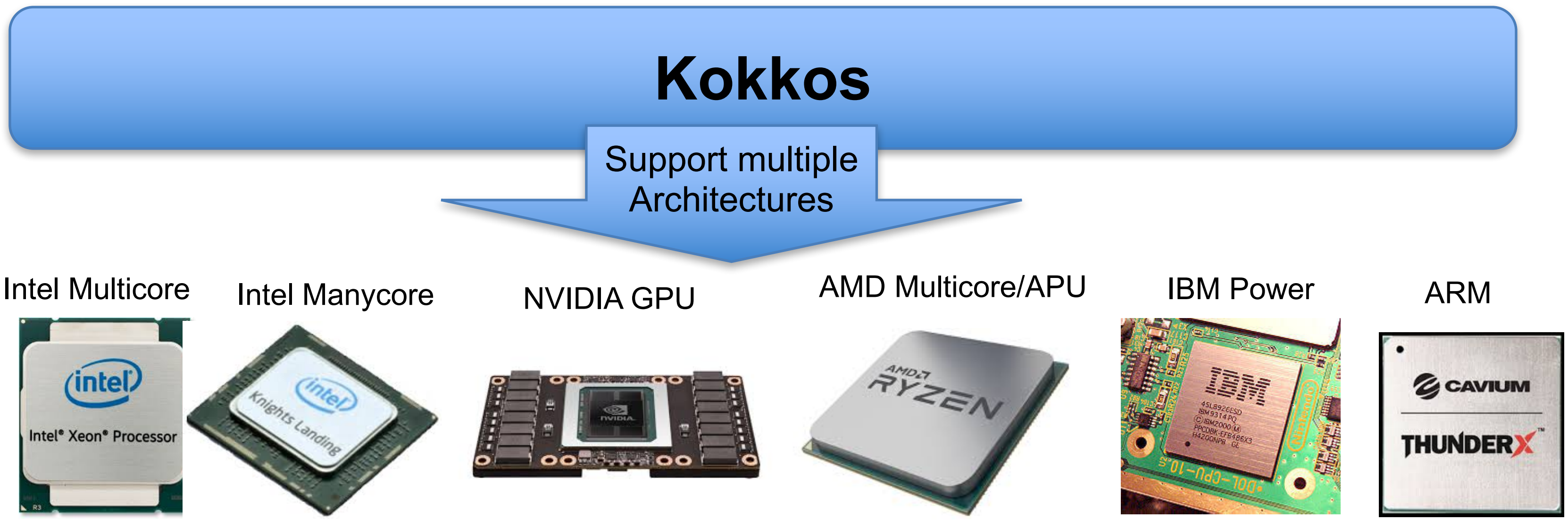
CPAPR_PDNR $(\mathcal{X}, \mathcal{M})$;
Input : Sparse N-mode Tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \dots I_N$ and the number of components R
Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \dots A^{(N)}]$
Initialize
repeat
  for $n = 1, \dots, N$ **do**
    Let $\Pi^{(n)} = (A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \dots A^{(1)})^T$
    for $i = 1, \dots, I_n$ **do**
      Find $b_i^{(n)}$ s.t. $\min\limits_{b_i^{(n)} \geq 0} f_{\text{row}}(b_i^{(n)}, x_i^{(n)}, \Pi^{(n)})$
    end
    $\lambda = e^T B^{(n)}$ where $B^{(n)} = [b_1^{(n)} \dots b_{I_n}^{(n)}]^T$
    $A^{(n)} \leftarrow B^{(n)} \Lambda^{-1}$, where $\Lambda = \text{diag}(\lambda)$
  end
until *all mode subproblems converged*;

Key Computations

PARALLEL CP-APR ALGORITHMS

Parallelizing CP-APR

- 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
    - Quad and Half float if needed.
- Kokkos::View() accommodates performance-aware multidimensional array data objects
  - Light-weight C++ class to
- 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 */
});
```

- Provide parallel loop operations using C++ language features
- Conceptually, the usage is no more difficult than OpenMP. The annotations just go in different places.

Why Kokkos?

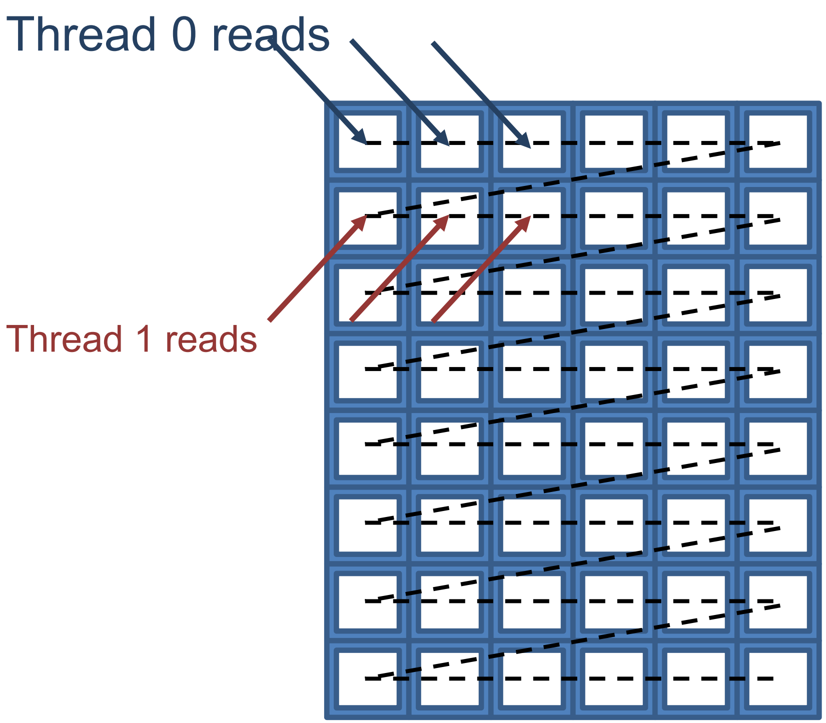
- Comply C++ language standard!
- Support multiple back-ends
  - Pthread, OpenMP, CUDA, Intel TBB and Qthread
- Support multiple data layout options
  - Column vs Row Major
  - Device/CPU memory
- Support different parallelism
  - Nesting support
  - Vector, threads, Warp, etc.
  - Task parallelism (under development)

Array Access by Kokkos

```
Kokkos::View<double **, Layout, Space>
```

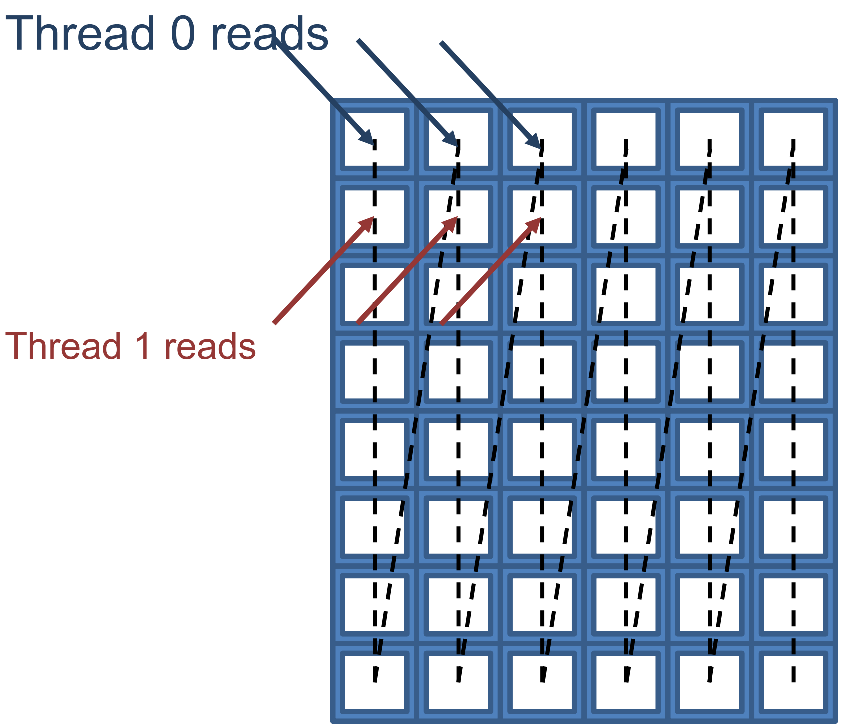
```
View<double **, Right, Space>
```

Row-major




```
View<double **, Left, Space>
```

Column-major

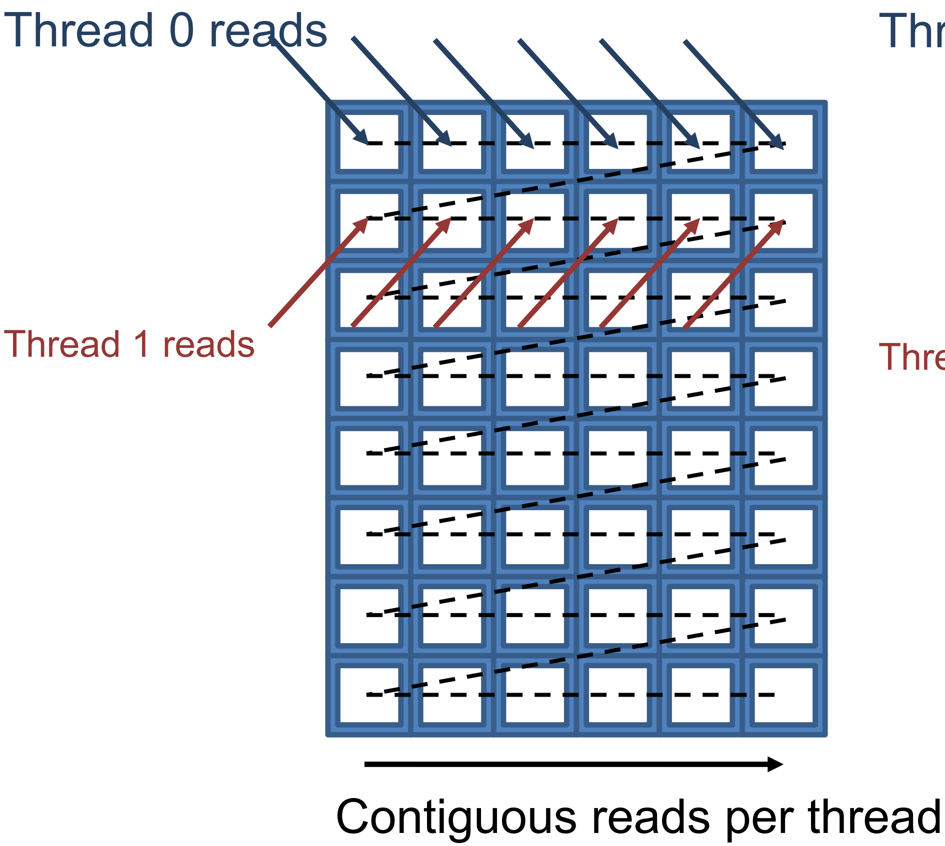


Array Access by Kokkos

```
Kokkos::View<double **, Layout, Space>
```

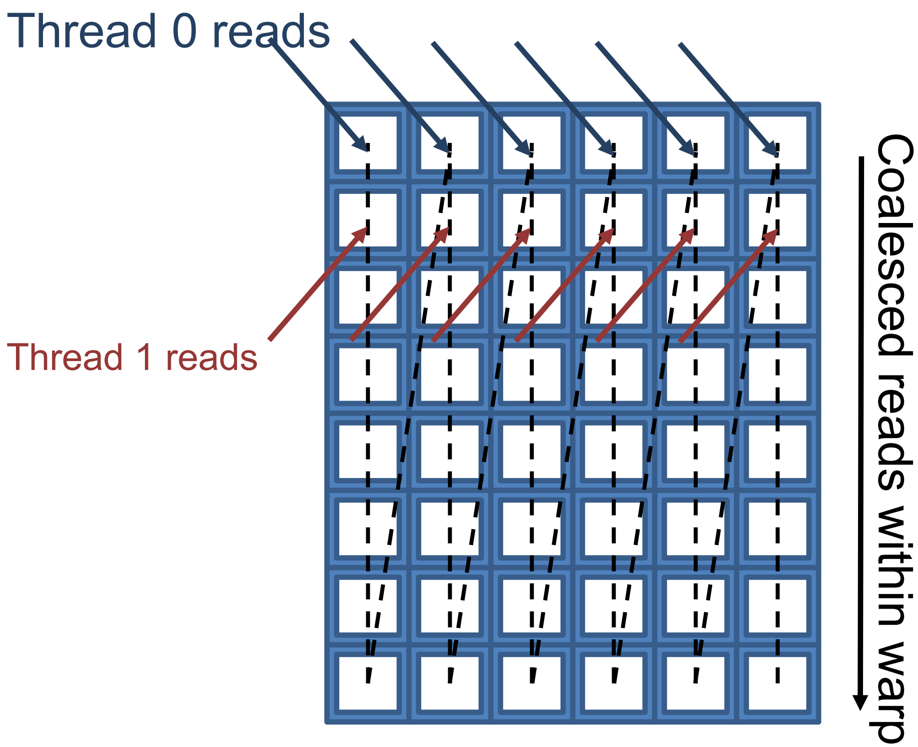
```
View<double **, Right, Host>
```

Row-major




```
View<double **, Left, CUDA>
```

Column-major



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 $I_1 \times I_2 \times \ldots I_N$ and the number of components R

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

initializeBuffer(X, R)

$\mathcal{E} \leftarrow$ computeIndexMap(X)

repeat

  for $n = 1, \ldots, N$ **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 $i = 1, \ldots, 10$ **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

Parallel CP-APR-PDNR

Algorithm 1: CP-APR-PDNR in source

<u>CP-APR-PDNR</u> X, M, R;

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

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

initializeBuffer(X, R)

$\mathcal{E} \leftarrow$ computeIndexMap(X)

repeat

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

    $M \leftarrow$ distribute(M, n) (Scale the elements of A^n by λ)

    $\Pi^{(n)} \leftarrow$ computePi$(A, \mathcal{E}^{(n)})$

    parallel_for $i = 1, \ldots, I_n$ **do**

      $a_i^n \leftarrow$ rowSolvePDNR$(a_i^n, X^n, \Pi^n, \mathcal{E}_i^{(n)})$

    end

    $M \leftarrow$ normalize(M, A, n)

  end

until *all mode subproblems converged*;

Data Parallel

Task Parallel

Notes on Data Structure

- Use Kokkos::View
- Sparse Tensor
  - Similar to the Coordinate (COO) Format in Sparse Matrix representation
- Kruskal Tensor & Khatri Rao Product
  - Provides options for row or column major
    - Kokkos::View provides an option to define the leading dimension.
    - Determined during compile or run time
- Avoid Atomics
  - Expensive in CPUs and Manycore
  - Use extra indexing data structure
- CP-APR-PDNR
  - Creates a pool of tasks
  - A dedicated buffer space (Kokkos::View) is assigned to individual task

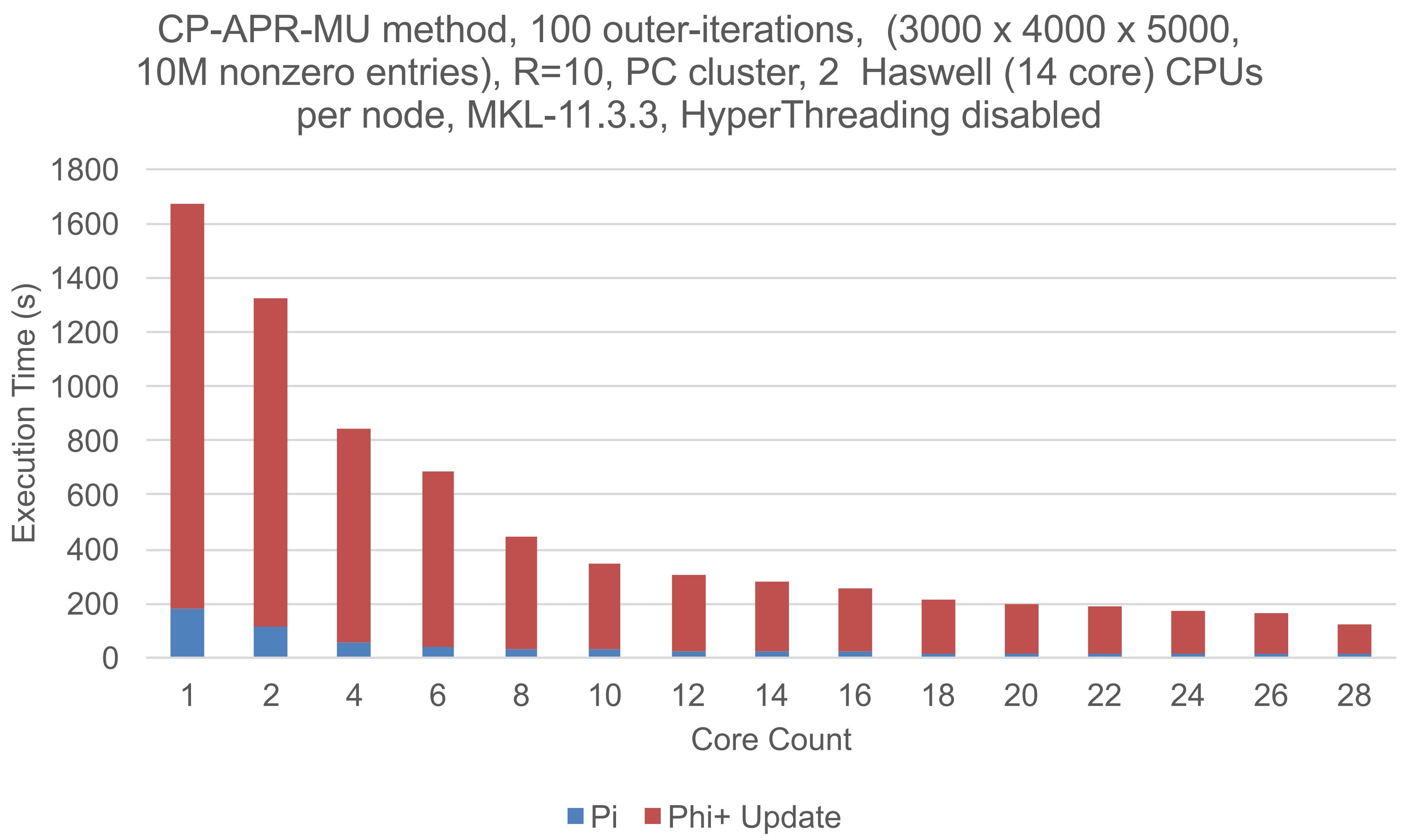
PERFORMANCE

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**
- Double Precision

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

CPAPR-MU on CPU (Random)

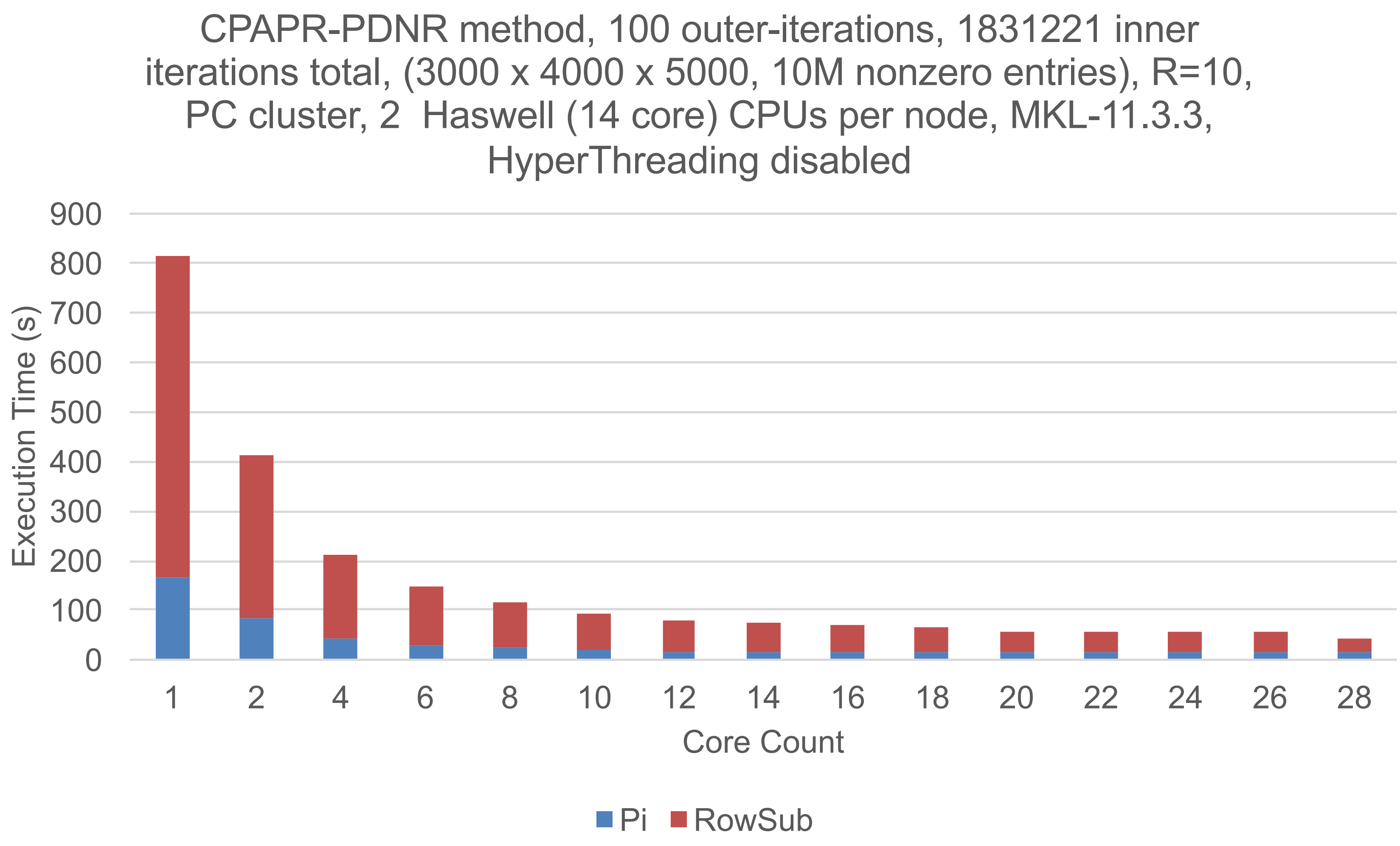


Results: CPAPR-MU Scalability

Data	Haswell CPU 1-core		2 Haswell CPUs 14-cores		2 Haswell CPUs 28-cores		KNL 68-core CPU		NVIDIA P100 GPU	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
Random	1715*	1	279	6.14	165	10.39	20	85.74	10	171.5
LBNL	131	1	32	4.09	32	4.09	103	1.27		
NELL-2	1226	1	159	7.77	92	13.32	873	1.40		
NELL-1	5410	1	569	9.51	349	15.50	1690	3.20		
Delicious	5761	1	2542	2.26	2524	2.28				

100 outer iterations for the random problem
10 outer iterations for realistic problems
* Pre-Kokkos C++ code on 2 Haswell CPUs:
  1-core, 2136 sec
  14-cores, 762 sec
  28-cores, 538 sec

CPAPR-PDNR on CPU(Random)



Results: CPAPR-PDNR Scalability

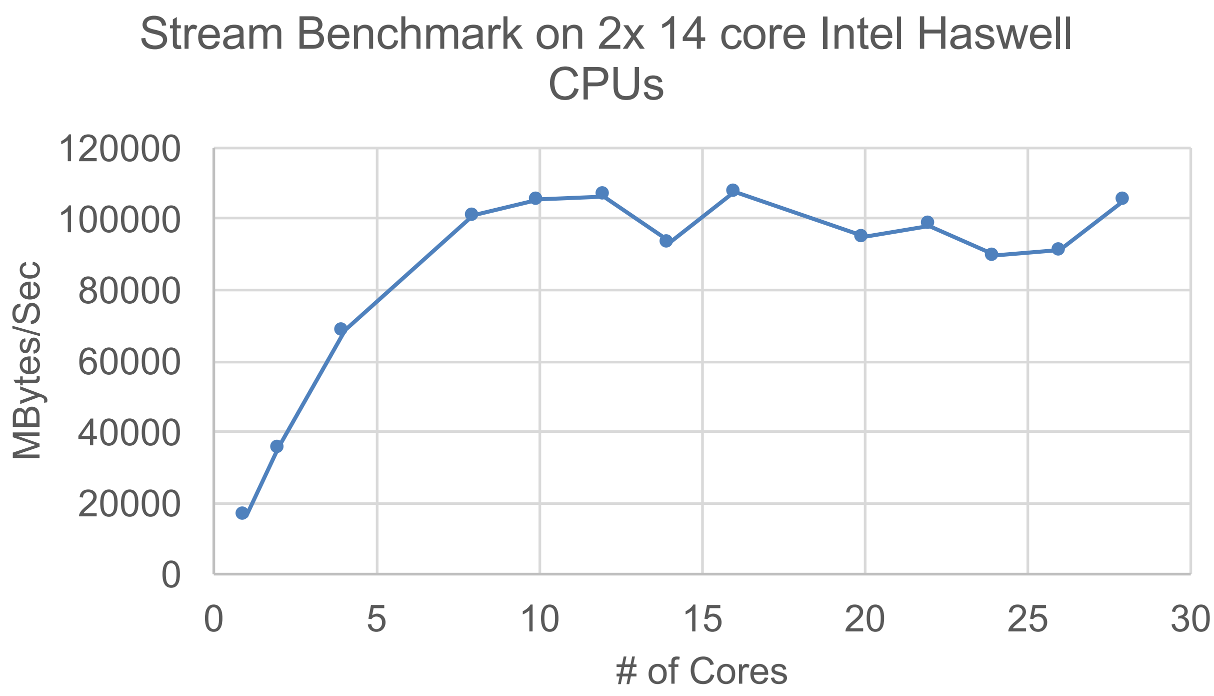
Data	Haswell CPU 1 core		2 Haswell CPUs 14 cores		2 Haswell CPUs 28 cores	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
Random	817*	1	73	11.19	44	18.58
LBNL	441	1	187	2.35	191	2.30
NELL-2	2162	1	326	6.63	319	6.77
NELL-1	17212	1	4241	4.05	3974	4.33
Delicious	18992	1	3684	5.15	3138	6.05

100 outer iterations for the random problem
10 outer iterations for realistic problems
* Pre-Kokkos C++ code spends 3270 sec on 1 core

Performance Issues

- Our implementation exhibits very good scalability with the random tensor.
  - Similar mode sizes
  - Regular distribution of nonzero entries
    - Some cache effects
    - Kokkos is NUMA-aware for contiguous memory access (first-touch)
- Some scalability issues with the realistic tensor problems.
  - Irregular nonzero distribution and disparity in mode sizes
  - Task-parallel code may have some memory locality issues to access sparse tensor, Kruskal Tensor, and Khatori-Rao product
  - Preprocessing could improve the locality
    - Explicit Data partitioning (Smith and Karypis)
    - Possible to implement using Kokkos

Memory Bandwidth (Stream Benchmark)



- All cores deliver approximately 8x performance improvement from single thread
- Hard to scale using all cores with memory-bound code.

Conclusion

- Development of Portable on-node Parallel CP-APR Solvers
  - Data parallelism for MU method
  - Mixed Data/Task parallelism for PDNR method
  - Multiple Architecture Support using Kokkos
- Scalable Performance for random sparse tensor
- Future Work
  - Projected Quasi-Newton for Row-subproblems (PQNR)
  - GPU and Manycore support for PDNR and PQNR
  - Performance tuning to handle irregular nonzero distributions and disparity in mode sizes