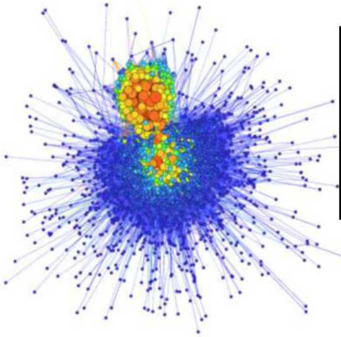
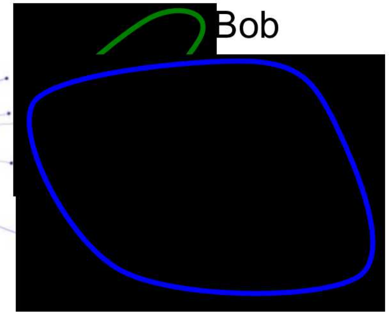
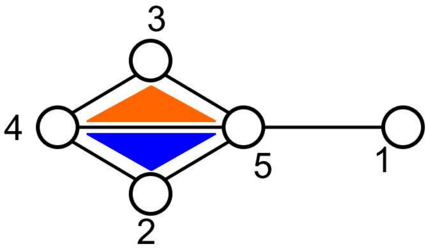
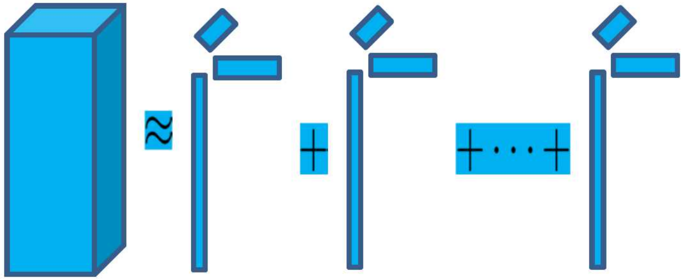
SAND2018-12353C

CCR
Center for Computing Research









High-Performance Portable Data Analytics Software Using the Kokkos Ecosystem

kokkos

Michael Wolf

CLSAC 2018 (November 1, 2018)

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.

Acknowledgements

- Kokkos (slides from Christian Trott, Siva Rajamanickam)
- Grafiki (previously TriData)
  - MW, Danny Dunlavy, Rich Lehoucq, Jon Berry, Nathan Ellingwood, Daniel Bourgeois (Rice)
- Tensor Work (slides from Dunlavy's TRICAP talk)
  - Data Analytics: Danny Dunlavy, Keita Teranishi, Rich Lehoucq, Richard Barrett, Tammy Kolda, Chris Forster (NVIDIA), Karen Devine
  - Related Work: Eric Phipps, Siva Rajamanickam
- Kokkos Kernels for Graph/Data Analytics
  - Siva Rajamanickam (PI), MW, Mehmet Deveci (Intel), Jon Berry, Abdurrahman Yassar (GT), Umit Catalyurek (GT)

Performance Portability Motivation

Intel Multicore

NVIDIA GPU

IBM Power

Intel Manycore

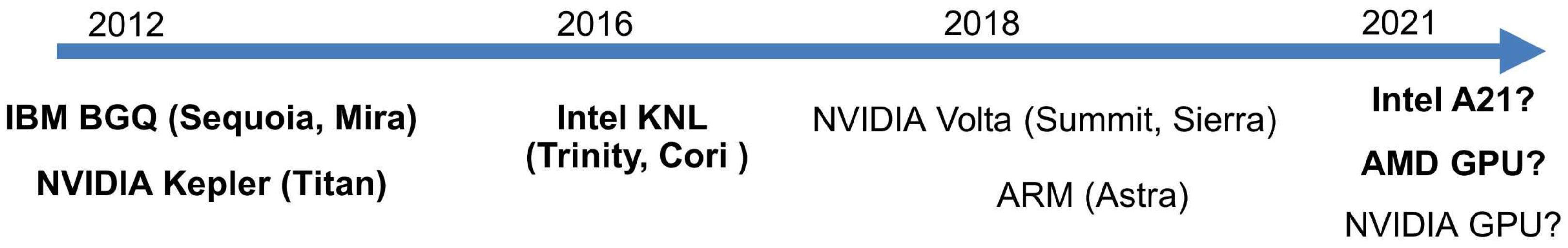
AMD Multicore/APU

ARM

- **Several many/multi-core architectures central to DOE HPC**
- **Applications struggle to obtain good performance on all of these**

Performance Portability Motivation

- Example: Architecture Change NVIDIA Pascal to Volta
  - Warps can arbitrarily, permanently diverge, and branches can now interleave
  - Took **2 man months** to fix in Kokkos for just **3 code positions**
  - Without abstraction: **~400 places** in Trilinos (excluding Kokkos) would need fixes
- Timeline for Architectures:
  - In Bold: requires new approach for performance for the first time



2012

IBM BGQ (Sequoia, Mira)

NVIDIA Kepler (Titan)

2016

Intel KNL (Trinity, Cori)

2018

NVIDIA Volta (Summit, Sierra)

ARM (Astra)

2021

Intel A21?

AMD GPU?

NVIDIA GPU?

1 Decade of HPC will have seen 4-5 “new” paradigms!

Performance Portability or Bust?

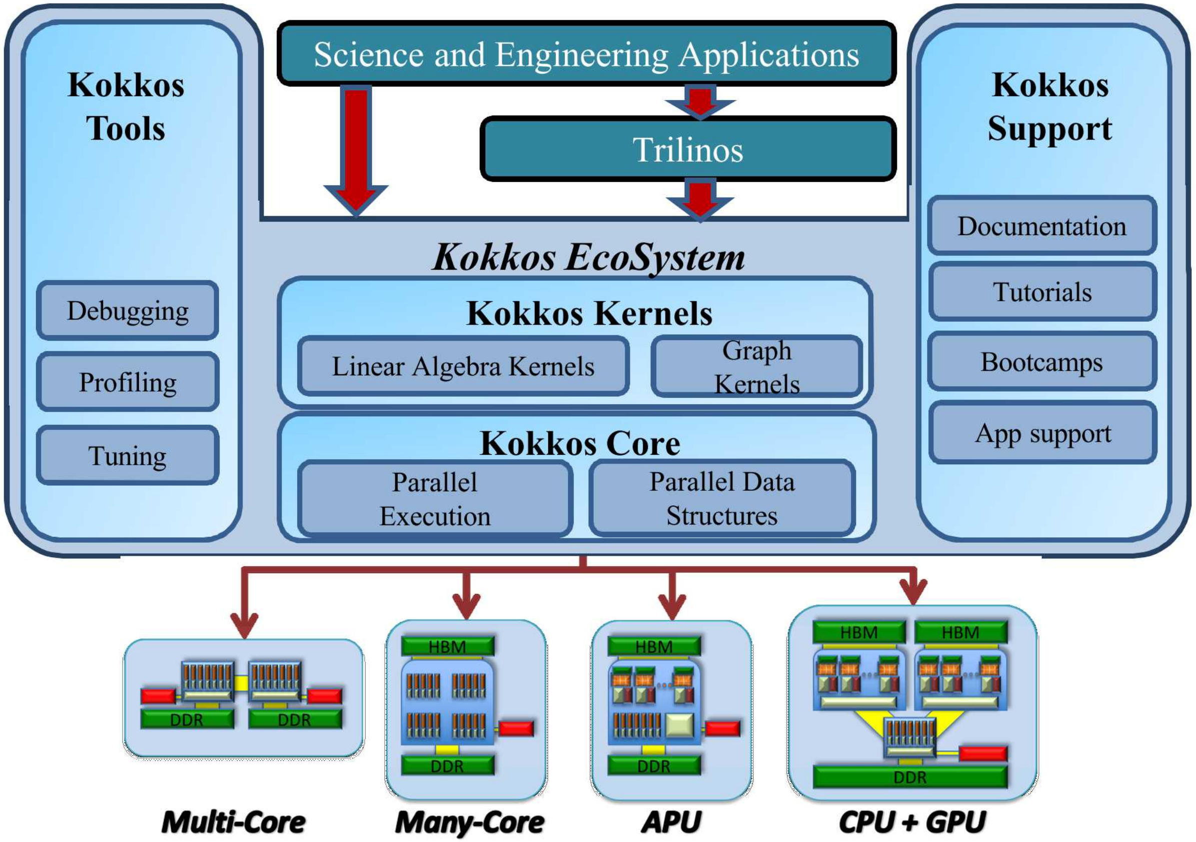
10 LOC / hour ~ 20k LOC / year

- Optimistic estimate: 10% of application needs to get rewritten for adoption of Shared Memory Parallel Programming Model
- Typical Apps: 300k – 600k Lines
  - Uintah: 500k, QMCPack: 400k, LAMMPS: 600k; QuantumEspresso: 400k
  - Typical App Port thus 2-3 Man-Years
  - Sandia maintains a couple dozen of those
- Large Scientific Libraries
  - E3SM: 1,000k Lines x 10% => 5 Man-Years
  - Trilinos: 4,000k Lines x 10% => 20 Man-Years

Sandia alone: 50-80 Man Years

Convincing applications to support even one MPI+X programming model challenging

Kokkos Ecosystem for Performance Portability



Kokkos Core: parallel patterns and data structures, supports several execution and memory spaces

Kokkos Kernels: performance portable BLAS, sparse, and graph algorithms and kernels

Kokkos Tools: debugging and profiling support

Kokkos Ecosystem addresses complexity of supporting numerous many/multi-core architectures that are central to DOE HPC enterprise

Why Kokkos?

- **Support multiple back-ends**
  - Pthread, OpenMP, CUDA, Intel TBB, Qthreads
  - Work closely with hardware vendors
- **Support multiple data layout options**
  - Column vs Row Major ; Device/CPU memory
- **Support different parallelism**
  - Nested loop support; vector, threads, warps, etc.
  - Task parallelism
- **Growing Kokkos Support**
  - Community: ORNL, LANL, CSCS, Juelich, Slack Channel (80+ members)
  - Kokkos abstractions migrating to C++ standard

Kokkos team eager to engage with new customers to support new applications and architectures

DOE Kokkos Users

We don’t actually know who all is using Kokkos. Partial ECP List:

Application	State
SNL ATDM Apps	Base (SPARC, EMPIRE, Nimble,…)
LANL ATDM Apps	In Parts
EXAALT	Base Code
QMCPack	Evaluation
ExaWind	Base Code
ExaAM	Experimenting
LatticeQCD	Experimenting
ProxyApp	Base Code (in parts)
COPA	Base Code
ExaGraph	Base Code (in parts)
ExaLearn	Committed (in parts)

Software Technology	State
SNL ATDM PMR	This is Kokkos ;-)
LANL ATDM PMR	Experimenting
KokkosSupport	
SNL ATDM DevTools	Base Code (in parts)
ExaPapi	Integrates KokkosTools
SNL ATDM Math	Base Code
ForTrilinos	Base Code
PEEKS	Base Code

Additionally:

- Many ASC applications at Sandia are porting or using Kokkos in their base code.
- Many applications leverage Kokkos through Trilinos framework’s solvers.

- **Kokkos (originally SNL effort) becoming community-wide effort**
- **Kokkos has a growing DOE user base**

Kokkos and Greater HPC Community

THE UNIVERSITY OF UTAH
IPP Max-Planck-Institut für Plasmaphysik
GT
ARL
U.S. NAVAL RESEARCH LABORATORY
University of BRISTOL
cea
JÜLICH FORSCHUNGSZENTRUM
NASA
UNIVERSITY OF CAMBRIDGE
CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre
AMD
intel
NVIDIA
ARM

- Many Institutions outside of DOE started experimenting with Kokkos or have projects that are already committed
- Additional institutions leveraging Kokkos indirectly via Trilinos solvers

What is Kokkos?

- **Templated C++ Library**
  - Goal: Write algorithms once, run everywhere (almost) optimally
  - Serve as substrate layer of sparse matrix and vector kernels
- **Kokkos::View() accommodates performance-aware multidimensional array data objects**
  - Light-weight C++ class
- **Parallelizing loops using C++ language standard**
  - Lambda, Functors
- **Extensive support of atomics**
- **Substantial DOE investment**
  - **ECP/ATDM software technology (Ecosystem ~$3M/year)**
  - **Many DOE ECP and ASC applications use Kokkos**

Parallel Loops 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 */
});
```

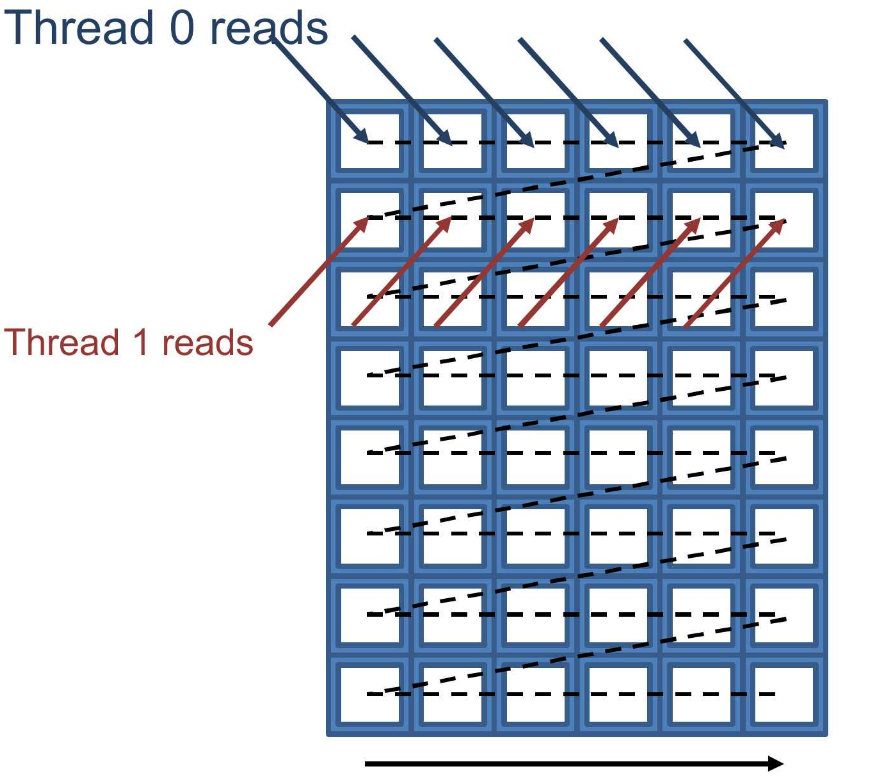
- Provide parallel loop operations using C++ language features
- Conceptually, the usage is no more difficult than OpenMP. The annotations just go in different places.

Array Access with Kokkos

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

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

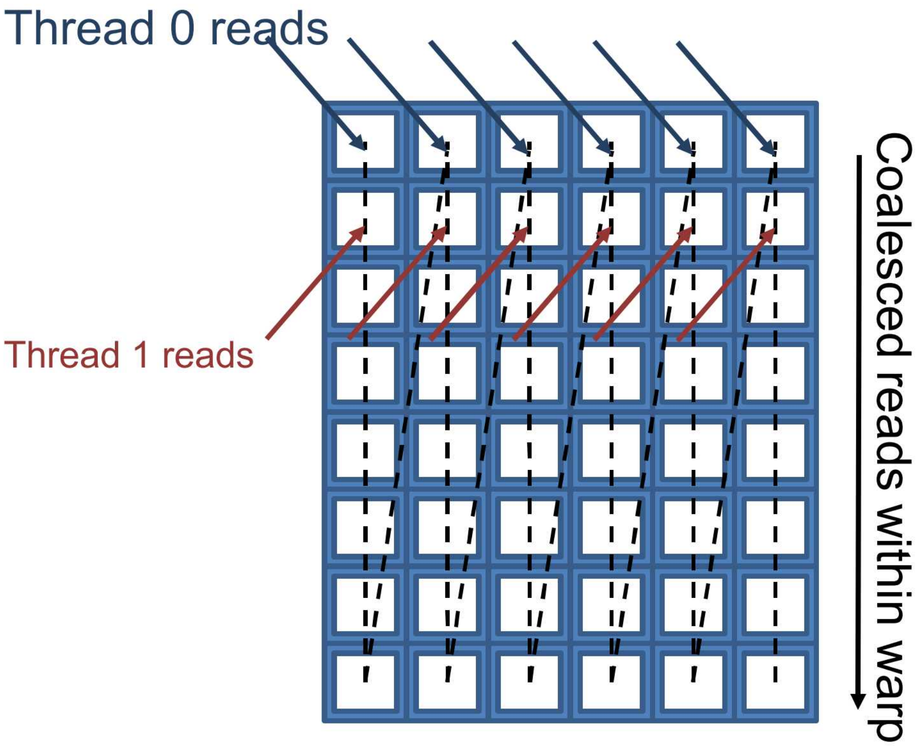
Row-major



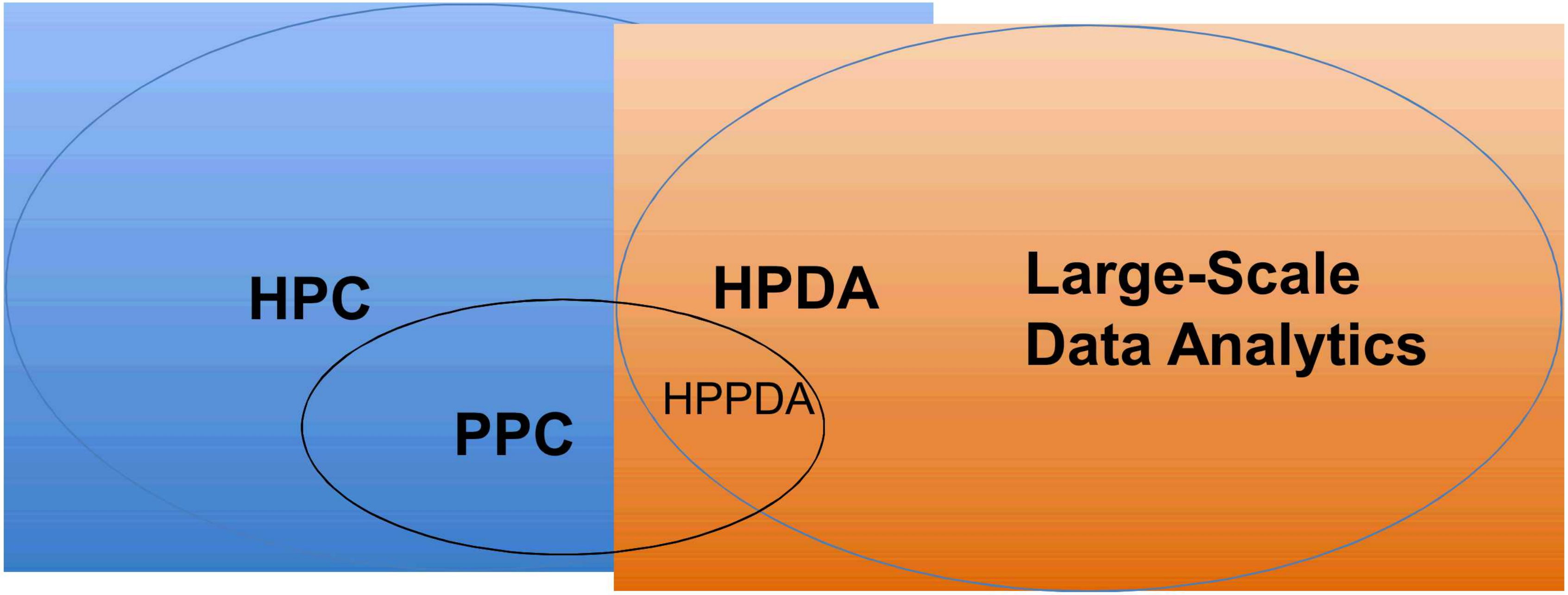
Contiguous reads per thread

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

Column-major



HPPDA and Kokkos



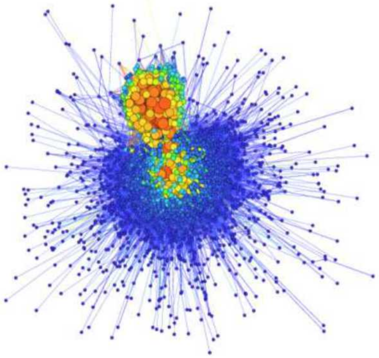
- **Performance-Portable Computing (PPC) (*Sandia: Kokkos*)**
- **High-Performance Data Analytics (HPDA)**
  - Use HPC to do big data analytics faster
- **High-Performance-Portable Data Analytics (HPPDA)**
  - Use PPC to enable HPDA on DOE platforms (***Sandia: Kokkos****)*

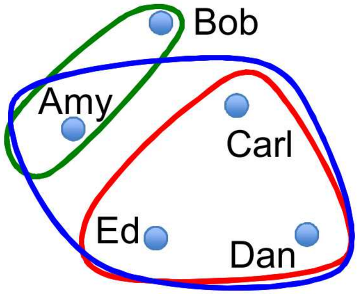
Leverage significant DOE investment in performance portability computing to impact large-scale data analytics

Use Case 1: Grafiki

- Formerly TriData – Trilinos for Large-Scale Data Analysis
  - Leverages Trilinos Framework (Sandia National Labs)
    - High performance linear algebra, traditional focus on CSE
    - High performing eigensolvers, linear solvers
    - Scales to billions of matrix rows/vertices
- Vision: Sparse Linear Algebra-Based Data Analysis
  - Apply sparse linear algebra techniques to data analysis
  - Target: very large data problems
  - Target: distributed memory and single node HPC architectures
- Additionally
  - Vehicle for improving how Trilinos can be leveraged for data analytics (e.g., submatrix extraction, preconditioning, load-balancing)
  - Support GraphBLAS-like linear algebra analysis techniques
- Focus: Graph and Hypergraph Analysis

Grafiki Capabilities





- Eigen solver based capabilities
  - Spectral Clustering, Vertex/Edge eigencentrality (graphs, hypergraphs)
  - Supports several eigensolvers (through Trilinos): LOBPCG, TraceMin-Davidson, Riemannian Trust Region, Block Krylov-Schur
- Linear solver based capability
  - Mean hitting time analysis on graphs
  - Support for different linear solvers (typically use CG) and preconditioners
- Other
  - K-means++, metrics (conductance, modularity, jaccard index)
  - Random graph and hypergraph models, hypothesis testing techniques/infrastructure for evaluation of clustering software

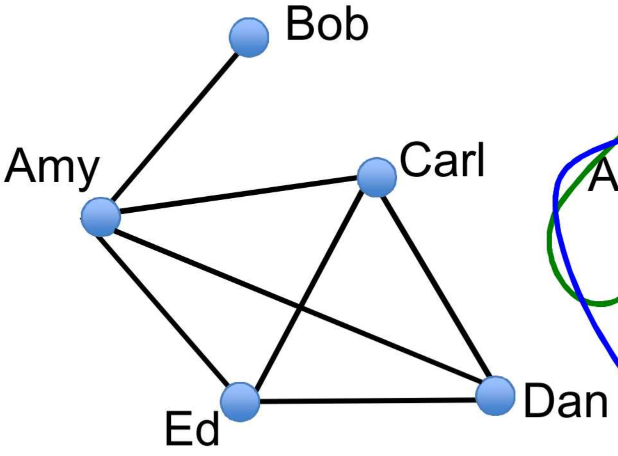
Hypergraphs

Emails

Users	1	2	3
Amy	x		x
Bob	x		
Carl		x	x
Dan		x	x
Ed		x	x

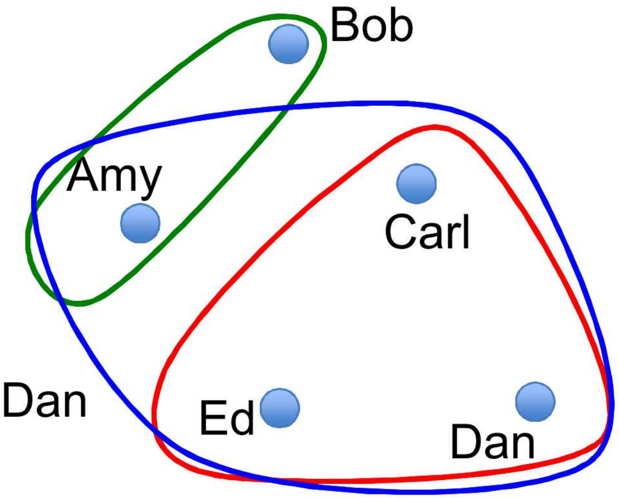
Relational Data

Graph



Edges connect
2 vertices

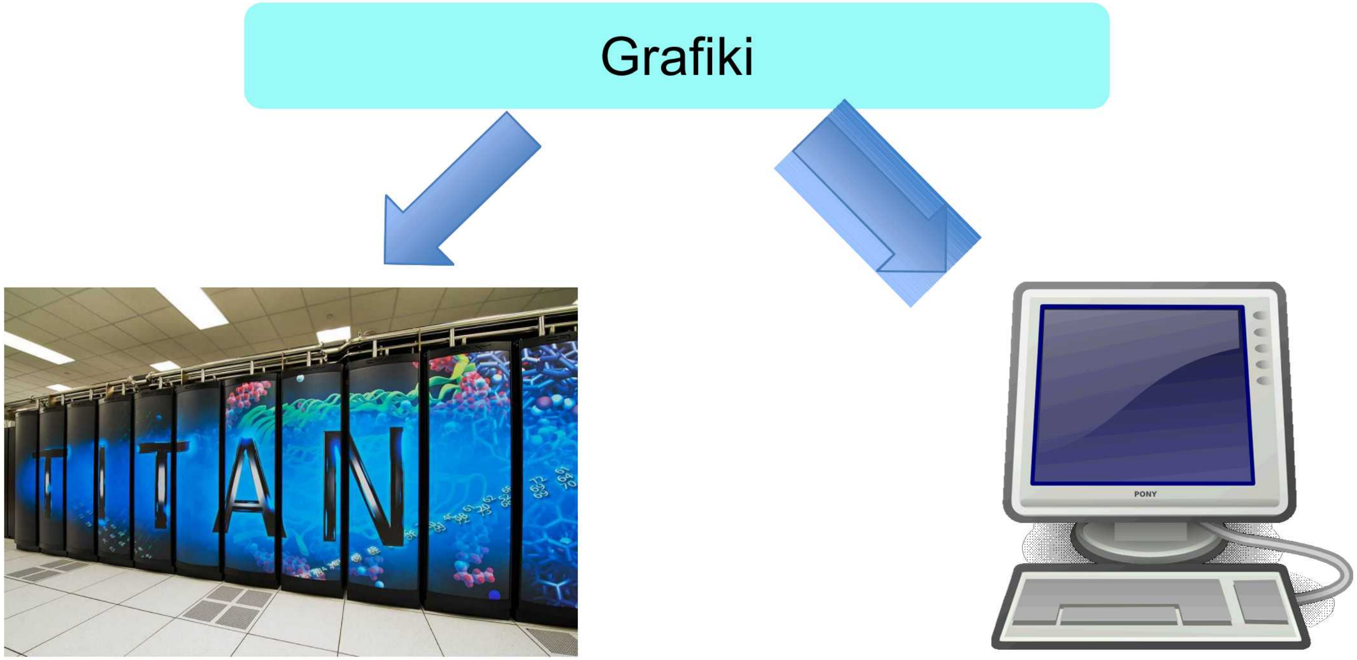
Hypergraph



Hyperedges connect
1 or more vertices

- Generalization of graph
- Convenient representation of relational data
- Computation and storage advantages

Grafiki Approach



Grafiki

Distributed Memory (DM)

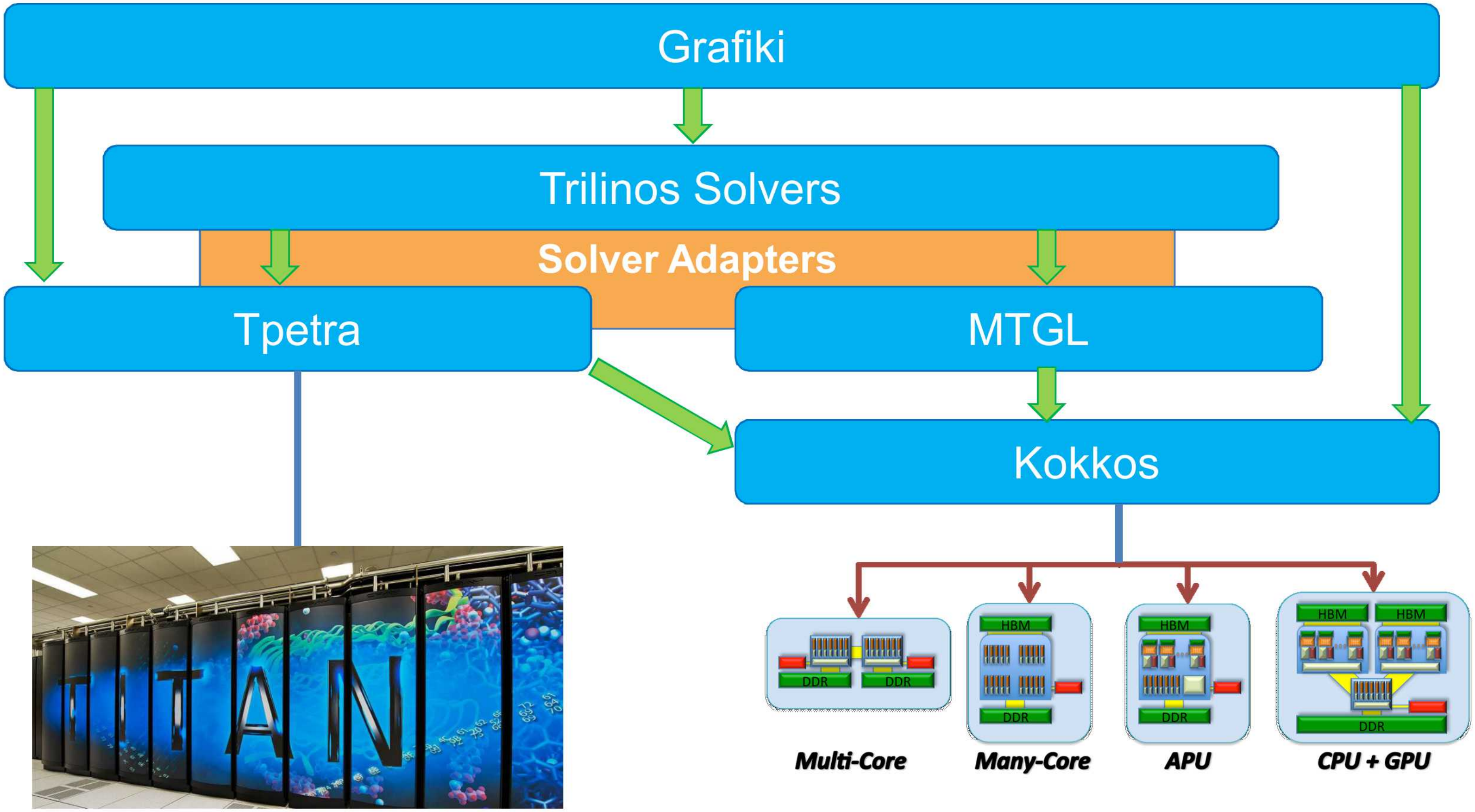
- Clusters, supercomputers
- Tpetra (MPI, DM data structures)
- **Kokkos** (node level parallelism)

Workstation

- CPUs, GPUs, KNLs, …
- **Kokkos**
- MTGL

Goal: Write algorithms once, run on both types of architectures

Grafiki Software Stack



Grafiki

Trilinos Solvers

Solver Adapters

Tpetra

MTGL

Kokkos

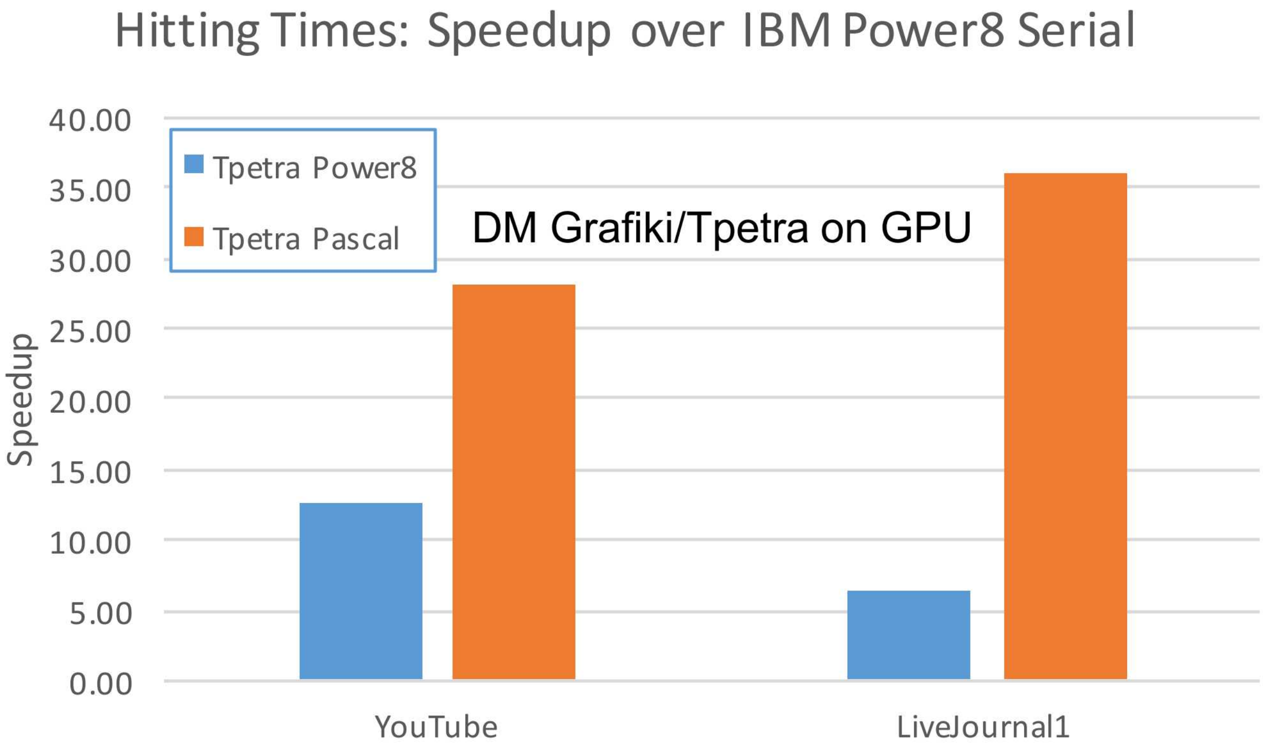
Distributed memory computations

Portable on-node performance

Flexible solver adapters enable solution for both architectures

Mean Hitting Time Results

MHT: Linear solver based analytic



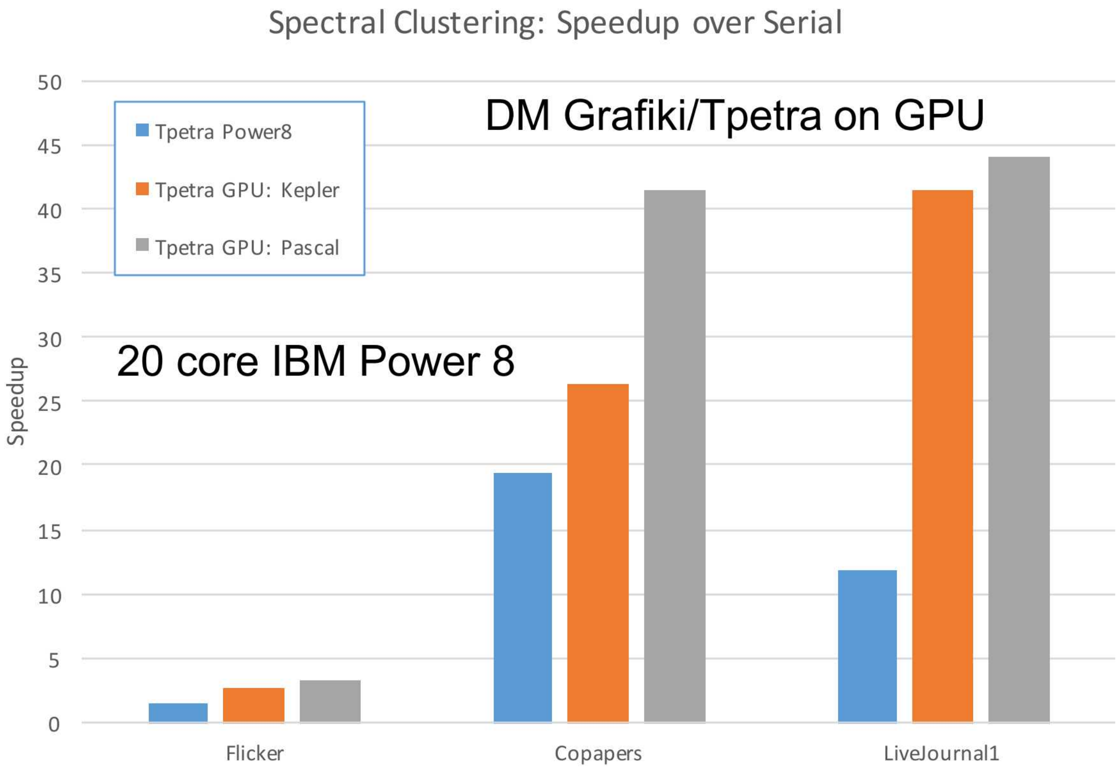
DM Grafiki/Tpetra on GPU

(Number of Edges) (2 M) (34 M)

- **Solver/Kokkos stack allows analytic to be written in architecture agnostic manner – no architecture optimization**
- **GPU computation is up to 35x speedup over host serial**

Grafiki Spectral Clustering Results

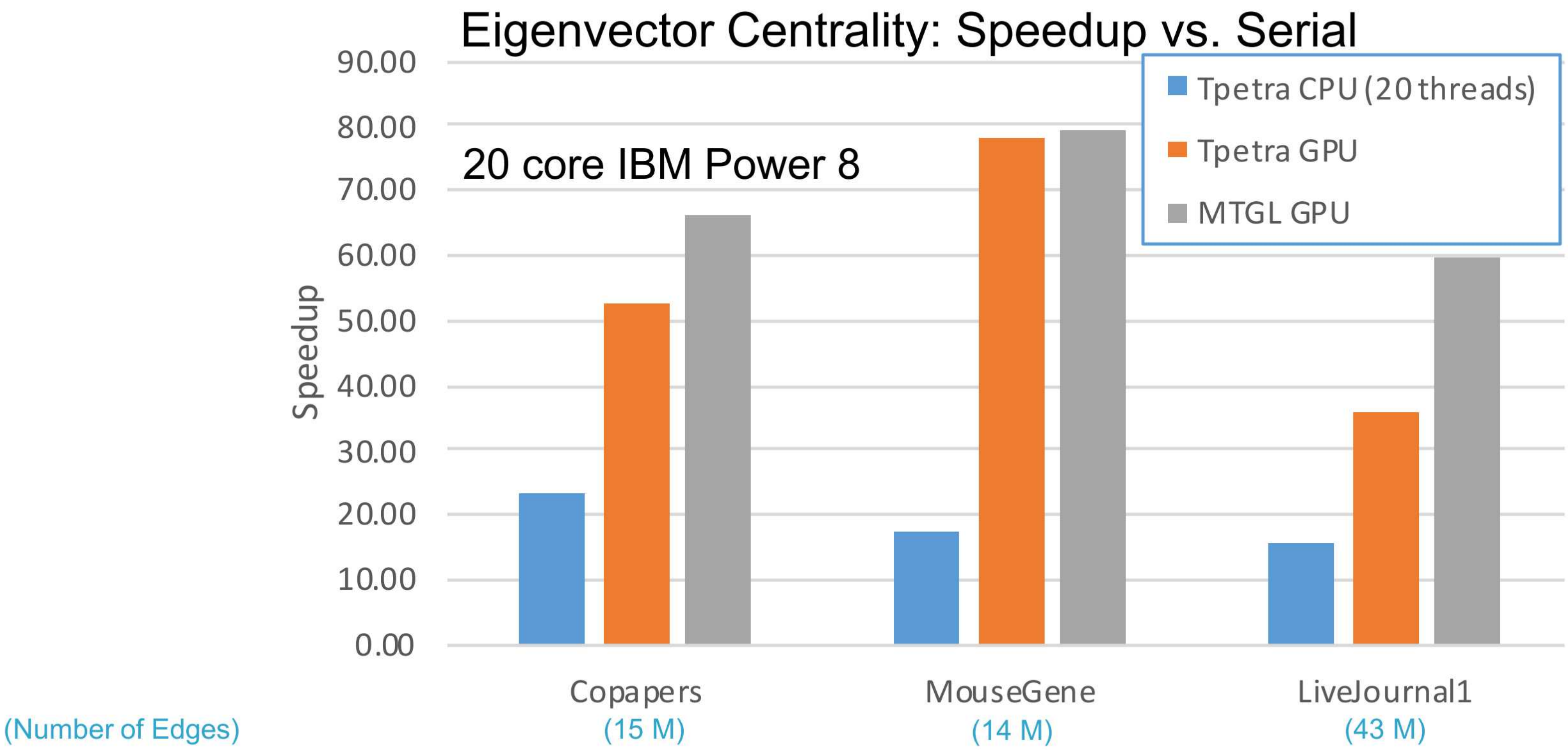
Spectral clustering: Eigensolver based analytic



- **Solver/Kokkos stack allows analytic to be written in architecture agnostic manner**
- **GPU computation up to 45x speedup over host serial**

Grafiki Centrality Results: Tpetra and MTGL

EV centrality: Eigensolver based analytic



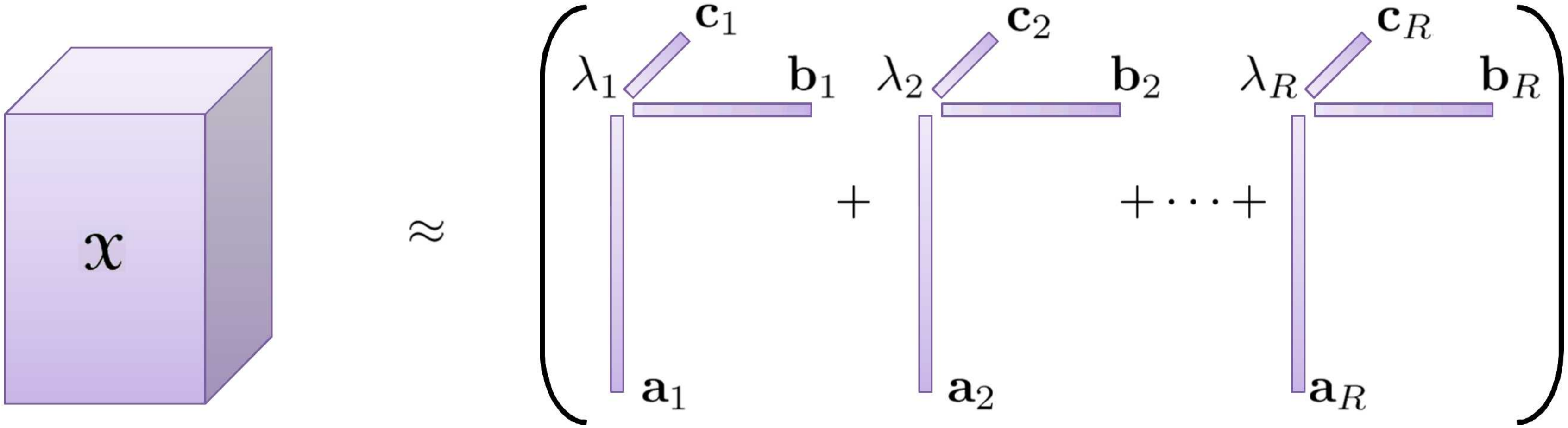
- **Solver/Kokkos stack allows analytic to be written in architecture agnostic manner**
- **GPU computation is up to 80x speedup over host serial**

Use Case 2: Scalable Tensor Factorizations

- **Motivation: Count Data**
  - Network analysis
  - Term-document analysis
  - Email analysis
  - Link prediction
  - Web page analysis
- **Large, Sparse Data**
  - Number of dimensions = 4, 5, 6, …
  - Example tensor size: 10^4 x 10^4 x 10^6 x 10^6 x 10^7
  - Example densities: 10^{-8} to 10^{-16}
- Targeting several multi/many-core architectures
  - Intel CPU, Intel MIC, NVIDIA GPU, IBM Power 9, etc.

CP Tensor Decomposition

CANDECOMP/PARAFAC (CP) Model



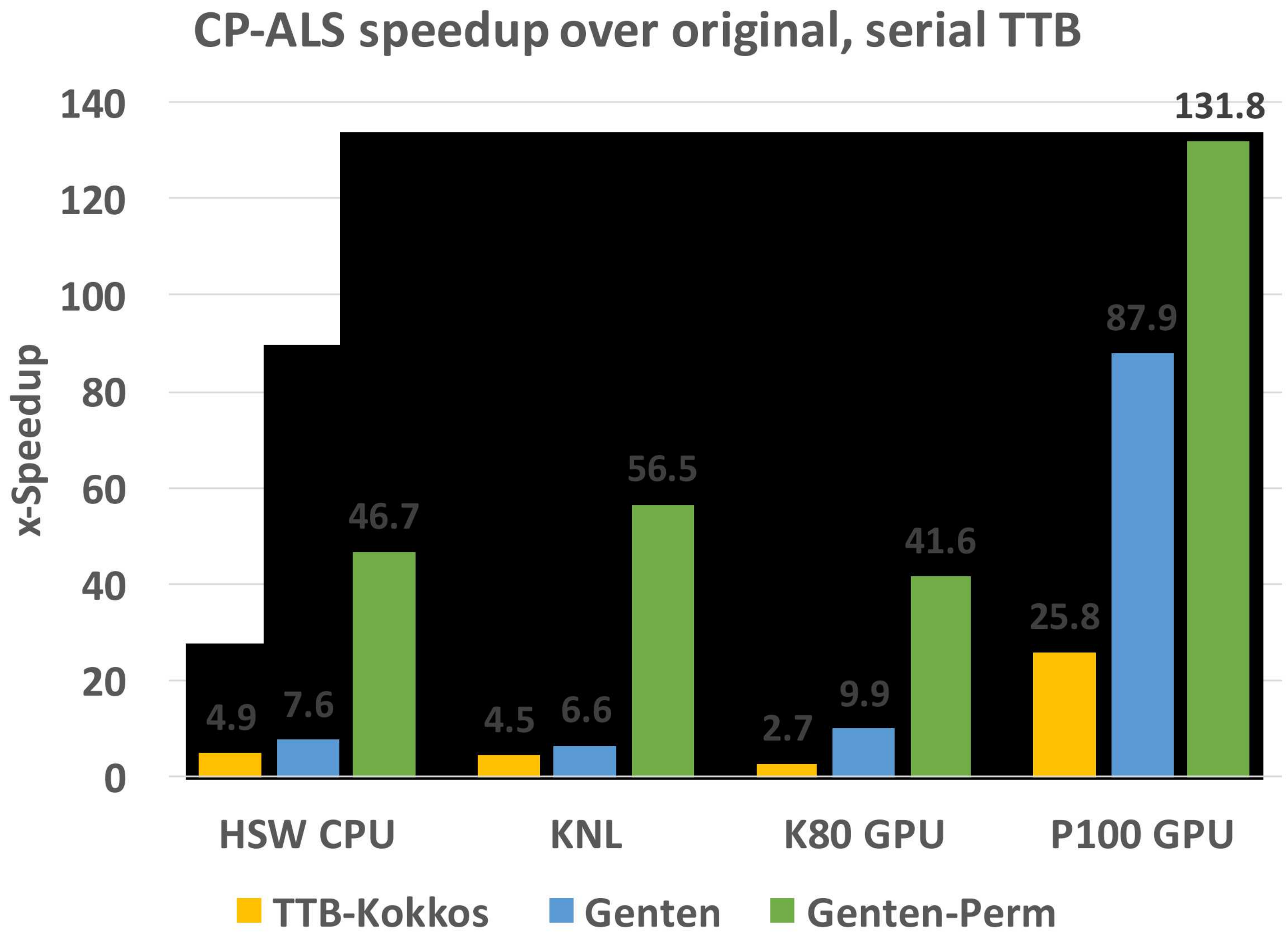
Model: $\mathcal{M} = \sum_r \lambda_r \, \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$

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

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

Hitchcock (1927), Harshman (1970), Carroll and Chang (1970)

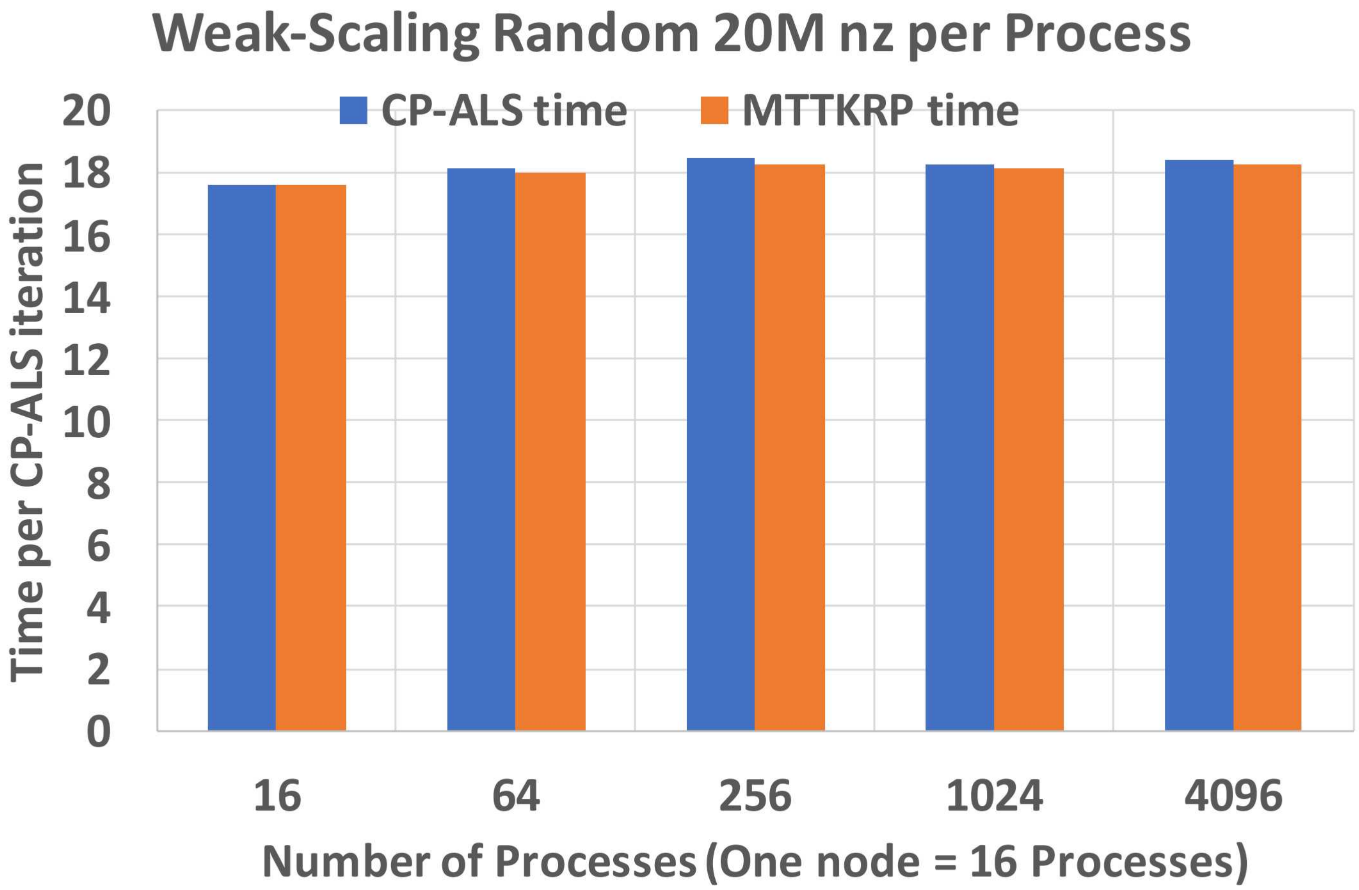
CP-ALS using Kokkos



POC: Eric Phipps (etphipp@sandia.gov)

CP-ALS using Kokkos + Trilinos

- CP-ALS for **huge** sparse tensors in distributed memory
- 1.6TB tensor (82B nonzeros) on 4096 cores



POC: Karen Devine (kddevin@sandia.gov)

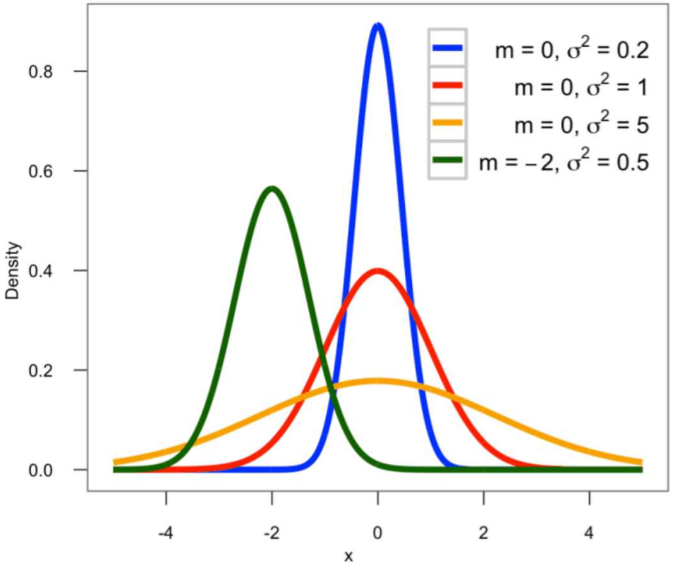
Poisson for Sparse Count Data

Gaussian (typical)

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

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

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

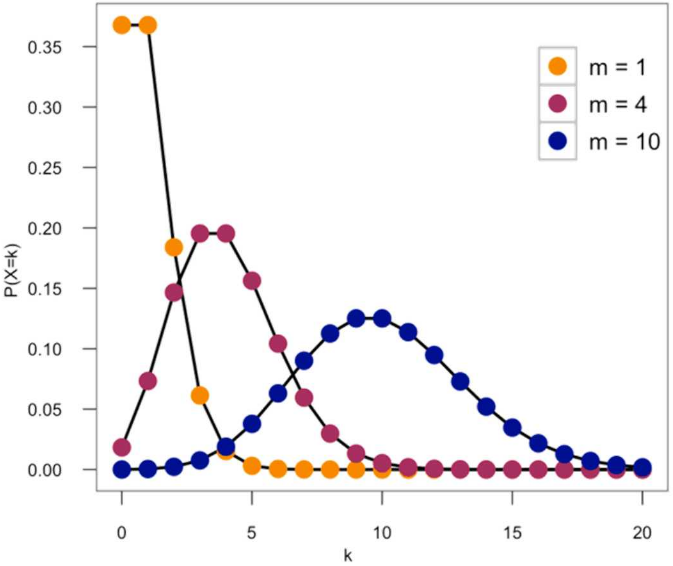


Poisson

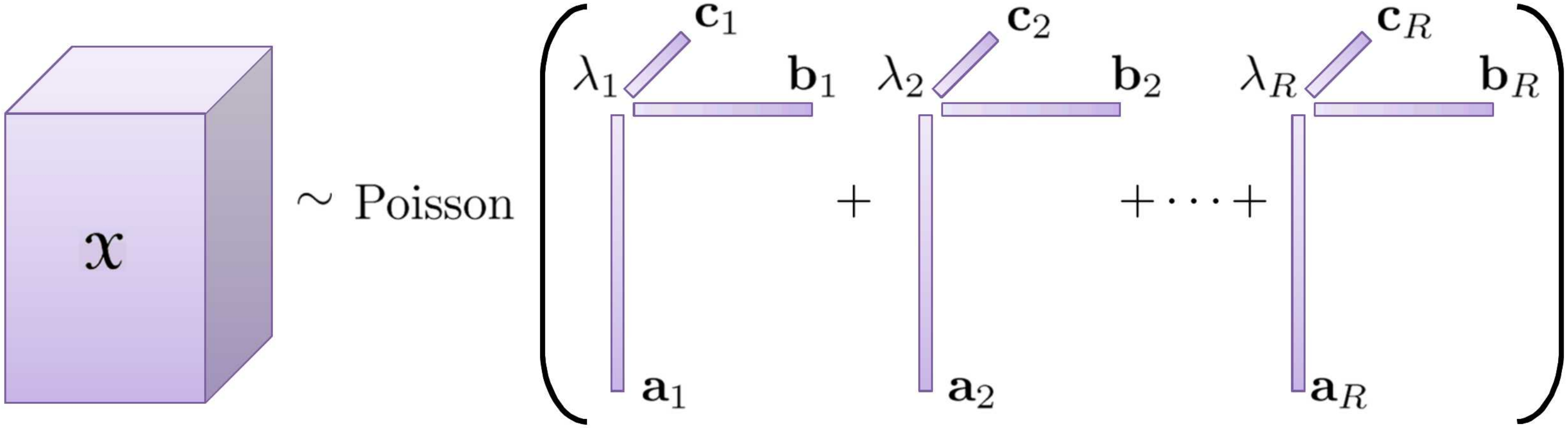
The random variable x is a discrete nonnegative integer.

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

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



Sparse Poisson Tensor Factorization



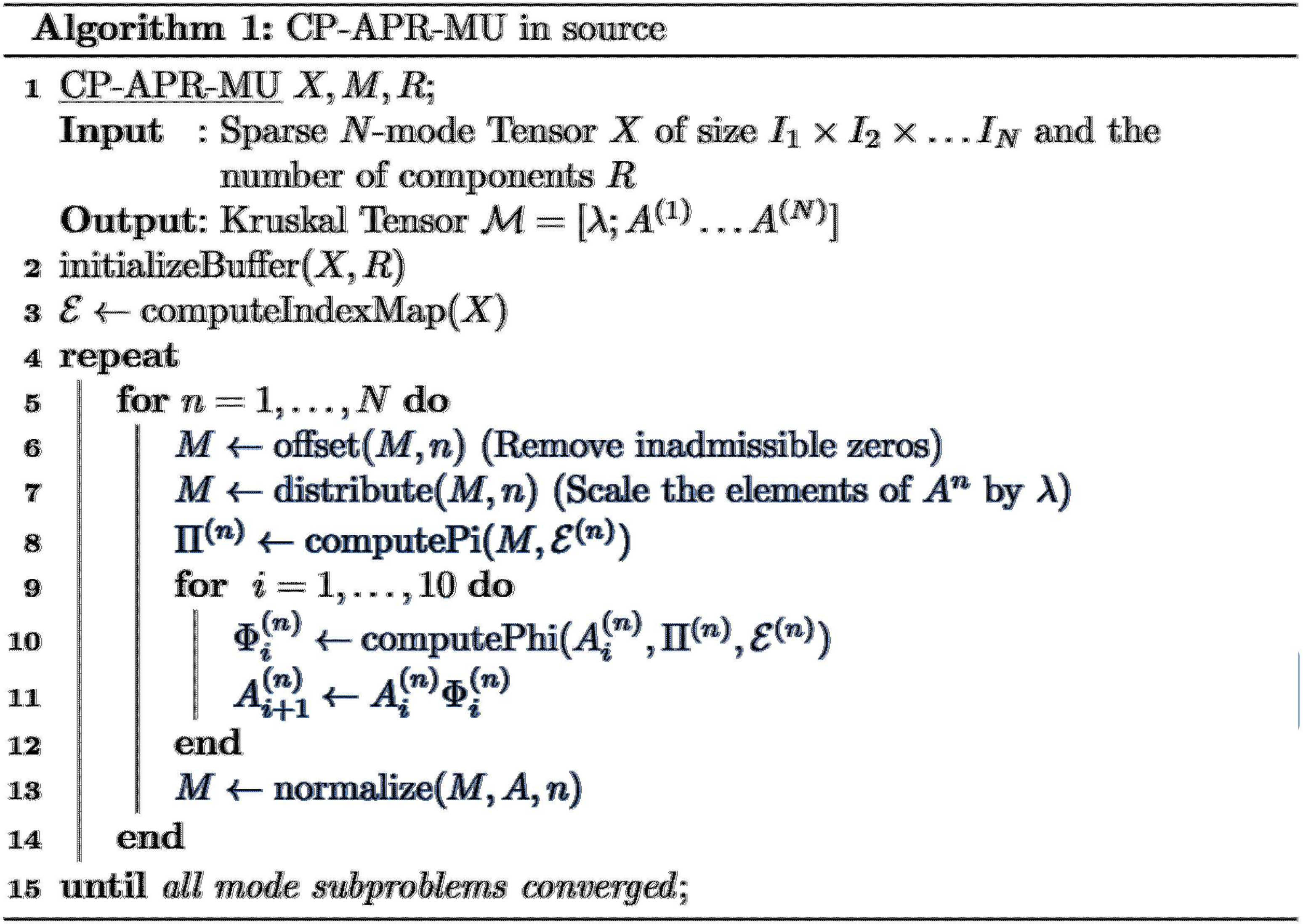
Model: Poisson distribution (nonnegative factorization)

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

- Nonconvex problem!
- Constrained minimization problem (decomposed vectors are non-negative)
- Alternating Poisson Regression (Chi and Kolda, 2011)
  - Assume (d-1) factor matrices are known and solve for the remaining one

- **Multiplicative Updates (CP-APR-MU)** by Chi and Kolda (2011)
- **Damped Newton and Quasi-Newton method for Row-subproblems (CP-APR-PDNR)** by Hansen, Plantenga and Kolda (2014)

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 $\lambda$)
        $\Pi^{(n)} \leftarrow$ 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;
```

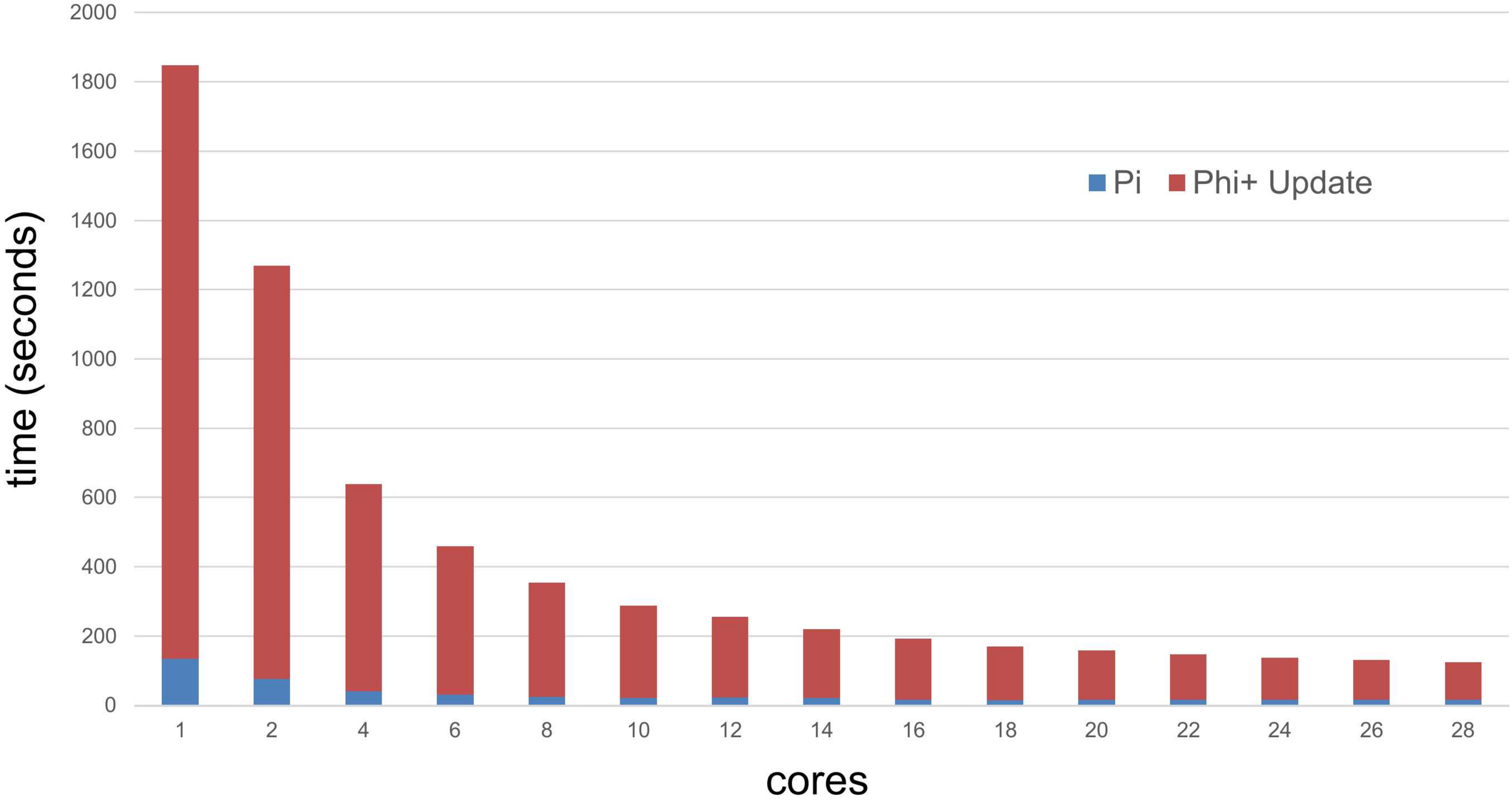
CP-APR-MU Performance Test

- 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

CP-APR-MU on CPU (Random)

CP-APR-MU method, 100 outer-iterations, (3000 x 4000 x 5000, 10M nonzero entries), R=100, 2 Haswell (14 core) CPUs per node, MKL-11.3.3, HyperThreading disabled

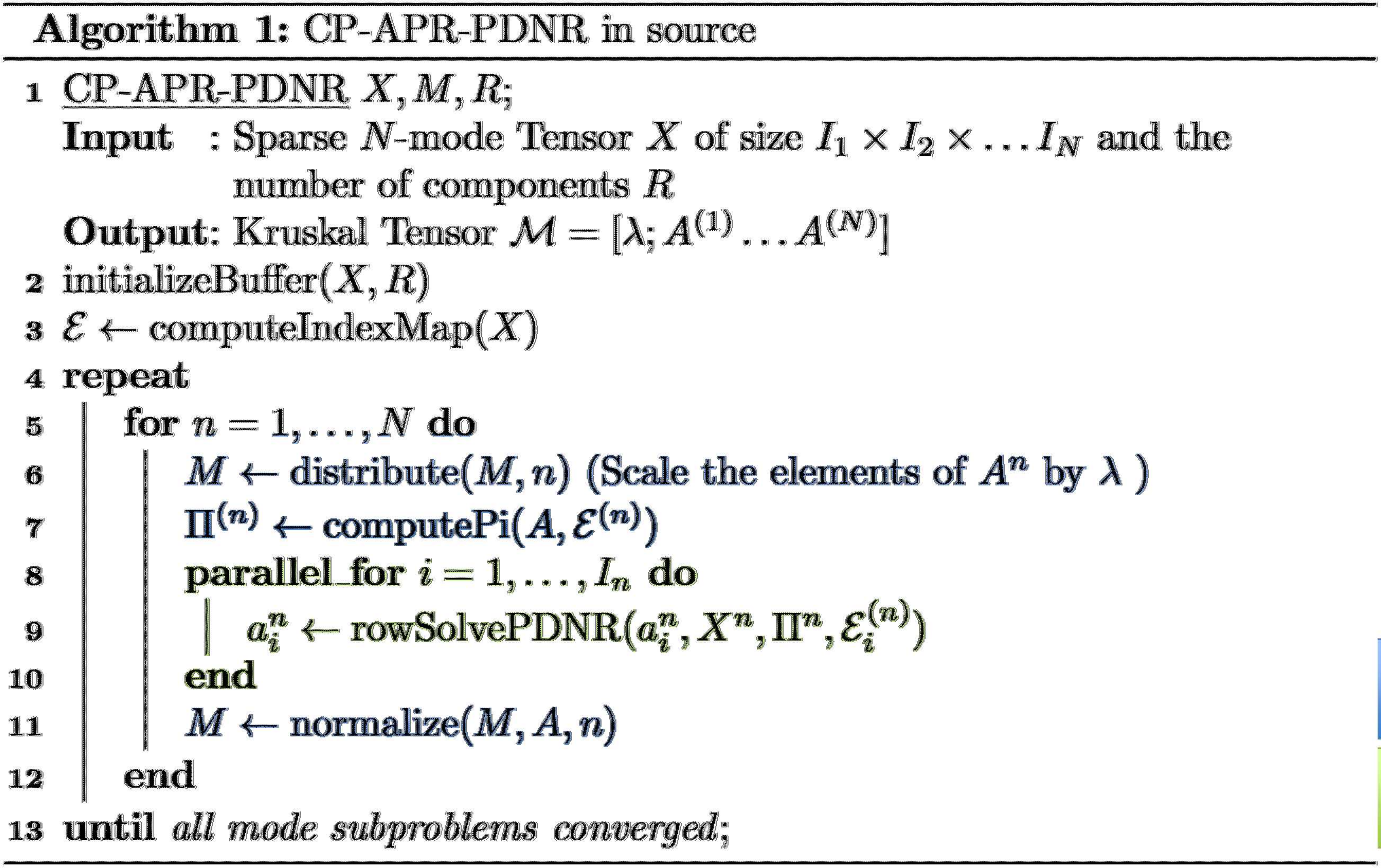


Results: CPAPR-MU Scalability

Data	CPU 1-core		KNL (Cache Mode) 68-core CPU		NVIDIA P100 GPU		NVIDIA V100 GPU	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
Random	1849*	1	84	22.01	44.76	41.31	30.05	**61.53**
LBNL	39	1	33	1.18	2.99	13.04	2.09	**18.66**
NELL-2	1157	1	100	11.02	47.17	24.52	28.80	**40.17**
NELL-1	3365	1	257	10.86				
Delicious	4170	1	3463	1.41				

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

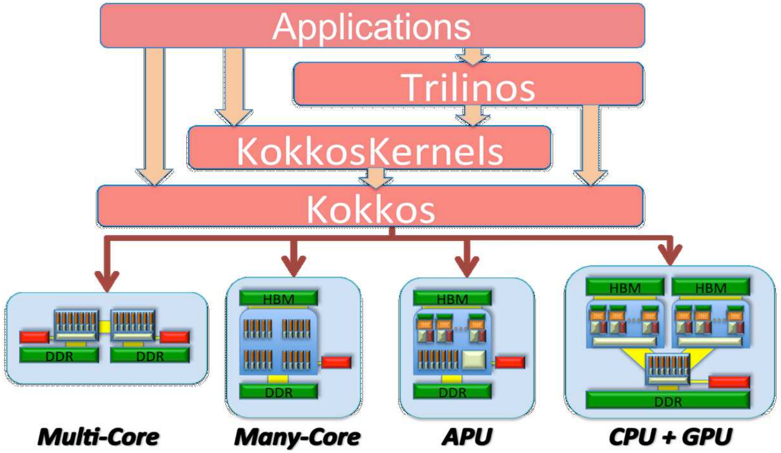
Parallel CP-APR-PDNR



Algorithm 1: CP-APR-PDNR in source

```
CP-APR-PDNR $X, M, R$;
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
        $M \leftarrow$ distribute($M, n$) (Scale the elements of $A^n$ by $\lambda$ )
        $\Pi^{(n)} \leftarrow$ computePi($A, \mathcal{E}^{(n)}$)
        parallel_for $i = 1, \dots, I_n$ 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;
```

Use Case 3: Finding Triangles with Kokkos Kernels for Node-Level Performance



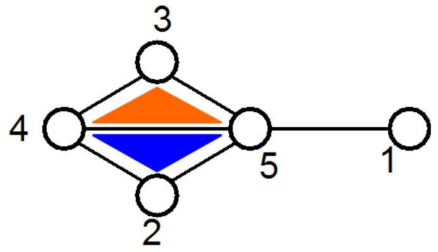
- Kokkos
  - Tools for performance portable node-level parallelism
  - Manages data access patterns, execution spaces, memory spaces
  - Performance portability not trivial for sparse matrix and graph algorithms
- Kokkos Kernels
  - Layer of performance-portable kernels for high performance
  - Sparse/Graph: SpMV, SpGEMM, triangle enumeration

Kokkos Kernels for performance-portable sparse/graph kernels

KokkosKernels and IEEE/DARPA Graph Challenge

KKTri

Linear Algebra Based Triangle Counting



KKMEM: KokkosKernels Matrix-Matrix Multiply

SpGEMM

$$\begin{array}{|c|c|c|c|}\hline * & & * & * \\ \hline * & & * & * \\ \hline & * & & \\ \hline * & & & * \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline * & * & & * \\ \hline & * & & \\ \hline & & * & \\ \hline * & & & * \\ \hline \end{array} \bullet \begin{array}{|c|c|c|c|}\hline * & & & \\ \hline * & & * & * \\ \hline & * & & \\ \hline * & & & * \\ \hline \end{array}$$

C A B

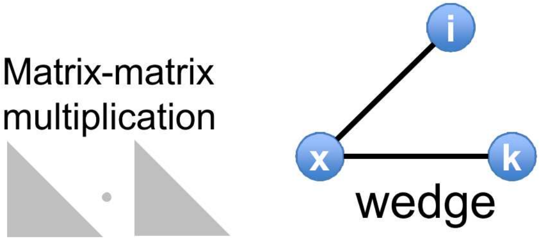
- 2017 IEEE/DARPA Graph Challenge Submission
  - Wolf, Deveci, Berry, Hammond, Rajamanickam: “Fast Linear Algebra-Based Triangle Counting with KokkosKernels.”
  - **Triangle Counting Champion** (focus: single node)
  - Counted 34.8B triangles in 1.2B edge graph in 43 secs (Twitter2010)

Vision: Build software on top of highly optimized KokkosKernels kernels (e.g., KKTri) to impact applications

Linear Algebra-Based Triangle Counting

C=L*L

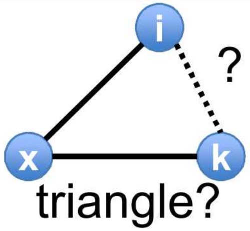
Matrix-matrix multiplication



- C(i,k) = # of wedges with endpoints i,k

D=C.*L

Element-wise multiplication



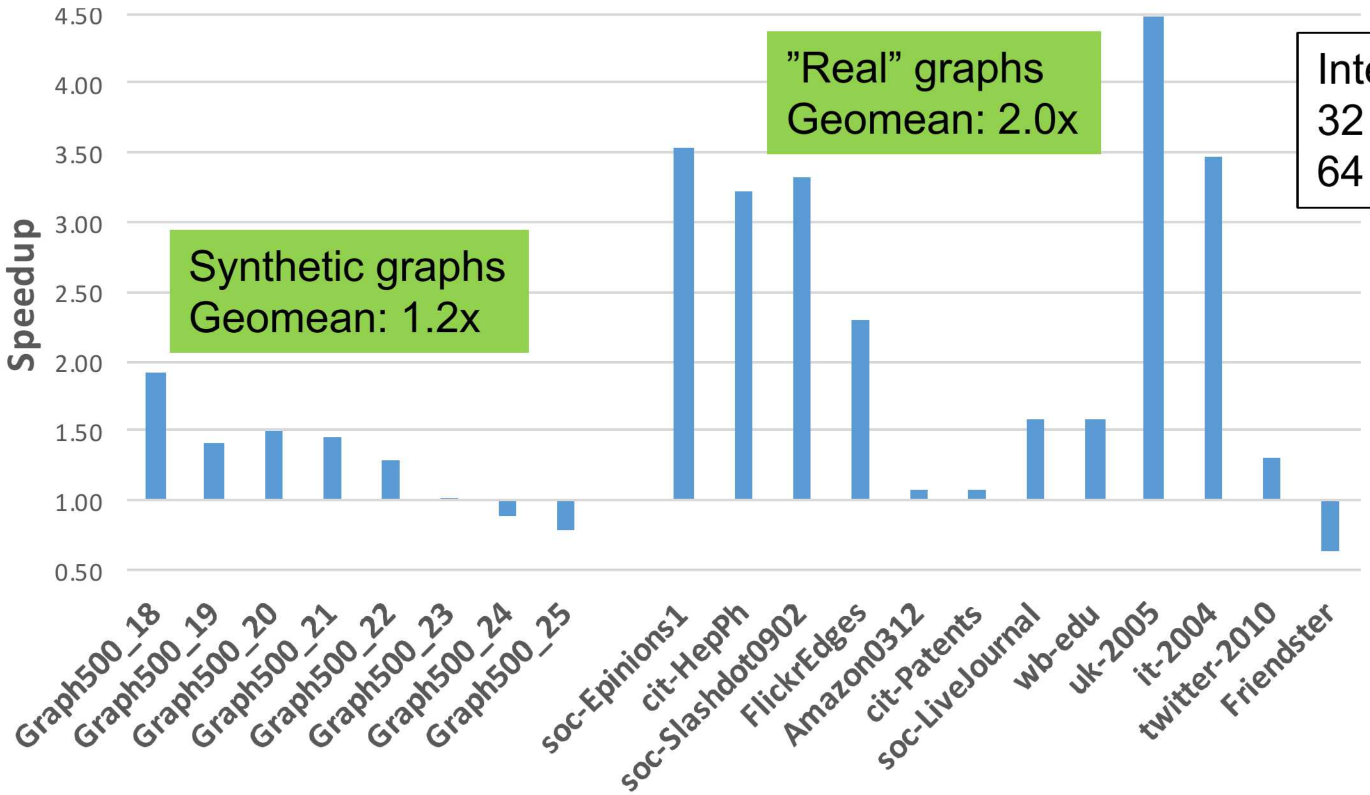
- D(i,k) = # of triangles with vertices i,k
- Filters out wedges i-x-k when edge i,k doesn't exist

- New linear algebra-based triangle counting method
  - Uses lower triangle part of adjacency matrix, L
  - Method: (L*L).*L)
  - "Visits" each triangle/wedge once
- Once triangle is "visited," Kokkos functor used to count triangles
  - Other operations can be performed on each triangle

Kokkos Functor enables "Visitor Pattern," which can add more flexibility to linear algebra approach

KKTri Speedup Relative to State of the Art

Comparison with Best Ligra Method (Shun, Tangwongsan, 2015)



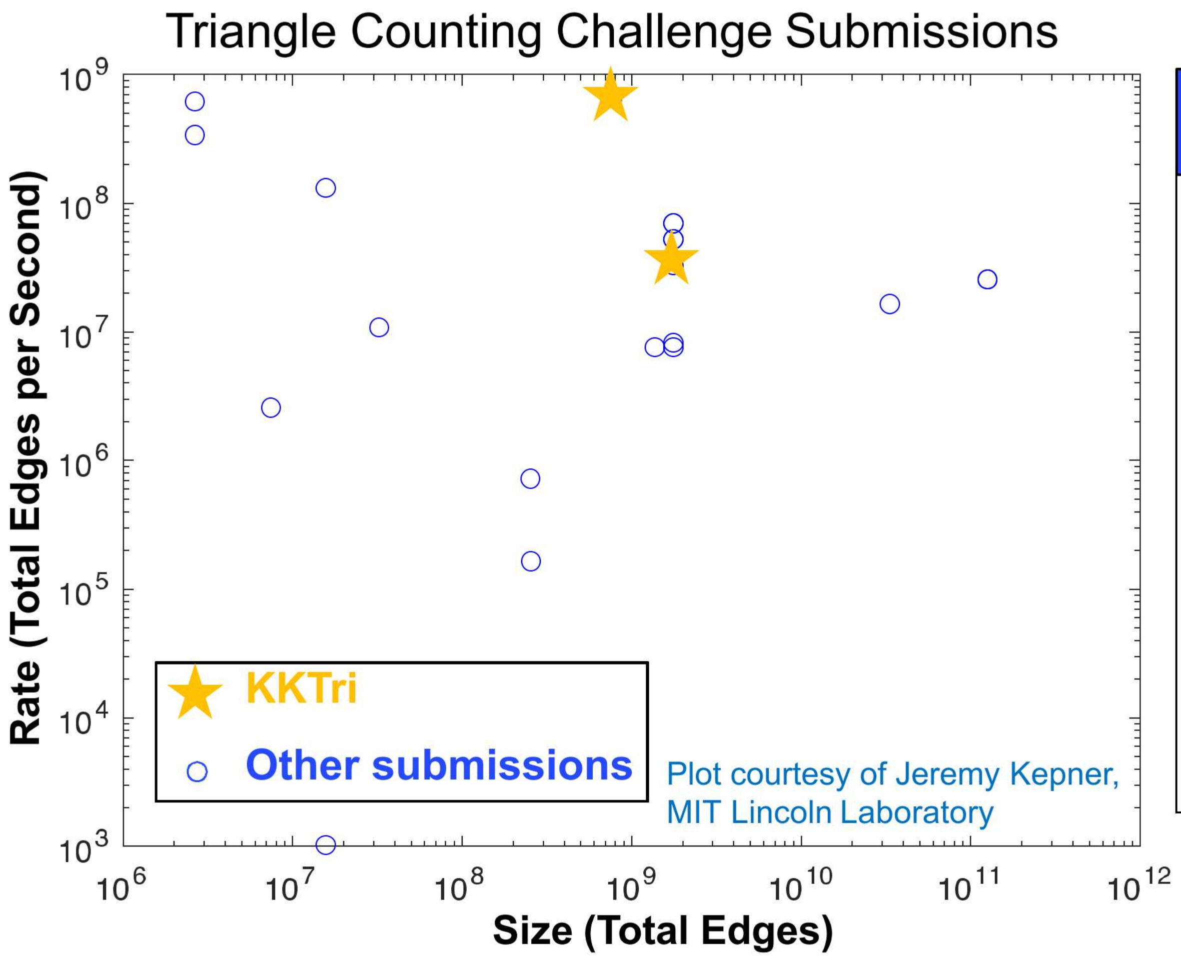
Synthetic graphs
Geomean: 1.2x

”Real” graphs
Geomean: 2.0x

Intel Haswell
32 cores
64 threads

KKTri’s linear algebra-based triangle counting outperforms state-of-the-art graph-centric method

Graph Challenge: Lessons Learned



Plot courtesy of Jeremy Kepner, MIT Lincoln Laboratory

Lessons Learned

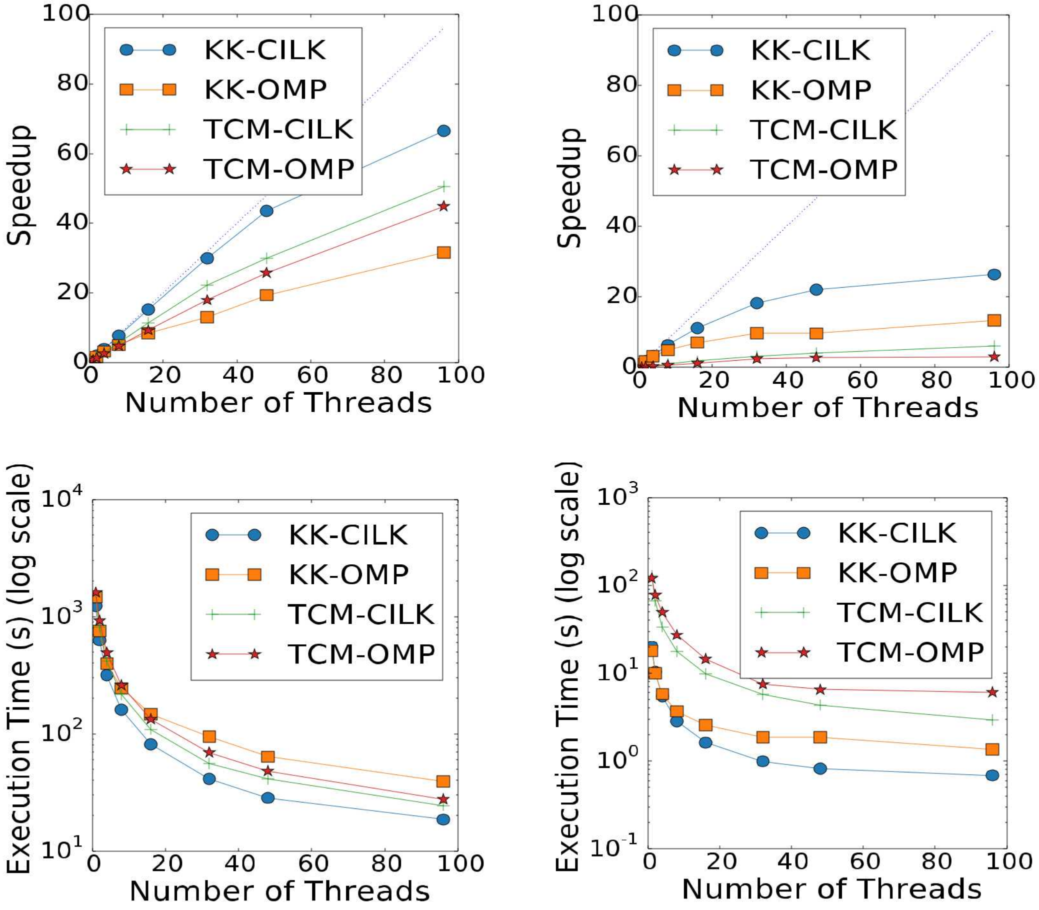
- Linear algebra approach can be competitive
- Avoiding unnecessary computation essential
- Data compression often helps performance
- Visitor pattern can add more flexibility to linear algebra approach

Linear algebra-based KKTri as good as or better than other state-of-the-art methods

GraphChallenge 2018

- Kokkos Kernels-based triangle counting KKTri-Cilk
  - Replaced Kokkos/OpenMP with Cilk
  - Demonstrated improved usage of hyperthreading
  - Faster than Kokkos/OpenMP implementation on 63 of 78 instances
- Example of HPDA driving Kokkos development
  - Focus on improving hyperthread usage; Cilk backend
- KKTri-Cilk surpasses 10^9 for the rate measure on a single multicore node.
- KKTri-Cilk is also faster than state-of-the-art graph library-based implementation on a single multicore node (up to 7x)
- 2017 IEEE/DARPA Graph Challenge Submission

Strong Scaling Experiments


Friendster

UK-2005

KKTri-Cilk scales the best in both problems

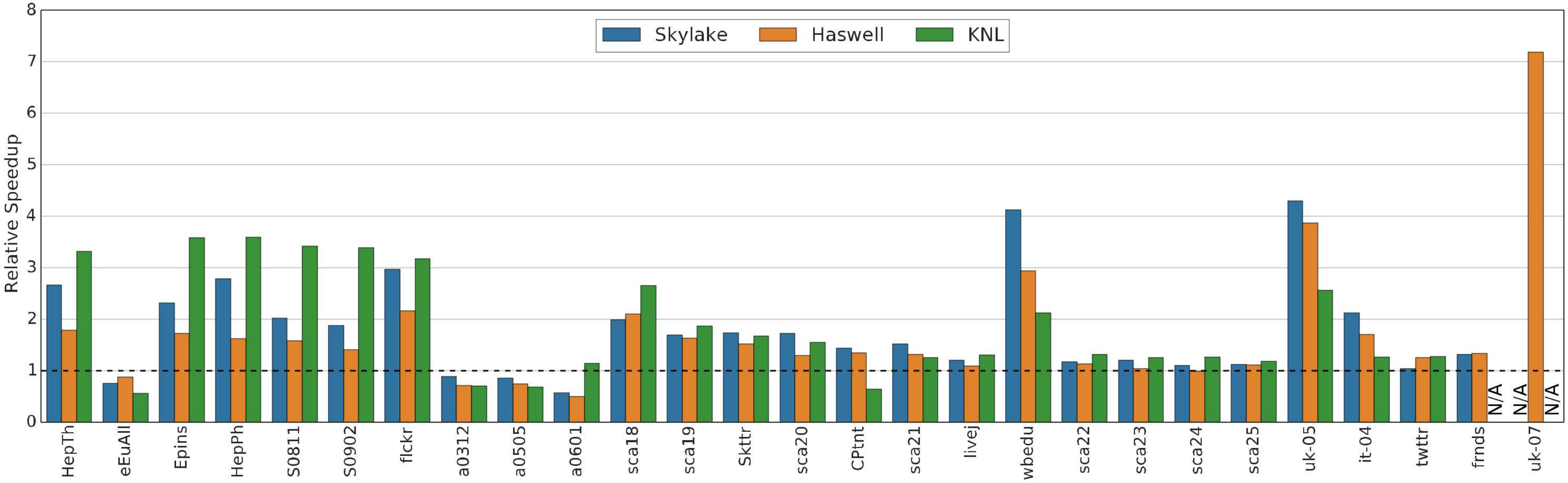
uk-2005 graph has a very good ordering: highly local computations (best rate).

Friendster graph's distribution is in between (best scalability).

Scaling is with respect to the best sequential execution time.

Slide credit: Siva Rajamanickam

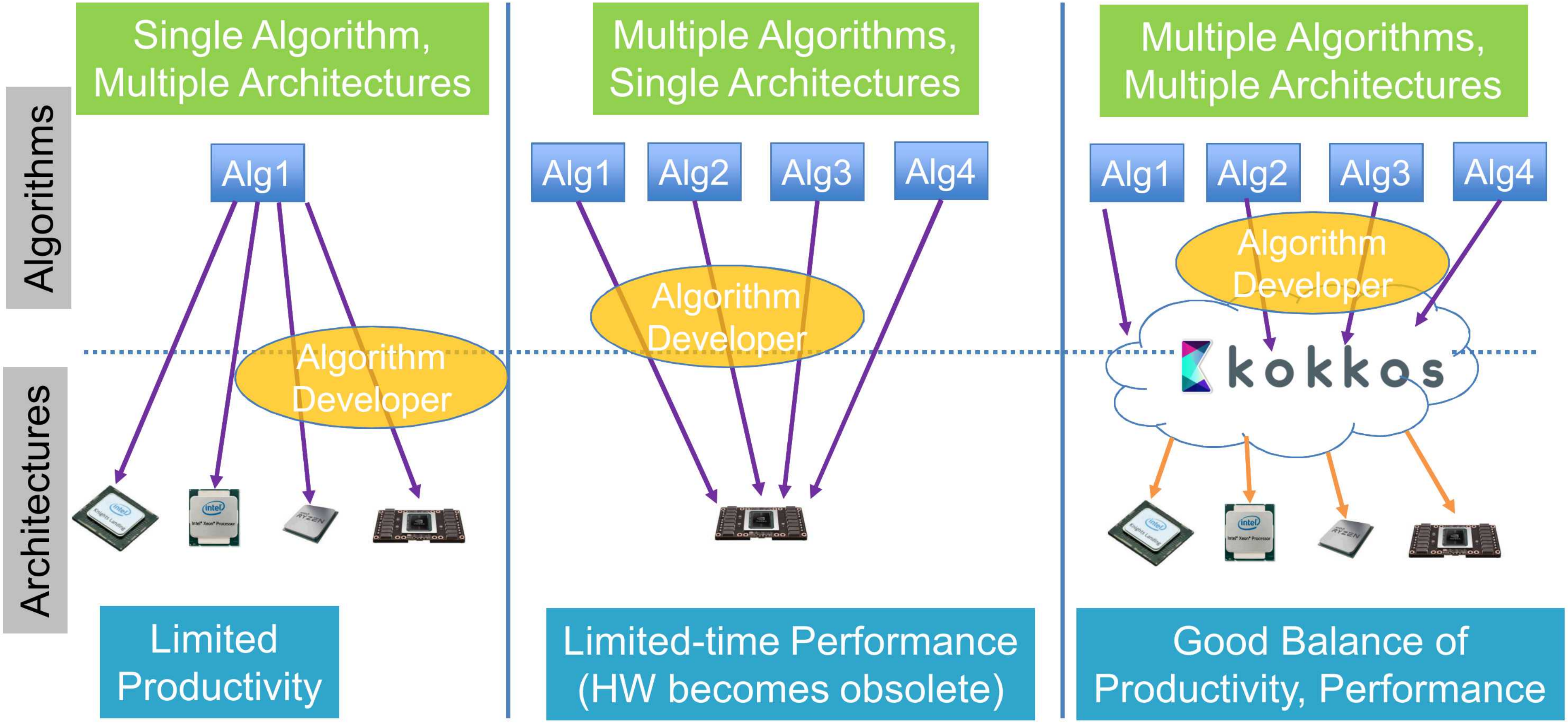
Relative Speedup Experiments



- Comparisons of KKTri-Cilk with TCM, a state-of-the-art graph library [Shun *et al.*]
- Relative speedup in 3 architectures compared to TCM
- KKTri outperforms TCM in 23 of 27 cases
- KKTri can achieve up to 7x speedup on graphs that have a good natural ordering such as wb-edu, uk-2005, and uk-2007

Slide credit: Siva Rajamanickam

Summary


Algorithms
Architectures
Single Algorithm, Multiple Architectures
Alg1
Algorithm Developer
Limited Productivity
Multiple Algorithms, Single Architectures
Alg1
Alg2
Alg3
Alg4
Algorithm Developer
Limited-time Performance (HW becomes obsolete)
Multiple Algorithms, Multiple Architectures
Alg1
Alg2
Alg3
Alg4
Algorithm Developer
kokkos
Good Balance of Productivity, Performance

- **Performance-Portable Kokkos enables productivity of algorithm developer and performance on several architectures**
- **3 Use Case: Grafiki, tensors, Kokkos Kernels triangle counting**

Conclusions

- Results show promise of Kokkos for HPDA
- Improvements to Kokkos (based on HPDA experience) will yield additional performance improvements
- **More work ahead**
  - Algorithms, optimized kernels, integration, architectures, …
  - Machine learning – ECP ExaLearn Co-Design Center
- **Much software is available**
  - Kokkos: https://github.com/kokkos/kokkos
  - Kokkos Kernels: https://github.com/kokkos/kokkos-kernels
  - SparTen: https://gitlab.com/tensors/sparten
  - GenTen: https://gitlab.com/tensors/genten
  - Triangle Counting: https://github.com/Mantevo/miniTri
  - Coming soon: Grafiki

Thank you

- Contact: mmwolf@sandia.gov

Extra

Kokkos

Kokkos – What about Alternatives?

- RAJA: Closest thing to Kokkos (if you include CHAI and Umpire)
  - More inward focused on LLNL apps
  - Used to be a couple years behind in basic performance portability capabilities: now in pretty good shape (lead of Coral2 procurement helped with vendor attention)
  - Nothing like KokkosKernels though.
- OpenMP 4.5: In Theory Vendor Supported
  - We need at least OpenMP 5.0 to have a chance with our apps
  - IBM Compiler is just about where we may have a chance to compile our codes
  - Compile time with IBM is atrocious though: ~5 days for Trilinos + App just for P9
  - No support for virtual functions: required by ASC production apps
  - NVIDIA doesn’t provide an OpenMP 4.5 compiler

Vendor Collaborations

- AMD: Strong engagement on Kokkos backend for AMD GPUs
  - AMD staff visited Sandia for multi day coding session
  - Appended PathForward F2F meeting with extra day for Kokkos discussions
- NVIDIA: Collaboration on C++ Proposals and Early Evaluation of NVSHMEM
  - Working on C++ executors, making sure they are usable for HPC
  - Implemented NVSHMEM backend for Kokkos Remote Memory Spaces
- ARM: Preparation for ARM HPC deployments
  - Helping to stabilize the software stack, find issues with compilers etc.
  - ARM developers participated in UK Kokkos training
- Intel: Working on ECP PathForward Architecture Backend for Kokkos

HPPDA and Kokkos

- **Performance-Portable Computing (PPC)**
  - ***Sandia: Kokkos***
- **High-Performance Data Analytics (HPDA)**
  - Use HPC to do big data analytics faster
  - Apply DOE HPC investment to analytics
- **High-Performance-Portable Data Analytics (HPPDA)**
  - Use PPC to enable HPDA on DOE platforms
  - ***Kokkos is key to Sandia work***
- **Strategically plan PPC/HPPDA**
  - Consider HPPDA an "app", represent on Kokkos-core
  - Create feedback between HPC/PPC and HPDA
  - Kokkos improvements yield better performing HPDA applications

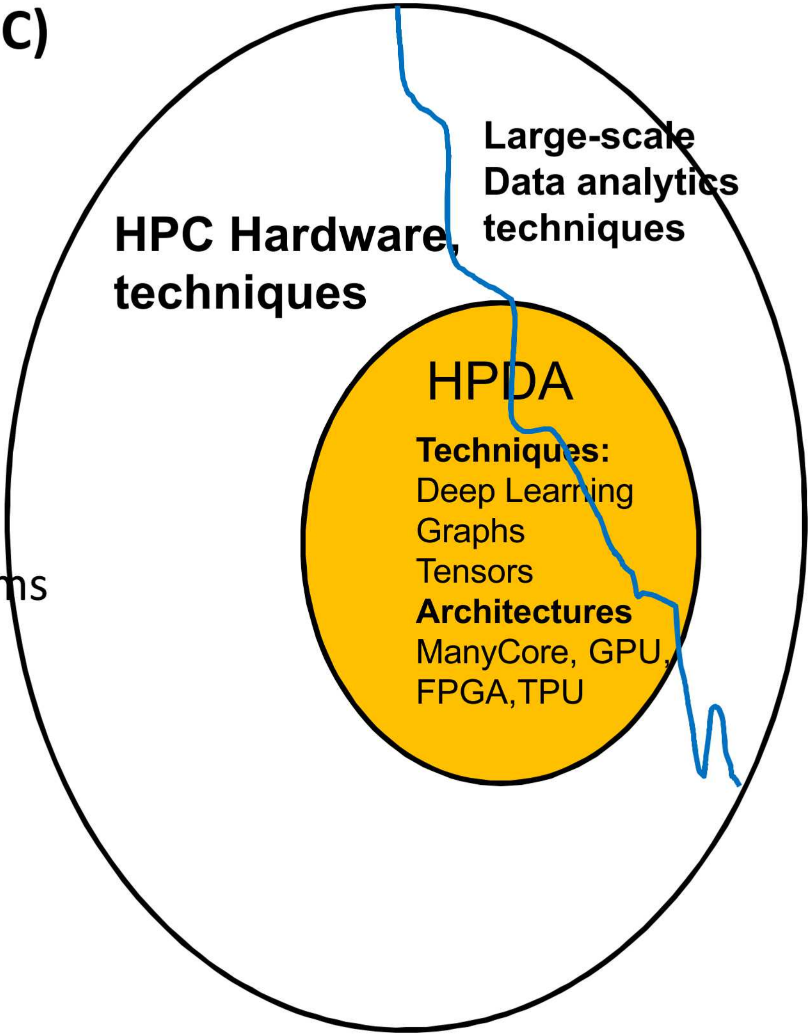
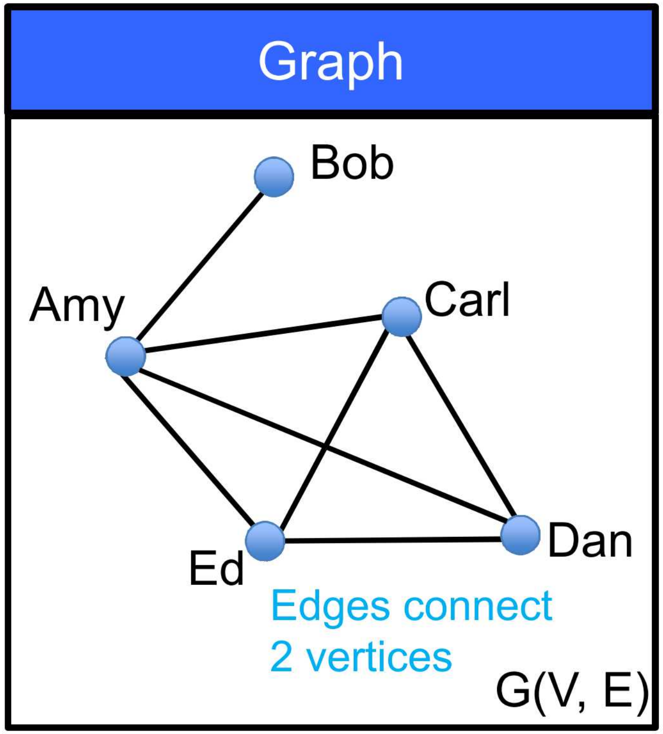


Image credit: Jon Berry

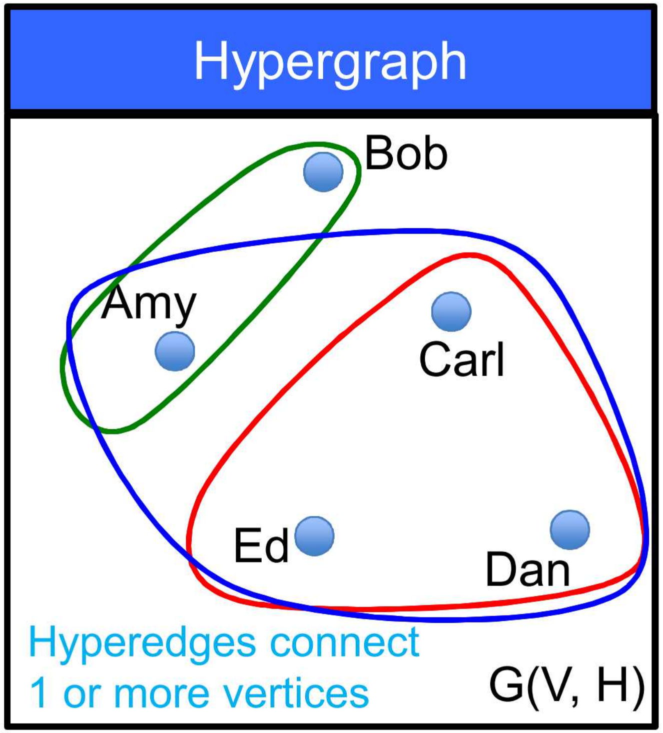
Grafiki

Hypergraphs



Graph

Edges connect 2 vertices



Hypergraph

Hyperedges connect 1 or more vertices

- Generalization of graph
  - Hyperedges represent multiway relationships between vertices
  - Hyperedge – set of 1 or more vertices
  - Key feature: hyperedges can connect more than 2 vertices

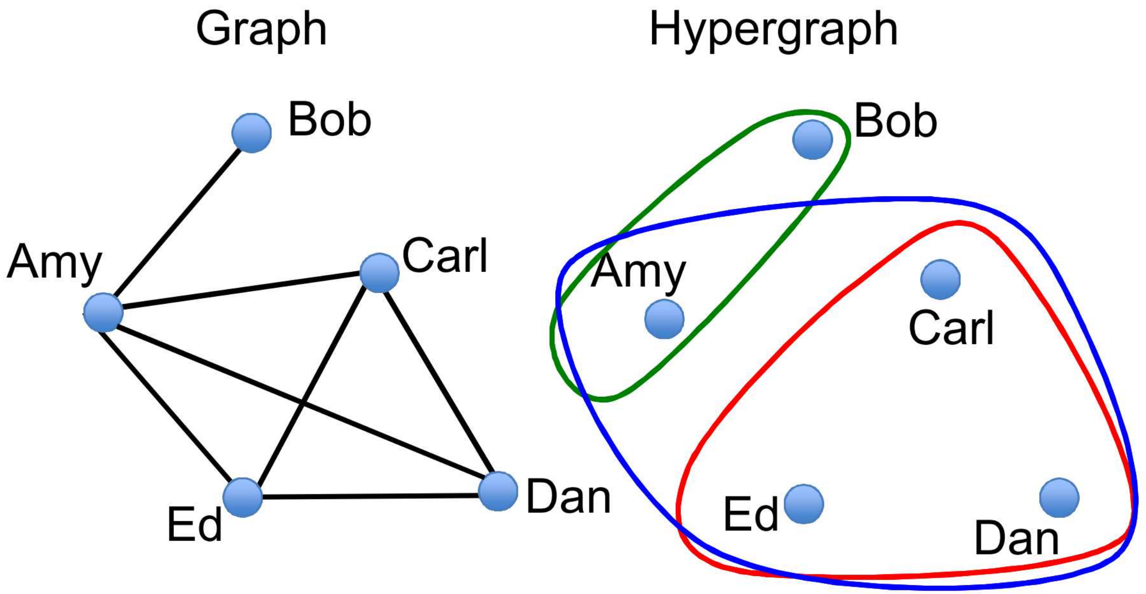
Why Hypergraphs?

Emails

Users

	1	2	3
Amy	x		x
Bob	x		
Carl		x	x
Dan		x	x
Ed		x	x

Relational Data



- Convenient representation of relational data
  - E.g., Each email represented by hyperedge (a subset of users)
- Multiway relationships can be represented nonambiguously
- Computation and storage advantages

Incidence Matrices

1		1
1		
	1	1
	1	1
	1	1

Hypergraph incidence matrix

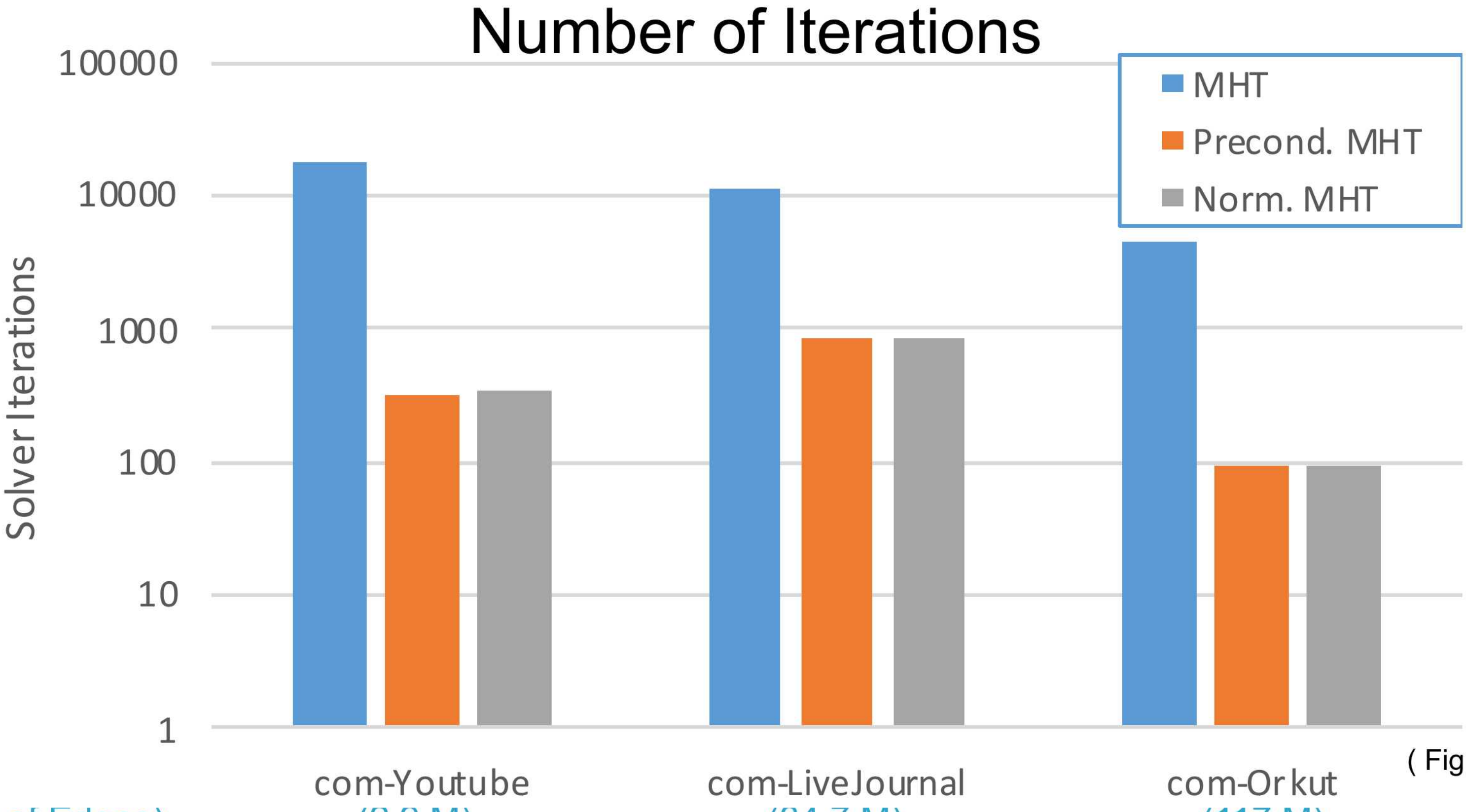
1				1	1	1			
1									
	1	1		1			1	1	
	1		1		1		1		1
		1	1			1		1	1

Graph Incidence matrix

- Compute with **hypergraph incidence matrices** when possible
  - Relational data is often stored as hypergraph incidence matrix*
  - Avoids costly SpGEMM operation for building adjacency matrices
  - Dynamic data: easier to update incidence matrices than adjacency matrices
  - Trilinos solver operators make this easy
- Hypergraphs require significantly **less storage space** and **fewer operations** than graphs generated using clique expansion

*Kepner, Jeremy, et al. "Dynamic distributed dimensional data model (D4M) database and computation system." *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012.

TriData Mean Hitting Time (MHT) Results



(Number of Edges) (2.9 M) (34.7 M) (117 M)

(Figure is U)

MHT for largest connected component
Original graphs from: snap.stanford.edu

Preconditioning and normalization greatly improves convergence (up to 50x reduction in number of iterations)

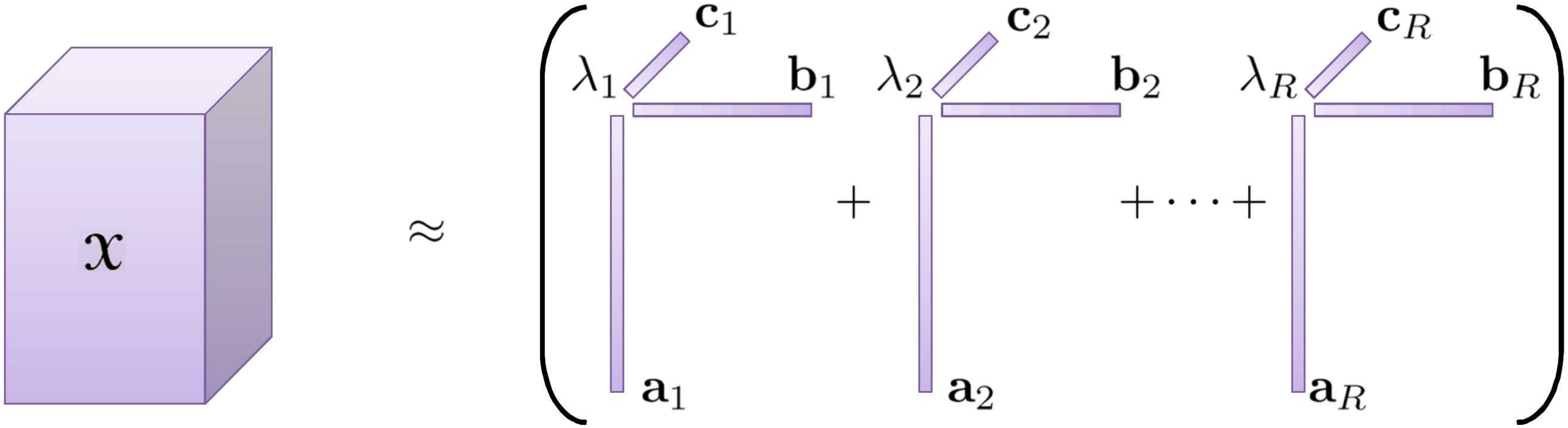
Tensors

Use Case 2: Scalable Tensor Factorizations

- **Motivation: Count Data**
  - Network analysis
  - Term-document analysis
  - Email analysis
  - Link prediction
  - Web page analysis
- **Large, Sparse Data**
  - Number of dimensions = 4, 5, 6, …
  - Example tensor size: 10^4 x 10^4 x 10^6 x 10^6 x 10^7
  - Example densities: 10^{-8} to 10^{-16}
- Targeting several multi/many-core architectures
  - Intel CPU, Intel MIC, NVIDIA GPU, IBM Power 9, etc.

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

Hitchcock (1927), Harshman (1970), Carroll and Chang (1970)

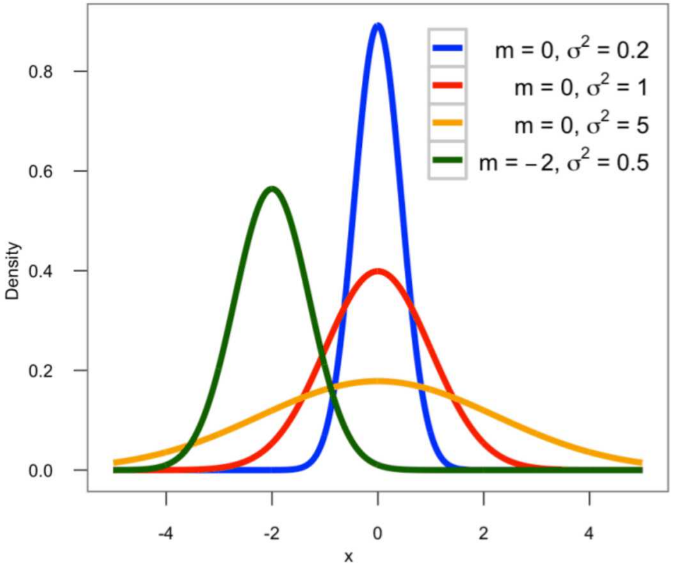
Poisson for Sparse Count Data

Gaussian (typical)

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

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

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

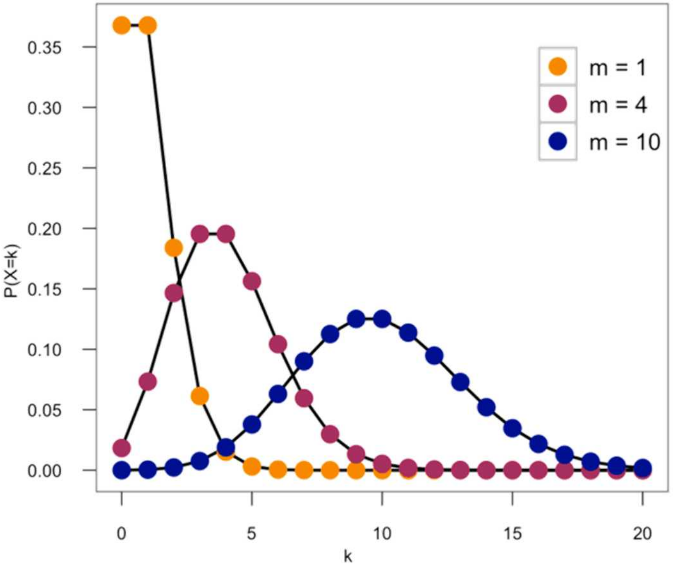


Poisson

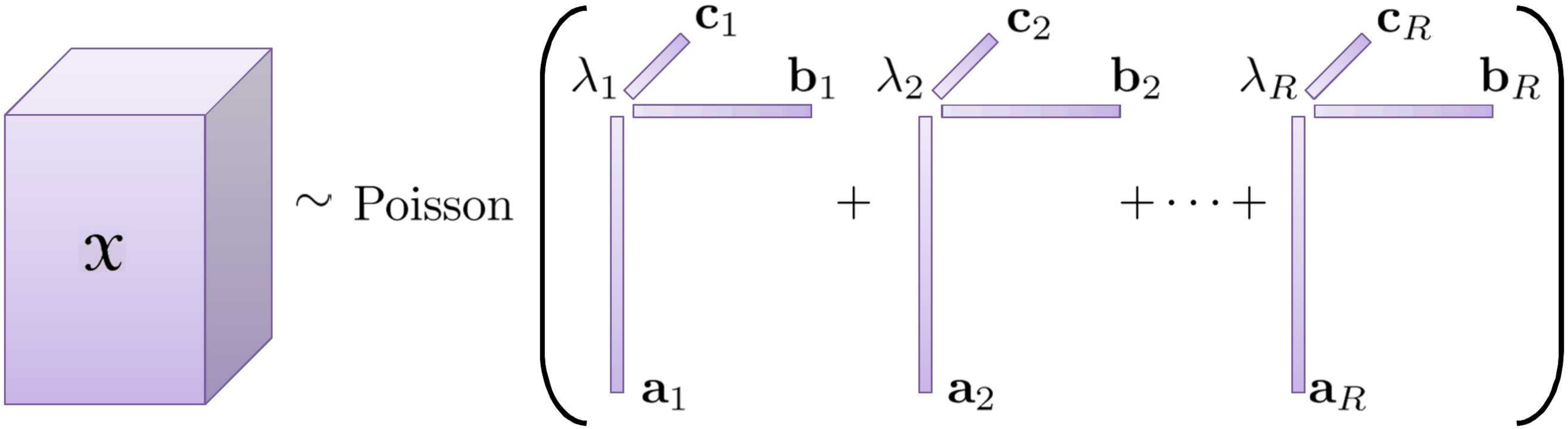
The random variable x is a discrete nonnegative integer.

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

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



Sparse Poisson Tensor Factorization


$\mathcal{X}$
$\sim$ Poisson
λ_1 $\mathbf{c}_1$ $\mathbf{b}_1$ $\mathbf{a}_1$
+
λ_2 $\mathbf{c}_2$ $\mathbf{b}_2$ $\mathbf{a}_2$
$+\cdots+$
λ_R $\mathbf{c}_R$ $\mathbf{b}_R$ $\mathbf{a}_R$

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 constraints
  - 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…

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

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

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

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

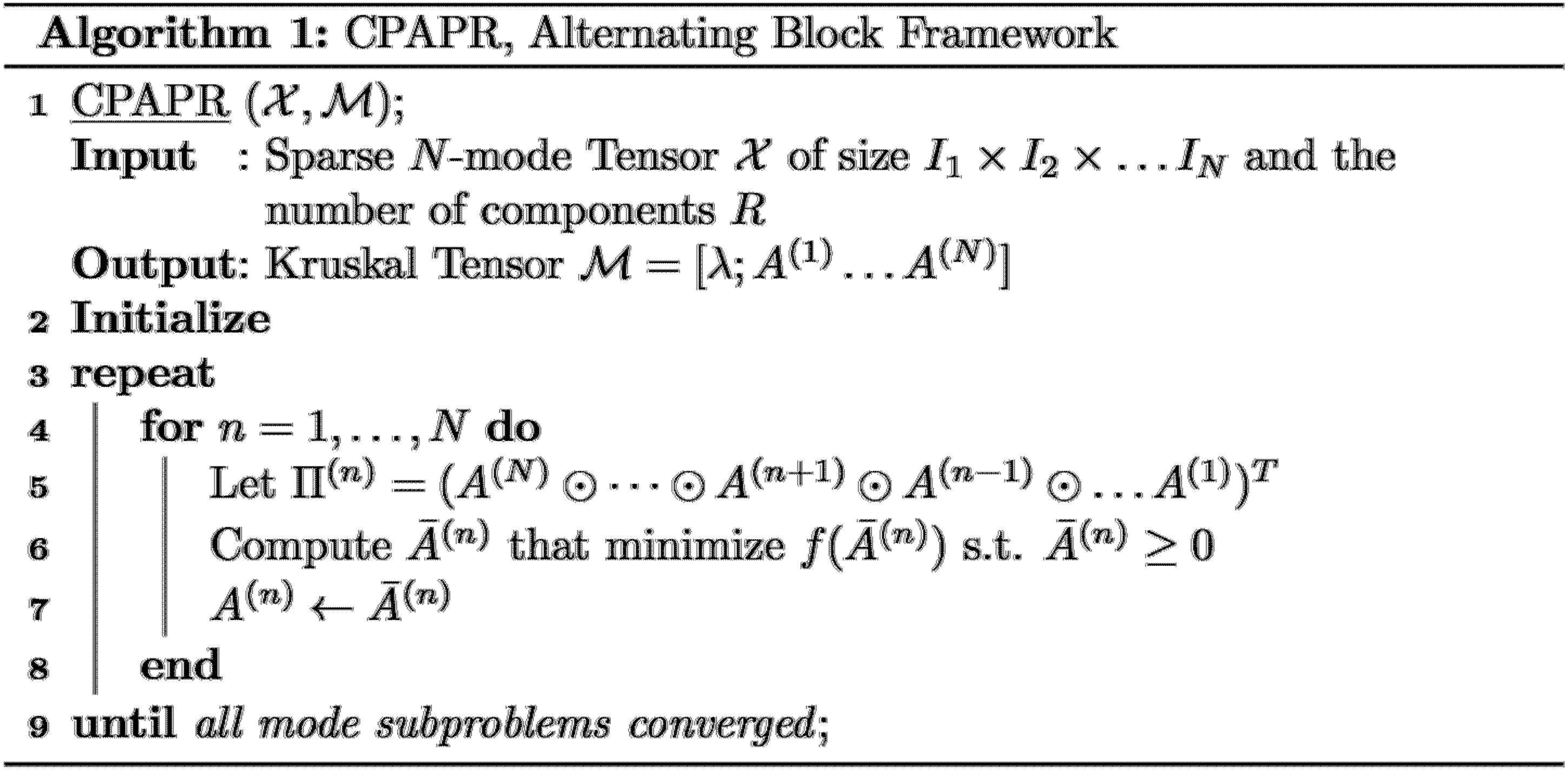
$$\left.\begin{aligned} &5.\ \ \bar{\mathbf{C}} \leftarrow \arg\min_{\bar{\mathbf{C}}\geq 0} \sum_{ijk} m_{ijk} - x_{ijk} \log m_{ijk} \ \text{ s.t. } \mathcal{M} = \sum_r \mathbf{a}_r \circ \mathbf{b}_r \circ \bar{\mathbf{c}}_r \\ &6.\ \ \boldsymbol{\lambda} \leftarrow \mathbf{e}^{\mathsf{T}} \bar{\mathbf{C}};\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

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



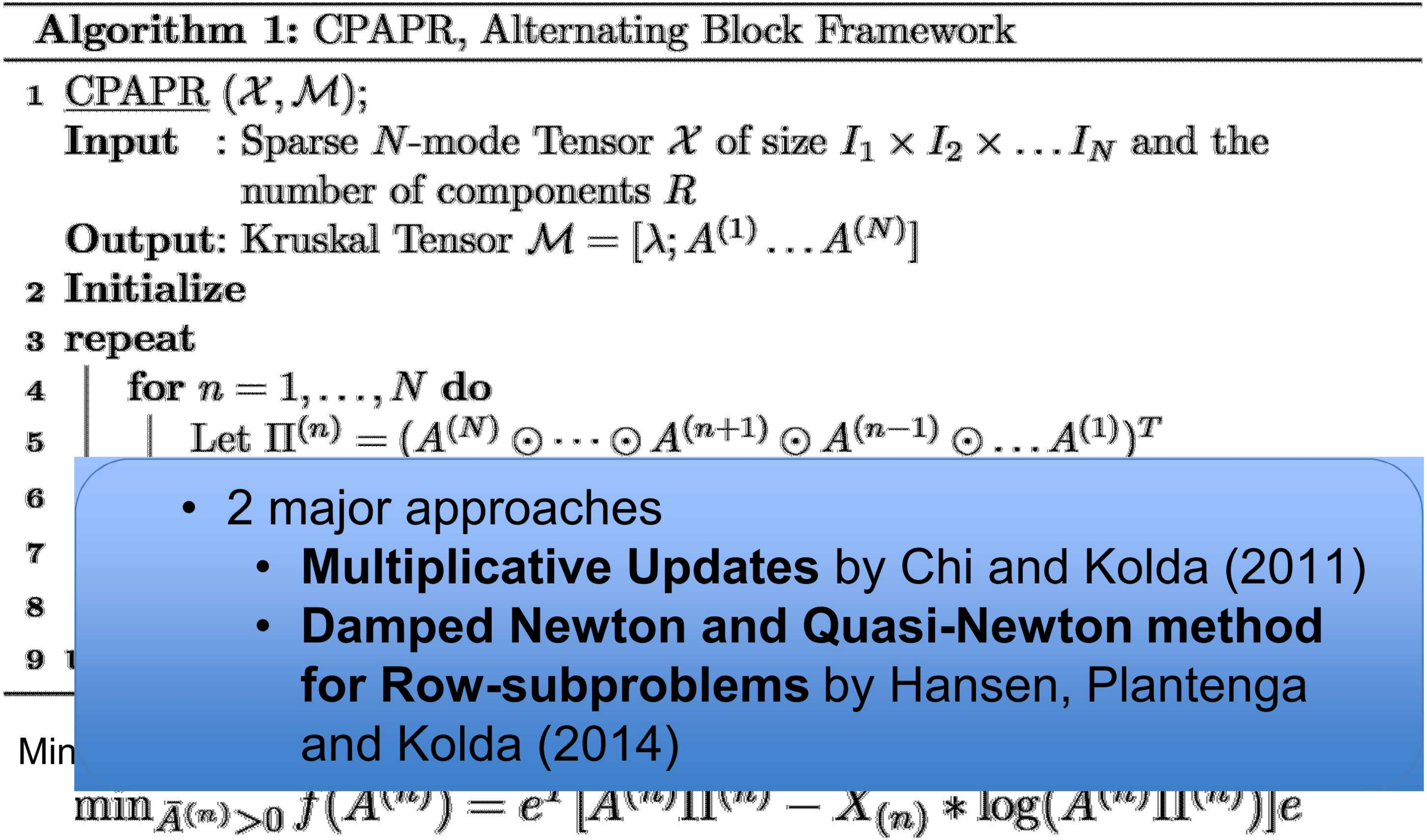
Algorithm 1: CPAPR, Alternating Block Framework

```
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:

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

CP-APR



Algorithm 1: CPAPR, Alternating Block Framework

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

Min

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

- 2 major approaches
  - **Multiplicative Updates** by Chi and Kolda (2011)
  - **Damped Newton and Quasi-Newton method for Row-subproblems** by Hansen, Plantenga and Kolda (2014)

Key Elements of MU and PDNR methods

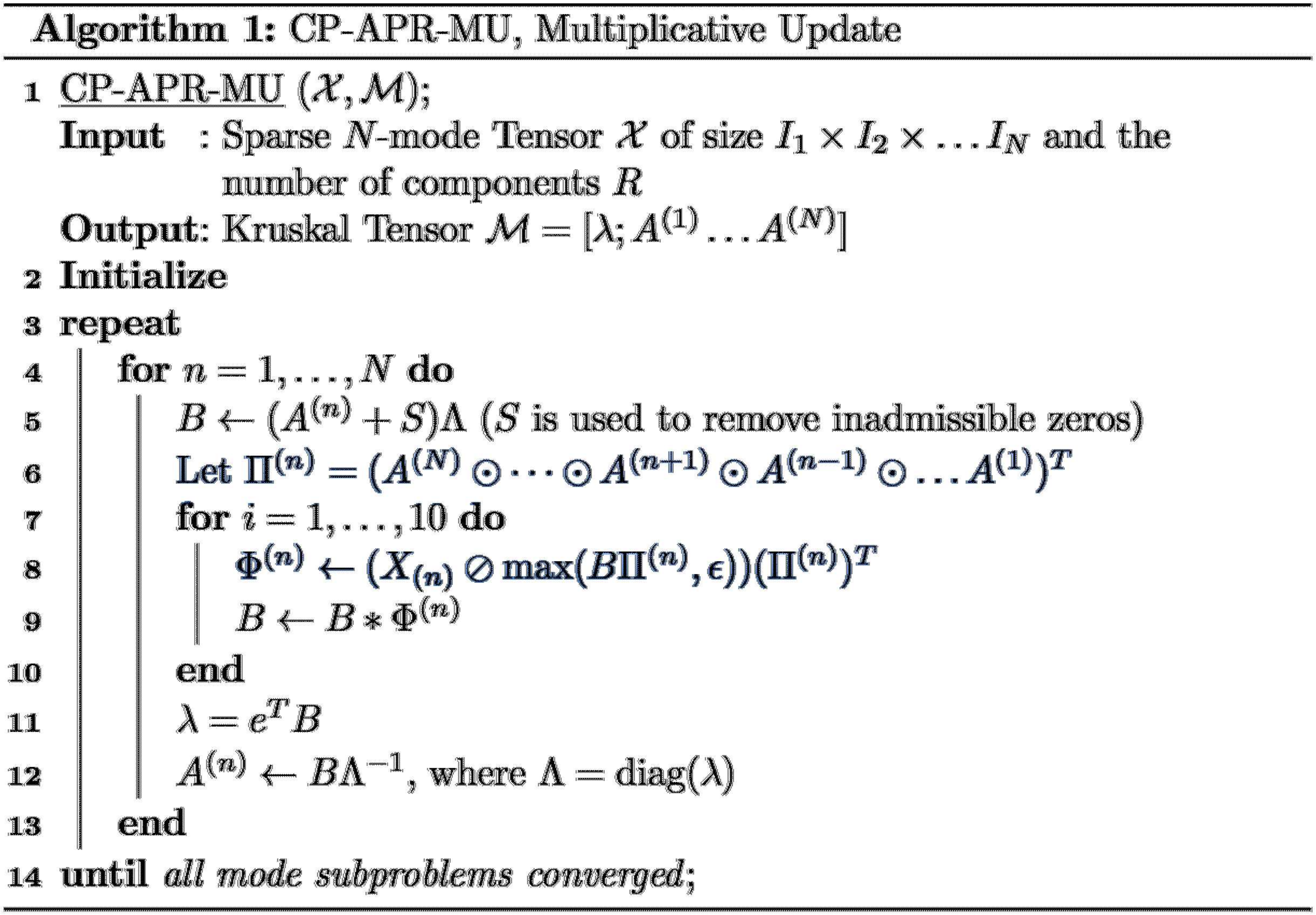
Multiplicative Update (MU)

- Key computations
  - Khatri-Rao Product $\Pi^{(n)}$
- Key features
  - Factor matrix is updated all at once

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 $I_1 \times I_2 \times \dots I_N$ 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
        $B \leftarrow (A^{(n)} + S)\Lambda$ ($S$ is used to remove inadmissible zeros)
        Let $\Pi^{(n)} = (A^{(N)} \odot \dots \odot A^{(n+1)} \odot A^{(n-1)} \odot \dots A^{(1)})^T$
        for $i = 1, \dots, 10$ 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*;
```

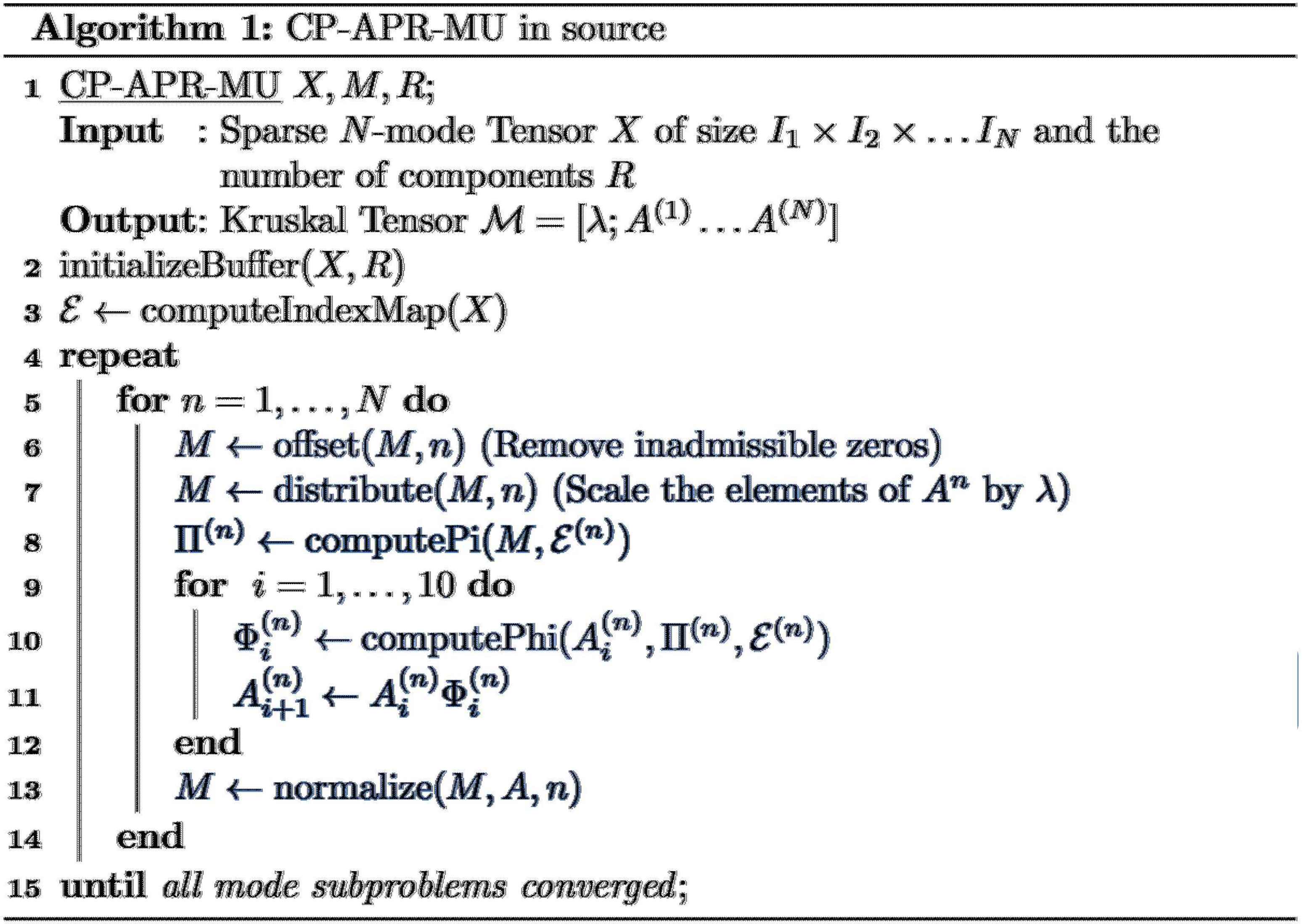
CP-APR-PDNR

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. $\min_{b_i^{(n)} \geq 0} f_{\text{row}}(b_i^{(n)}, x_i^{(n)}, \Pi^{(n)})$
        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

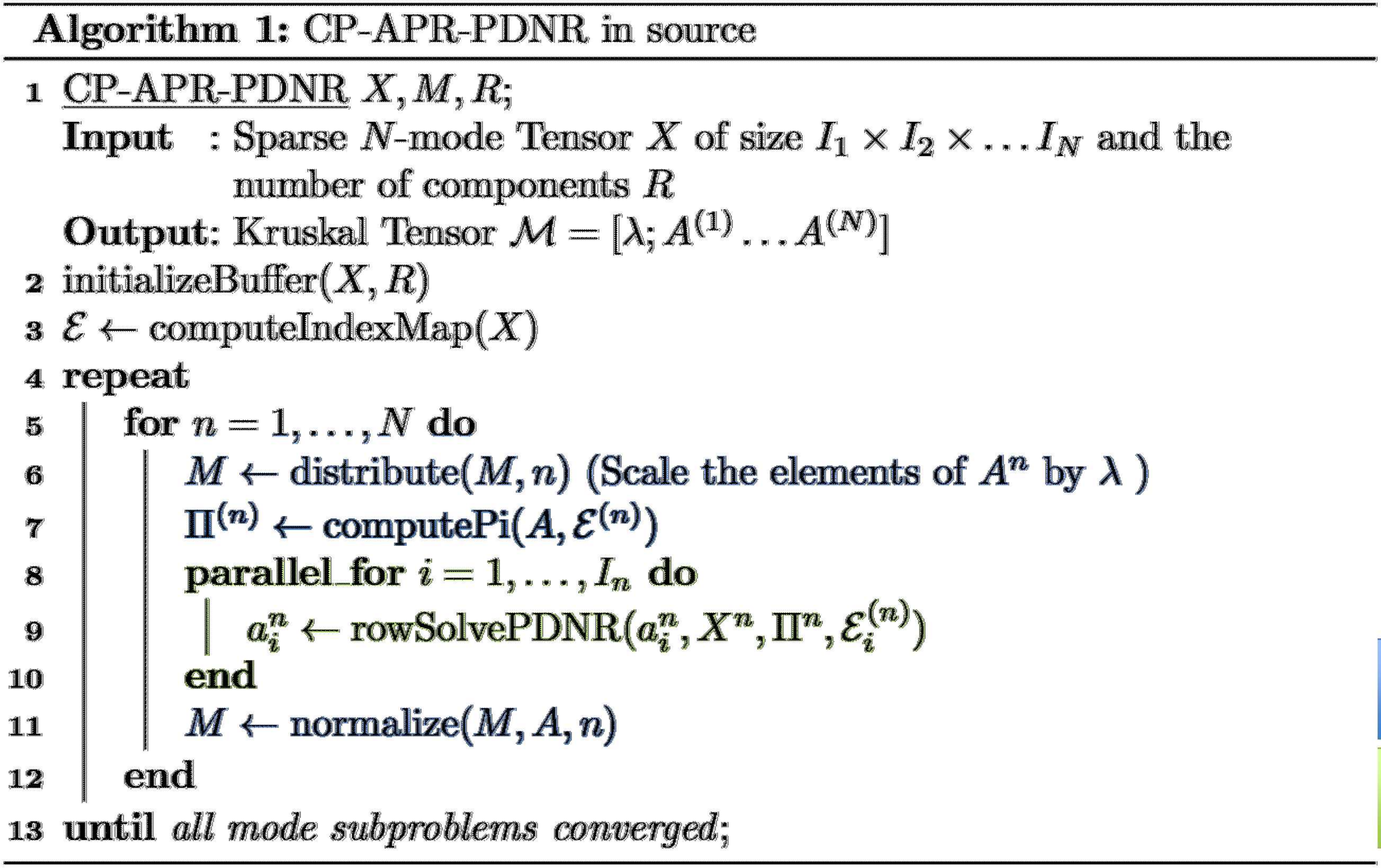
Parallel CP-APR-MU



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 $\lambda$)
        $\Pi^{(n)} \leftarrow$ 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;
```

Parallel CP-APR-PDNR



Algorithm 1: CP-APR-PDNR in source

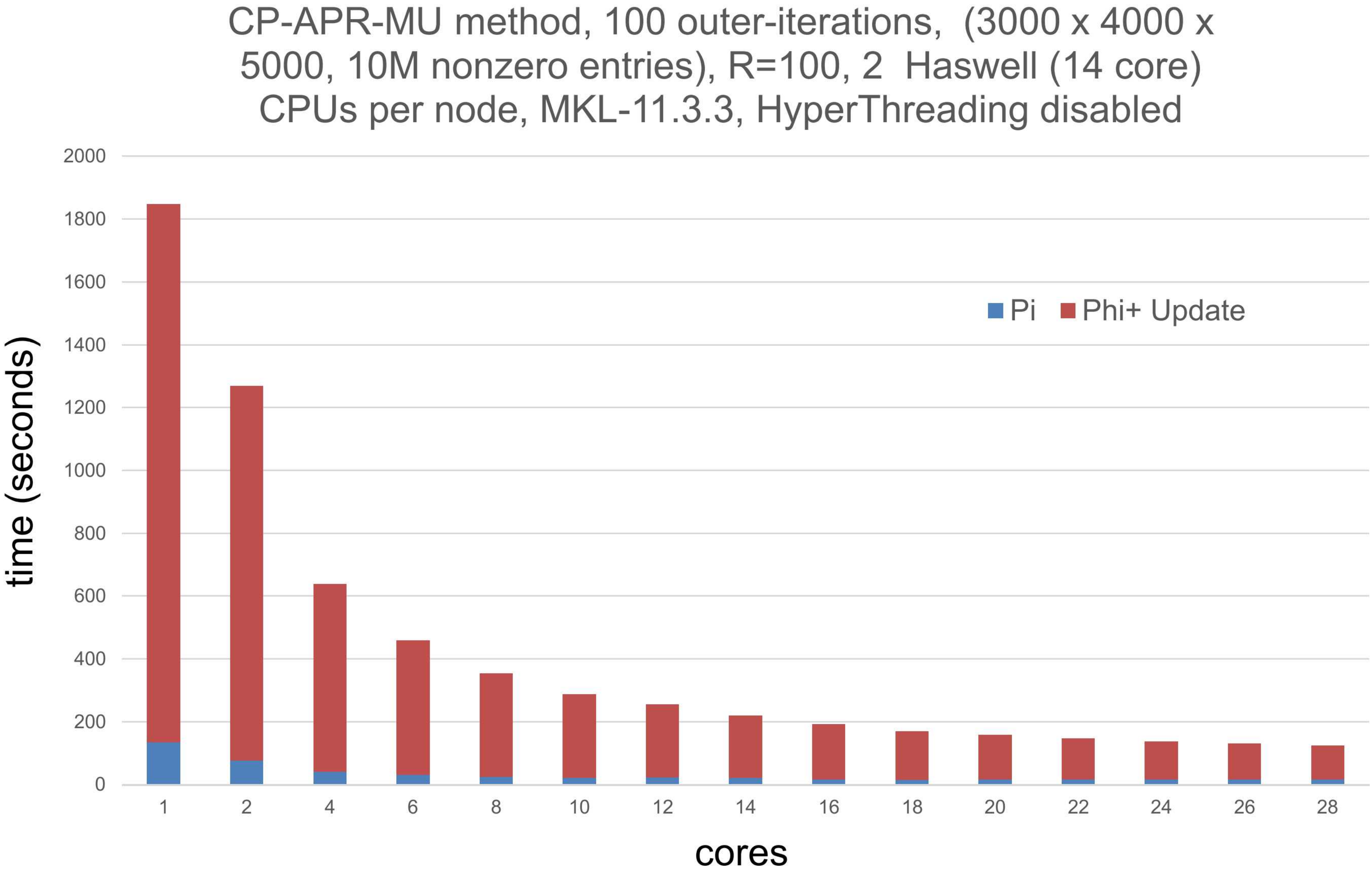
```
CP-APR-PDNR $X, M, R$;
Input  : Sparse $N$-mode Tensor $X$ of size $I_1 \times I_2 \times \dots I_N$ and the
         number of components $R$
Output: Kruskal Tensor $\mathcal{M} = [\lambda; A^{(1)} \dots A^{(N)}]$
initializeBuffer($X, R$)
$\mathcal{E} \leftarrow$ computeIndexMap($X$)
repeat
    for $n = 1, \dots, N$ do
        $M \leftarrow$ distribute($M, n$) (Scale the elements of $A^n$ by $\lambda$ )
        $\Pi^{(n)} \leftarrow$ computePi($A, \mathcal{E}^{(n)}$)
        parallel_for $i = 1, \dots, I_n$ 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;
```

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)



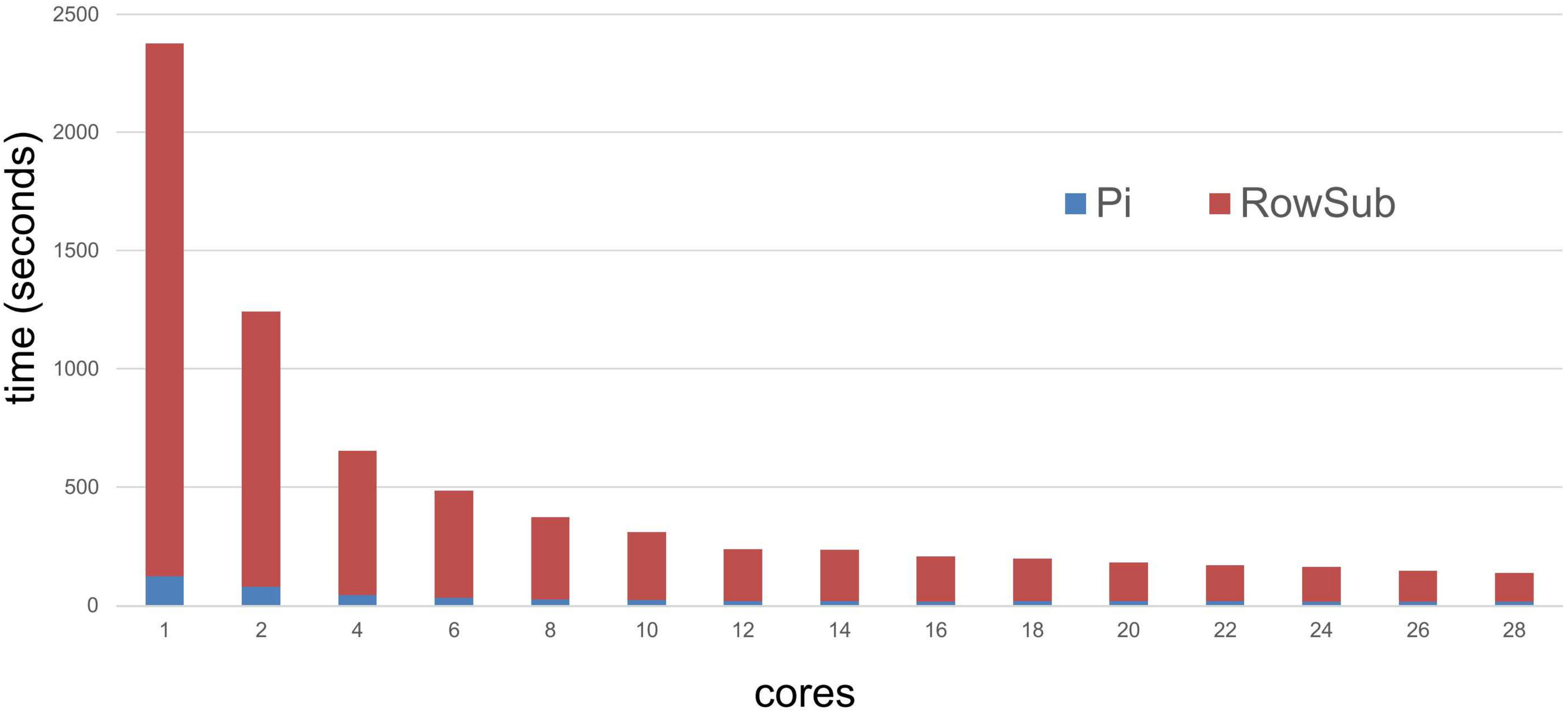
Results: CPAPR-MU Scalability

Data	**CPU 1-core**		**KNL (Cache Mode) 68-core CPU**		**NVIDIA P100 GPU**		**NVIDIA V100 GPU**	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
Random	1849*	1	84	22.01	44.76	41.31	30.05	**61.53**
LBNL	39	1	33	1.18	2.99	13.04	2.09	**18.66**
NELL-2	1157	1	100	11.02	47.17	24.52	28.80	**40.17**
NELL-1	3365	1	257	10.86				
Delicious	4170	1	3463	1.41				

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

CPAPR-PDNR on CPU(Random)

CpAPR-PDNR method, 100 outer-iterations, 1831221 inner iterations total, (3000 x 4000 x 5000, 10M nonzero entries), R=10, 2 Haswell (14 core) CPUs per node, OpenBLAS, LAPACK-3.7.0, HyperThreading disabled

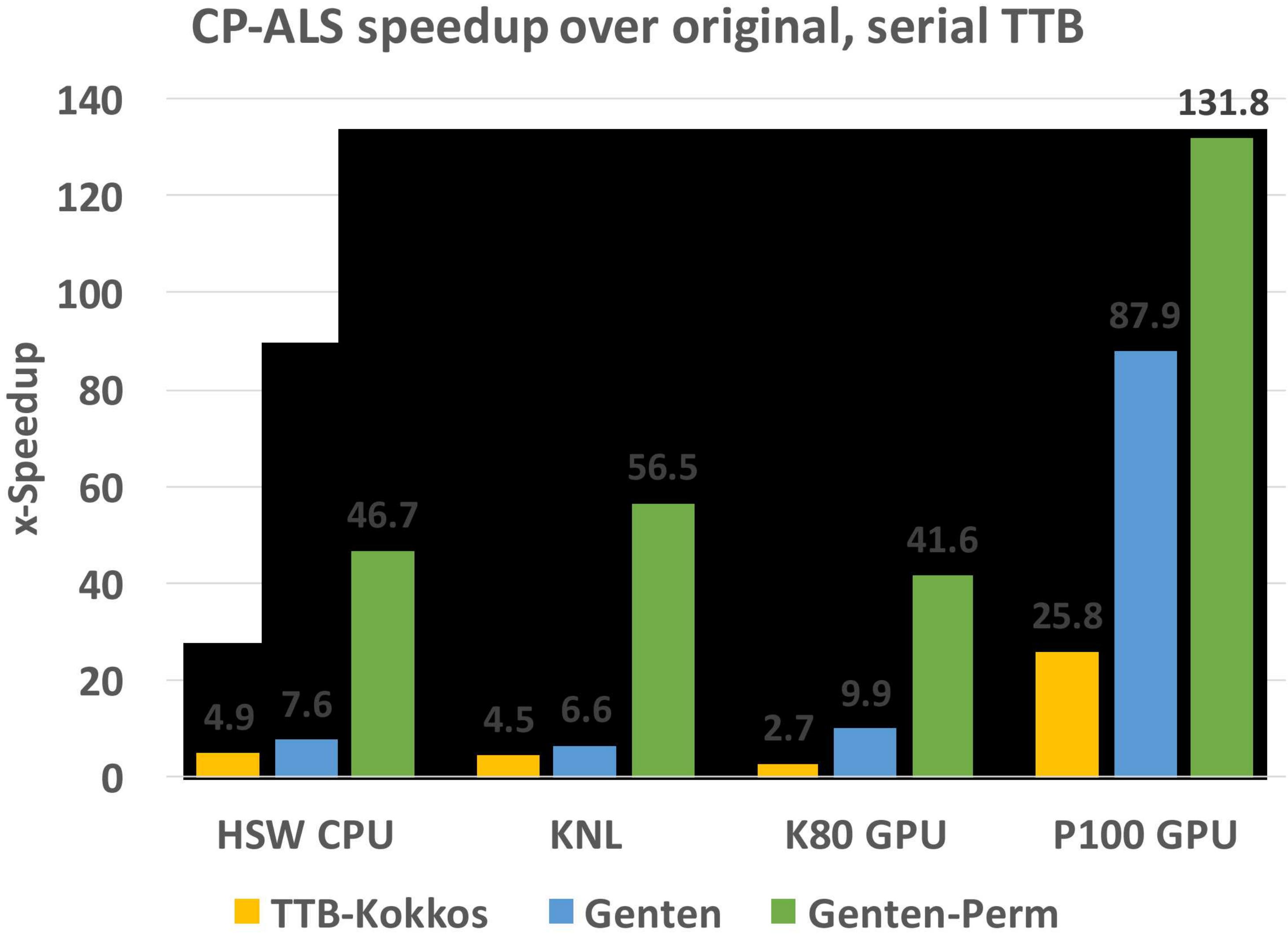


Results: CPAPR-PDNR Scalability

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	238	1	23.7	10.03	14.6	16.28
LBNL	1049	1	187	2.35	191	2.30
NELL-2	5378	1	326	6.63	319	6.77
NELL-1	17212	1	4241	4.05	3974	4.33
Delicious	28053	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

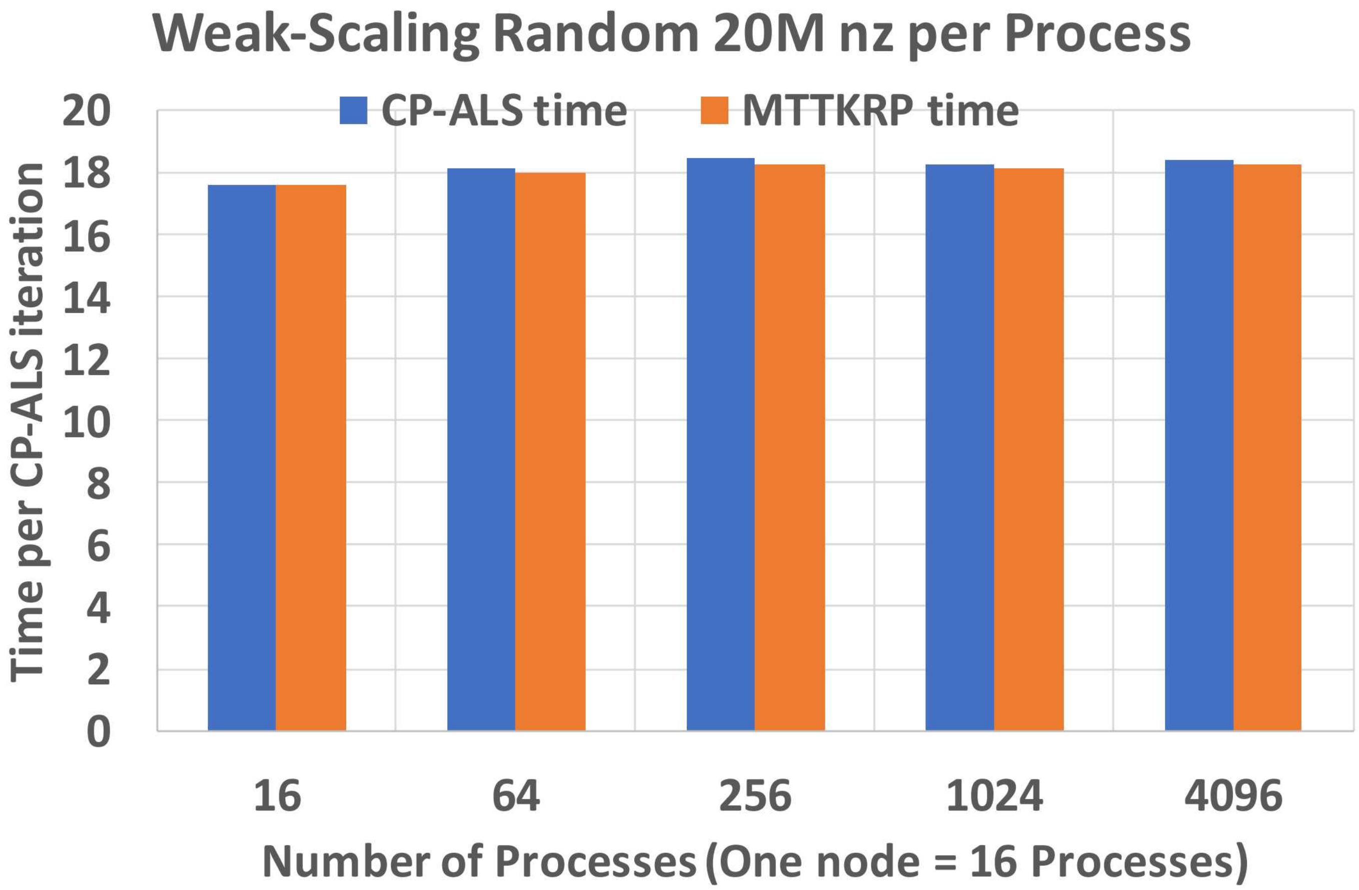
CP-ALS using Kokkos



POC: *Eric Phipps (etphipp@sandia.gov)*

CP-ALS using Kokkos + Trilinos

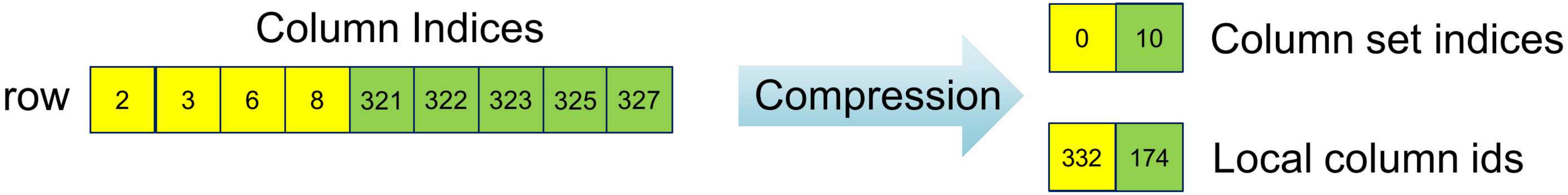
- CP-ALS for **huge** sparse tensors in distributed memory
- 1.6TB tensor (82B nonzeros) on 4096 MPI processes



POC: Karen Devine (kddevin@sandia.gov)

Graph Challenge

GC 2: Matrix Compression



Column Indices

row	2	3	6	8	321	322	323	325	327

Compression

0	10	Column set indices
332	174	Local column ids

- Compression used to improve performance
  - Encodes columns using fewer integers
  - Reduces number of operations and memory required in symbolic phase
  - Allows ”vectorized” bitwise union/intersection of different rows
- Effectiveness of compression varies greatly with data
  - Large random graphs compress poorly (R-Mat <1% compression storage)
  - However, still helpful for many random graphs (e.g., power-law) – effective for dense rows (improves load balance, operation count)

Compression consistently improves triangle counting performance

GC 3: Visitor Pattern

- KKMEM based triangle counting supports visitor pattern
  - Concept fundamental to BGL and MTGL
- Functor passed to triangle identification function, which allows method to be run once triangle is found
  - For triangle counting: triangleCount++;
  - Flexibility allows for more complex analysis of triangles, miniTri

Visitor pattern support provides additional flexibility to analysts

BGL = Boost Graph Library, MTGL = MultiThreaded Graph Library

Summary

- Overview of Kokkos Ecosystem for Performance Portability
  - Parallel patterns, data structures to support performant parallel computing on many/multi-core nodes
  - **Goal:** Write algorithms once, run everywhere (almost) optimally
  - Core to increasing number of DOE HPC/ECP applications
  - Potential to greatly impact High Performance Data Analytics (HPDA)
- Two examples of Kokkos impacting HPDA
  - **Grafiki** – linear algebra based graph/hypergraph analysis
  - Scalable tensor analysis (**SparTen**, **GenTen**)
  - Kokkos enabled fair performance on GPUs/CPUs; improvements needed
- Example of highly optimized Kokkos-based graph analysis kernel
  - Triangle counting/enumeration KKTri (using Kokkos Kernels)