Exploiting Scientific Software to Solve Problems in Data Analytics

Karen Devine, Sandia National Laboratories
With Erik Boman, Daniel Dunlavy, Alicia Klinvex, Tammy Kolda, Siva Rajamanickam, and Michael Wolf

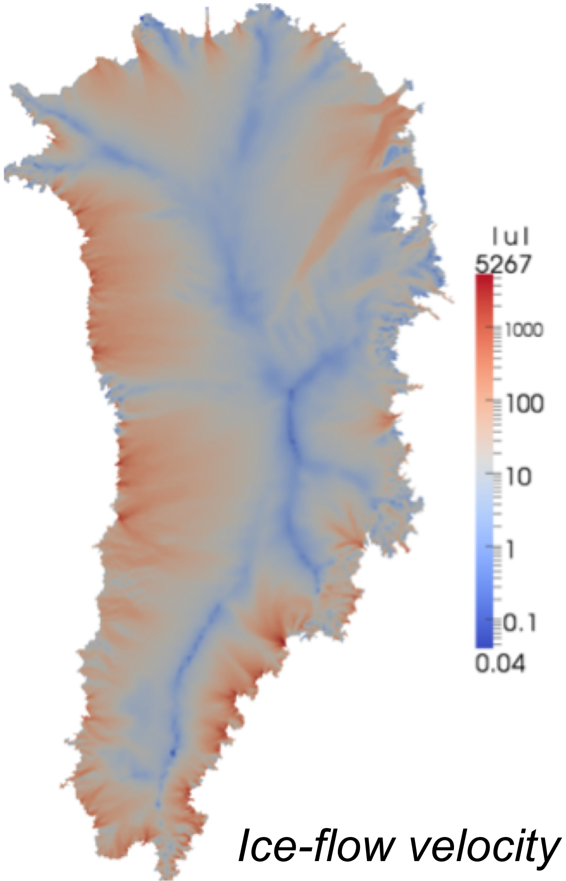
IPAM Workshop on HPC for Computationally and Data-Intensive Problems
November 5-9, 2018

Exceptional service in the national interest

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2016-10893 PE.

Exploiting scientific software to solve problems in data analytics

- DOE has made tremendous investments in physics-based simulations for scientific discovery and stockpile stewardship
  - Supercomputing hardware
  - Numerical libraries and operating systems
  - Physics models and simulation codes
- Tremendous opportunities for data science
  - Exploit hardware and libraries developed for physics-based simulation
  - Use directly, or with some twists
  - Potential to save development time for new applications



Ice-flow velocity magnitude [m/yr] on the surface of the Greenland Ice Sheet, as computed by the Albany/FELIX finite-element code (Perego, SNL)

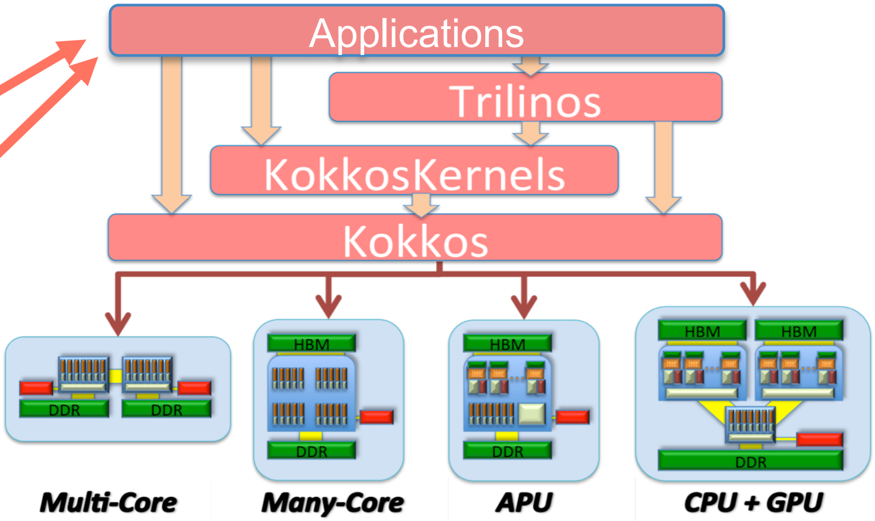
Case study:
Exploiting Trilinos for data analytics

- Trilinos solver framework
  - High performance linear algebra data structures and algorithms
  - High performing eigensolvers, linear solvers, partitioners
  - Scales to billions of matrix rows/columns/entries

- Today's talk:
  - Using Trilinos directly: spectral hypergraph clustering in Grafiki
  - Adding some twists: two-dimensional matrix partitioning
  - Building new applications: sparse tensor decomposition

Trilinos: open-source toolkit of mathematical algorithms for HPC

- Capabilities: Component-based approach
  - Matrix/Vector classes
  - Linear solvers / preconditioners
  - Eigensolvers
  - Nonlinear solvers
  - Optimization
  - Discretization
  - Load balancing
  - Simple mesh generation
  - Time integrators
  - Automatic differentiation
- Distributed memory (MPI)
- On-node parallelism via Kokkos performance portability layer
- Traditional realms:
  - Solid mechanics, fluid flow, electrical circuits, etc.
- Goal: Investigate use for large-scale data analysis



Applications

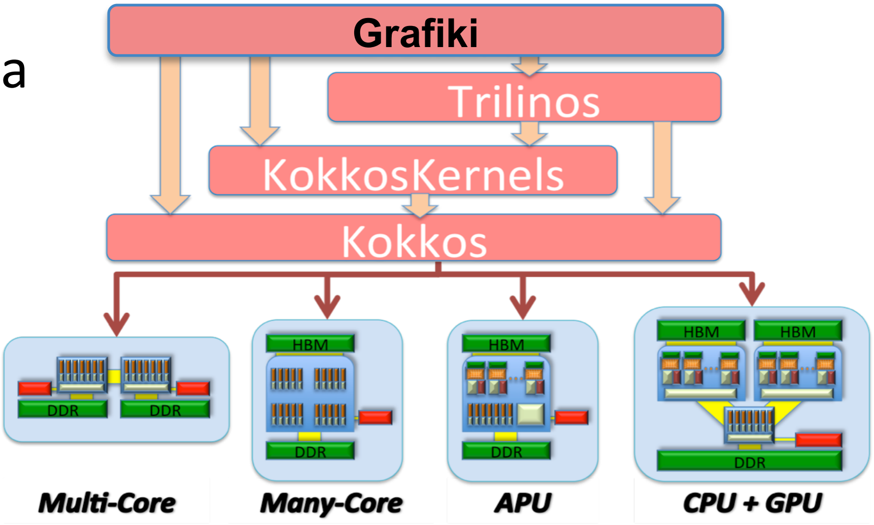
Trilinos

KokkosKernels

Kokkos

Using Trilinos directly: Grafiki -- Trilinos for Large-Scale Data Analysis

- *Michael Wolf, Alicia Klinvex, Daniel Dunlavy*
- Grafiki: formerly TriData
- Goal: Sparse linear algebra-based data analysis
  - Target: very large data problems
  - Target: distributed memory and single node HPC architectures
- Additionally
  - Testbed for improving how Trilinos can be leveraged for data analytics
  - Support GraphBLAS-like linear algebra analysis techniques
- Focus: Graph and Hypergraph Analysis



Grafiki uses many components of Trilinos

- Tpetra parallel matrix/vector/communication classes
  - MPI+X (OpenMP, CUDA via Kokkos performance portability layer)
  - Supporting > 2B rows/cols/nonzeros
  - Compressed Sparse Row matrices for graph storage
  - Multivectors for eigenvector storage
- Anasazi eigensolver package
  - Spectral Clustering, Vertex/Edge eigencentrality (graphs, hypergraphs)
- Belos linear solver package
  - Mean hitting time analysis on graphs

Grafiki Example: Evaluating use of Hypergraphs in Clustering of Relational Data

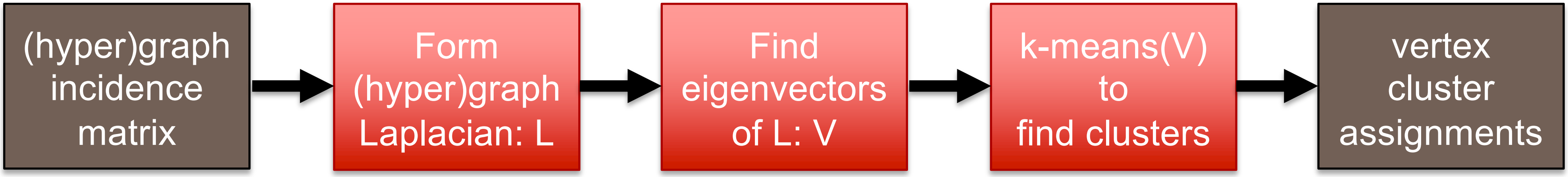
- Clustering: Determine groupings of data objects given relationships among the objects
  - Relationships may be represented as graph or hypergraph
- Focus: **spectral clustering**
  - Compute the smallest eigenpairs of the graph or hypergraph Laplacian
  - Normalized graph Laplacian:

$$L_G = I - D_{vG}^{-1/2}(H_G H_G^T - D_{vG}) D_{vG}^{-1/2}$$

  - Hypergraph Laplacian (Zhou, et al., 2006)

$$L_h = I - D_{vH}^{-1/2} H_H D_{eH}^{-1} H_H^T D_{vH}^{-1/2}$$

  - Eigenvectors used to group vertices into clusters (sorting, kmeans++, …)



(hyper)graph incidence matrix → Form (hyper)graph Laplacian: L → Find eigenvectors of L: V → k-means(V) to find clusters → vertex cluster assignments

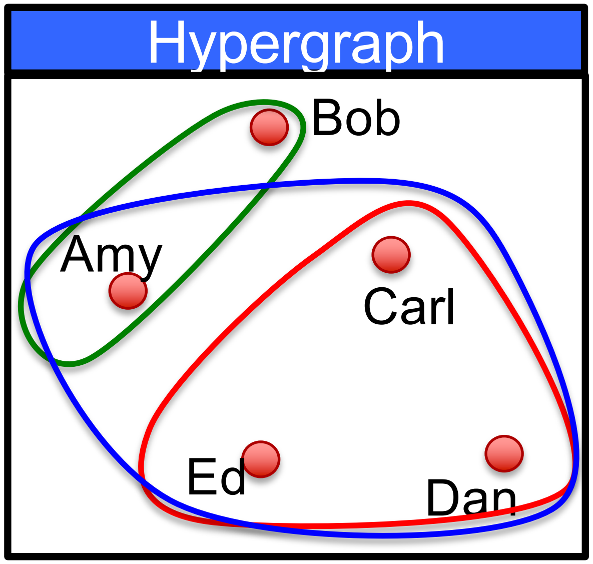
What is a hypergraph?

Hyperedges: Emails

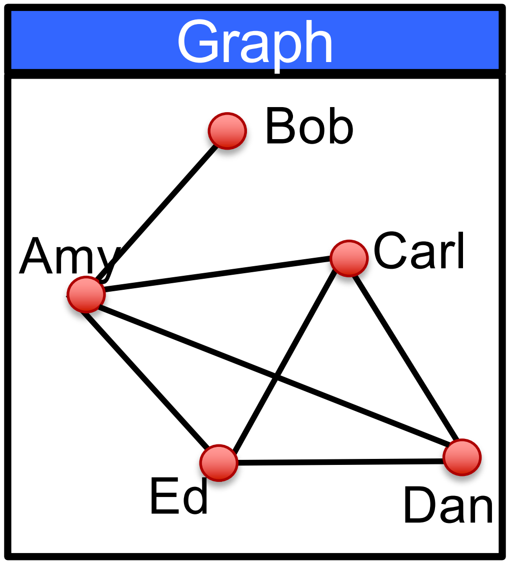
Vertices: Users

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

Relational data / hypergraph incidence matrix



Hyperedges connect
one or more vertices

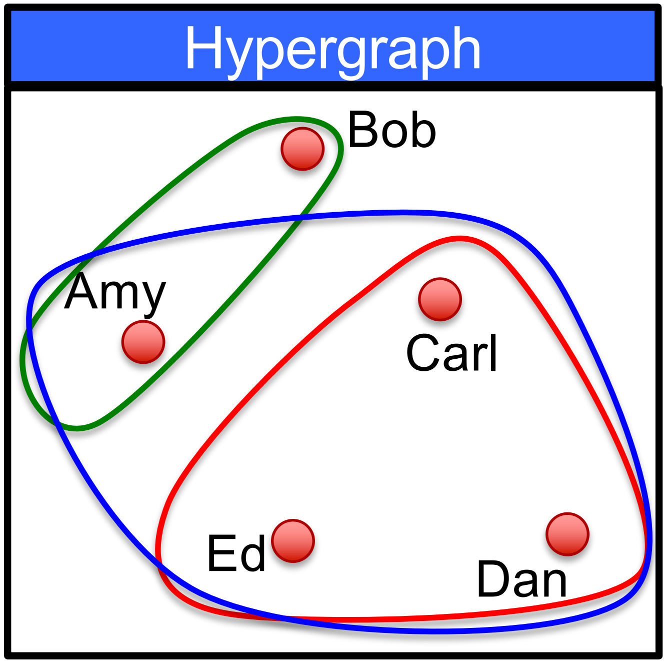


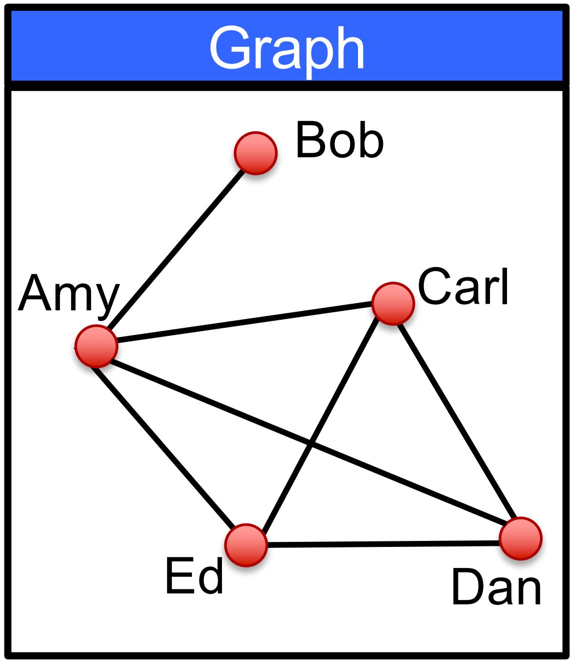
Edges connect
two vertices

- Generalizations of graphs
  - Hyperedges represent multiway relationships between vertices
- Convenient representations of relational data
  - Each email (subset of users) can be represented by hyperedge
  - Relational data often stored as hypergraph incidence matrices

Hypergraphs represent multiway relationships unambiguously

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





- Typically graph models lose information
  - Were Carl, Dan, and Ed involved in same email?
  - Fix: multi-graphs + metadata, changes to algorithms

Hypergraph to Graph: Clique Expansion

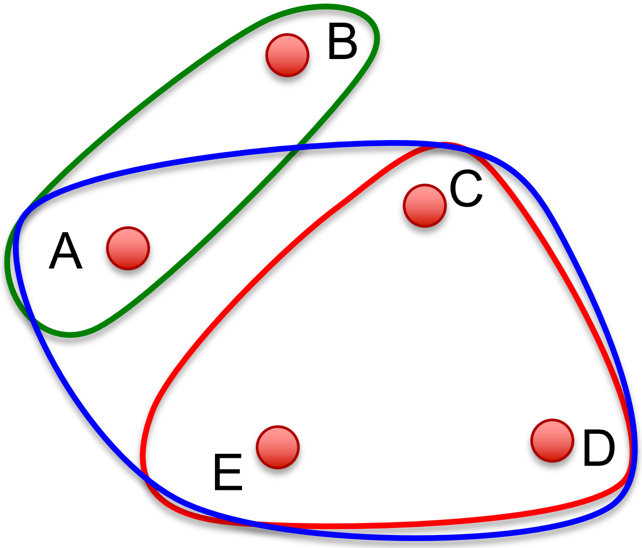
Hyperedges

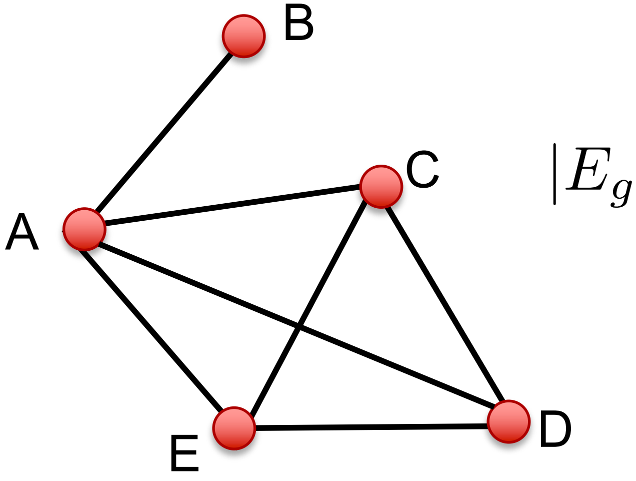
Vertices

	1	2	3
A	1		1
B	1		
C		1	1
D		1	1
E		1	1

Graph Edges

	1	2	3	4	5	6	7	8	9	10
A	1				1	1	1			
B	1									
C		1	1		1			1	1	
D		1		1		1		1		1
E			1	1			1		1	1





$$|E_g| = \sum_{e_h \in E_h} \binom{d(e_h)}{2}$$

hyperedge cardinality

Graph obtained through clique expansion of hypergraph

Hypergraphs have computational advantages

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

$$|E_g| = \sum_{e_h \in E_h} \binom{d(e_h)}{2}$$

- Hypergraphs require significantly **less storage space** than graphs generated using clique expansion
- Hypergraph incidence matrices require **fewer operations** for matrix-vector multiplication

Trilinos interface avoids need to explicitly form Laplacians

- Laplacians:

$$L_G = I - D_{vG}^{-1/2}(H_G H_G^T - D_{vG})D_{vG}^{-1/2}$$
$$L_h = I - D_{vH}^{-1/2} H_H D_{eH}^{-1} H_H^T D_{vH}^{-1/2}$$

- One option: Explicitly form/store Laplacian
- Instead: **Trilinos interfaces allows implicit representation of Laplacian as series of SpMV and vector addition operations**
  - Operator interface describes how to apply a matrix to a vector
  - Eigensolvers use the Operator interface
  - Store only incidence matrix, degree matrices
  - Avoid expensive matrix-matrix products
  - Support dynamic graphs – easier to change incidence matrix than Laplacian

Computational advantages to not explicitly forming Laplacians

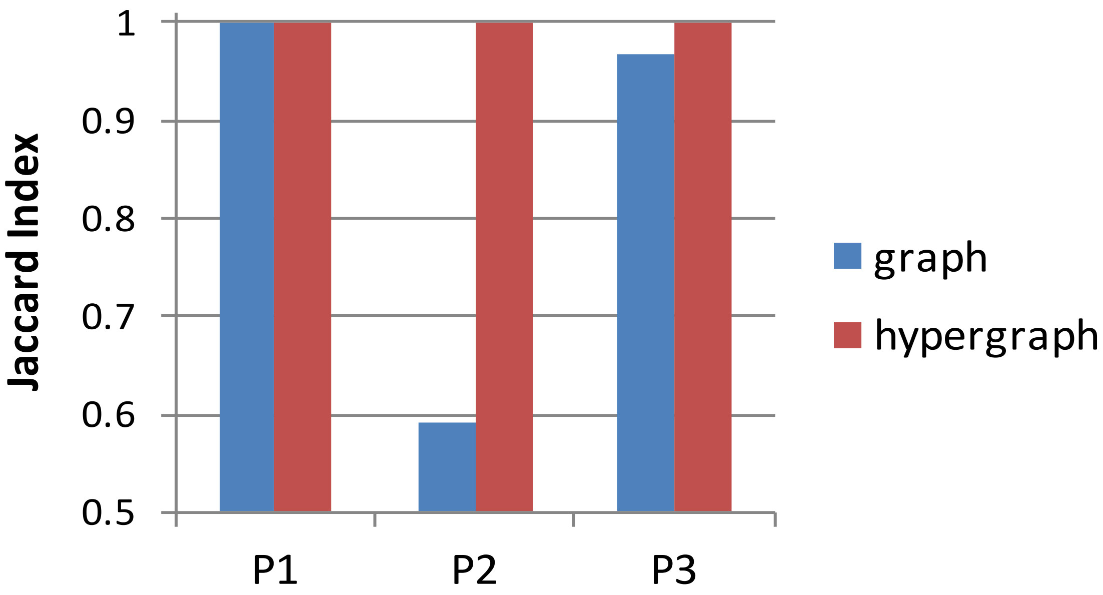
SpMV = Sparse matrix-dense vector multiplication

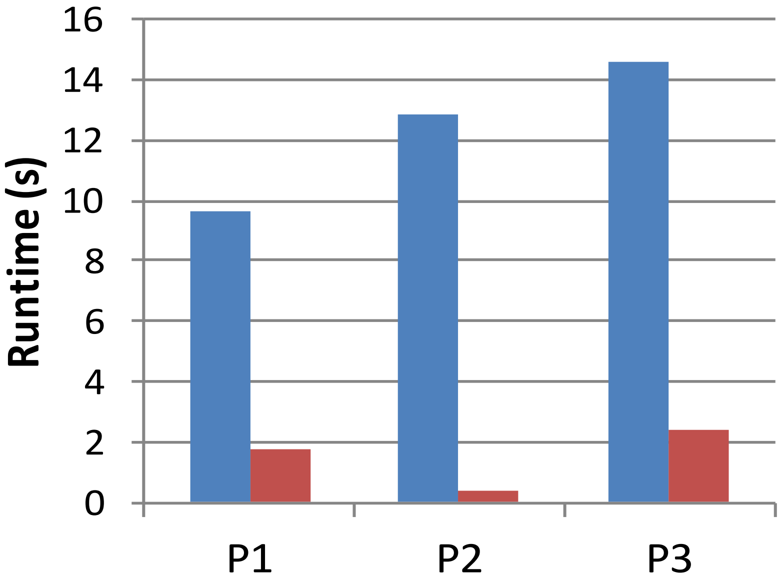
Using Trilinos enables easy comparison of models

- **Quality:** Jaccard index
  - T = "ground truth" assignments
  - P = predicted assignments
  - J(T,P)=1 ➔ exact match

$$J(T,P) = \frac{|T \cap P|}{|T \cup P|}$$

	P1	P2	P3
Number of clusters	10	5	10
Vertices per cluster*	10,000	10,000	10,000
Intra-cluster hyperedges*	40,000	20,000	20,000
Inter-cluster hyperedges*	50,000	200,000	200,000
Intra-cluster h-edge cardinality*	5	10	5
Inter-cluster h-edge cardinality*	5	3	5



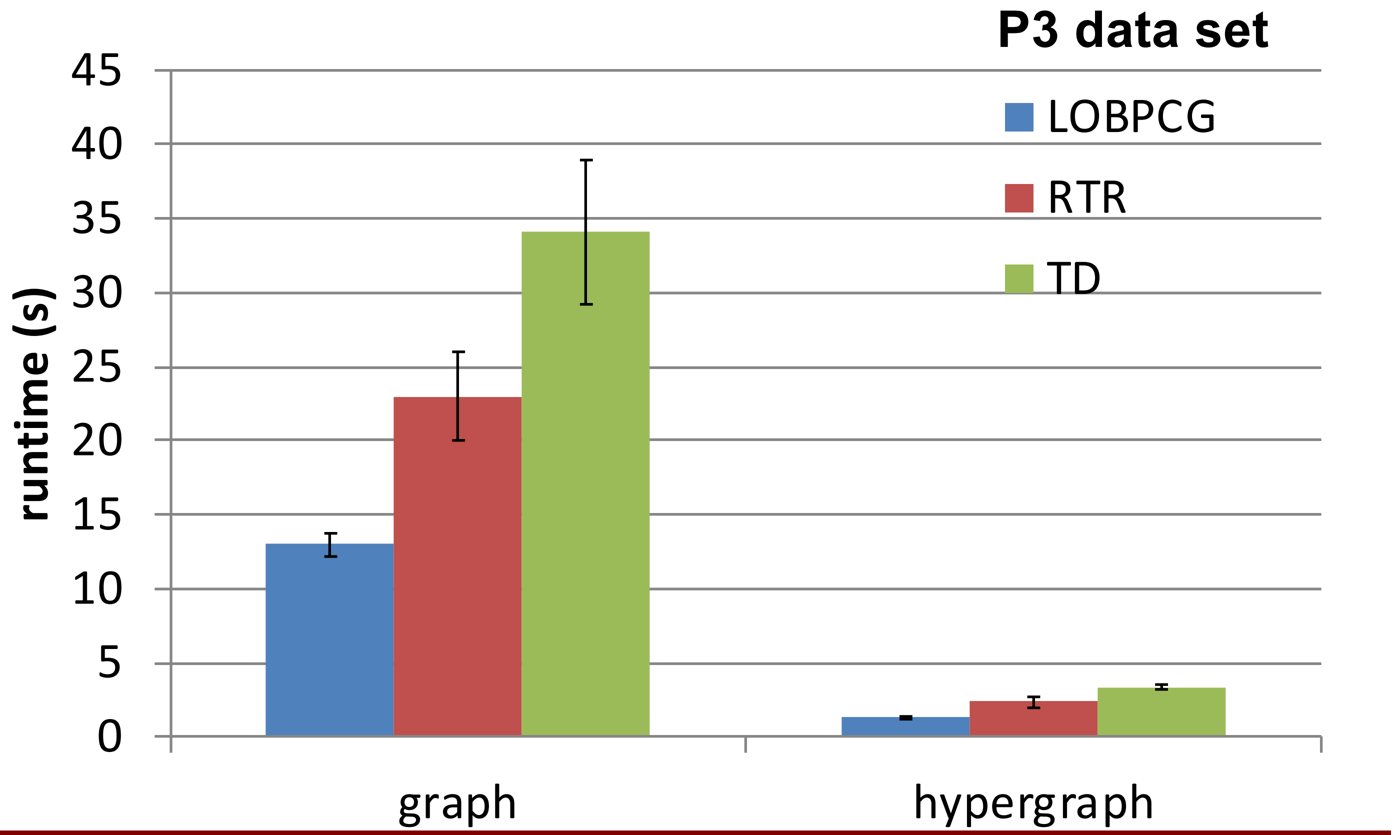


Hypergraph clusters more similar to "ground truth" than graph clusters

Hypergraph less expensive computationally than graph (up to 30x faster)

Trilinos' Anasazi toolkit enables easy comparison of eigensolvers

- Locally Optimal Block Preconditioned Conjugate Gradient method (LOBPCG)
- Riemannian Trust Region method (RTR)
- TraceMin-Davidson (TD)

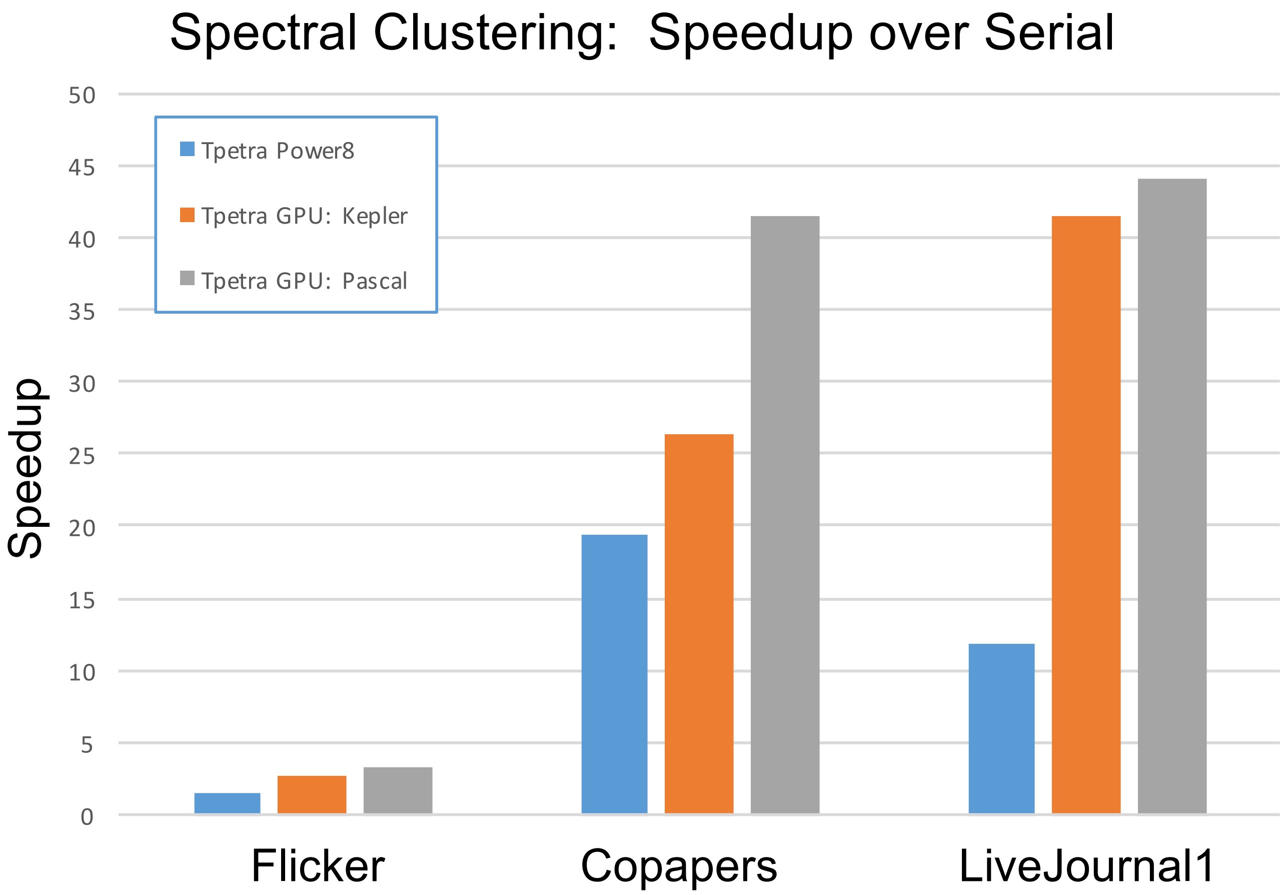


Relatively loose convergence tolerance (e.g., 10^{-2}) suffices for clustering

"Clustering network data through effective use of eigensolvers and hypergraph models," A. Klinvex, M. Wolf, D. Dunlavy, 14th Copper Mountain Conf. on Iterative Methods, 2016.

Use of Trilinos' Tpetra classes enables performance portability

- CPU: 20 core IBM Power 8, 3.42 GHz
- GPU: NVIDIA Kepler
- GPU: NVIDIA Pascal P100

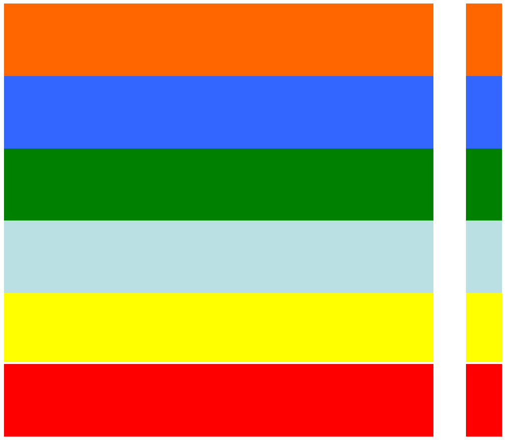


- **One Grafiki implementation runs on all three platforms**

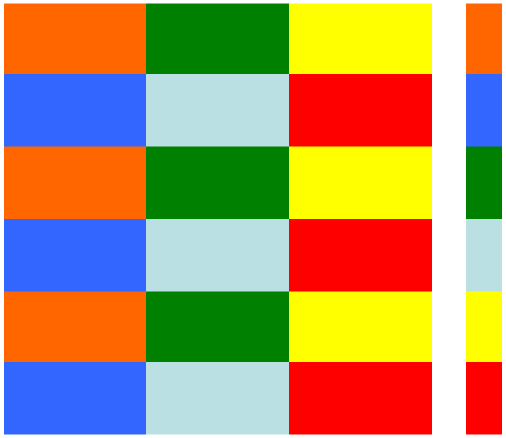
"TriData: High Performance Linear Algebra-Based Data Analytics," M. Wolf, D. Dunlavy, R. Lehoucq, J. Berry, D. Bourgeois. SIAM Conf. Parallel Processing, 2018

Adding a twist: 2D matrix partitioning for power-law graphs

- *Erik Boman, Siva Rajamanickam, Karen Devine*
- Goal: reduce MPI communication overhead in solvers for non-physics data (e.g., power-law graphs, social networks)
- Approach:
  - Exploit Trilinos' flexible parallel distributions to reduce the number of processors with which communication is needed in sparse matrix-vector multiplication (SpMV)
  - Combine graph partitioning and flexible layouts to further reduce communication



1D matrix distribution

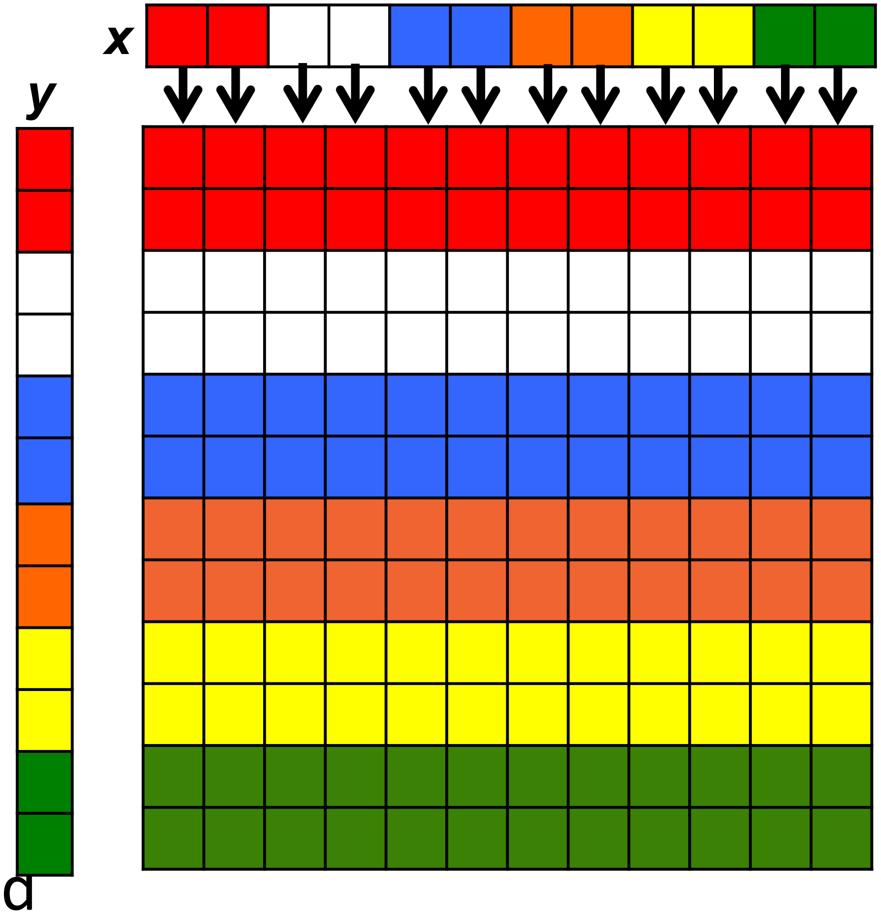


2D matrix distribution

"Scalable Matrix Computations on Scale-Free Graphs using 2D Graph Partitioning," E. Boman, K. Devine, S. Rajamanickam. SC13, 2013

Typical matrix partition: 1D distribution

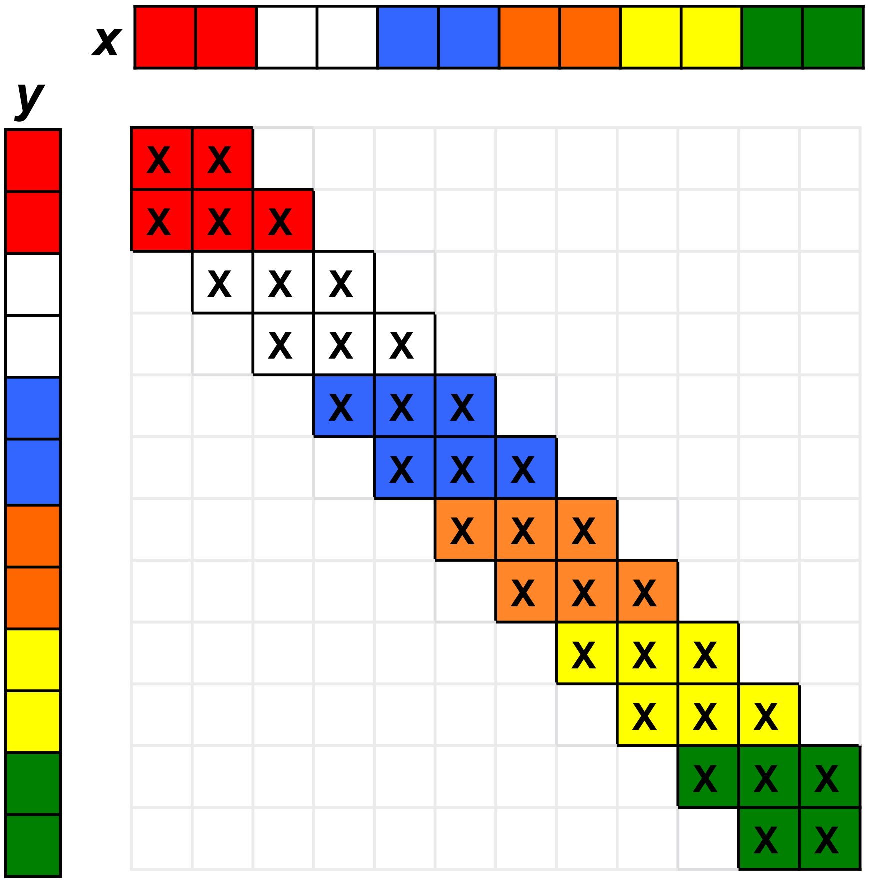
- Entire row (or column) of matrix assigned to a single processor
- Vectors use same distribution
- During SpMV,
  - Expand (vertical): processor receives (via communication) *x* vector entries needed to match non-zeros in owned rows.
  - Each processor does local partial products with owned nonzeros
  - Fold (horizontal): no communication required if *y*-vector layout matches matrix
- Non-zero structure of matrix determines communication needed



Trilinos' Tpetra Maps describe parallel distribution of matrix and vectors

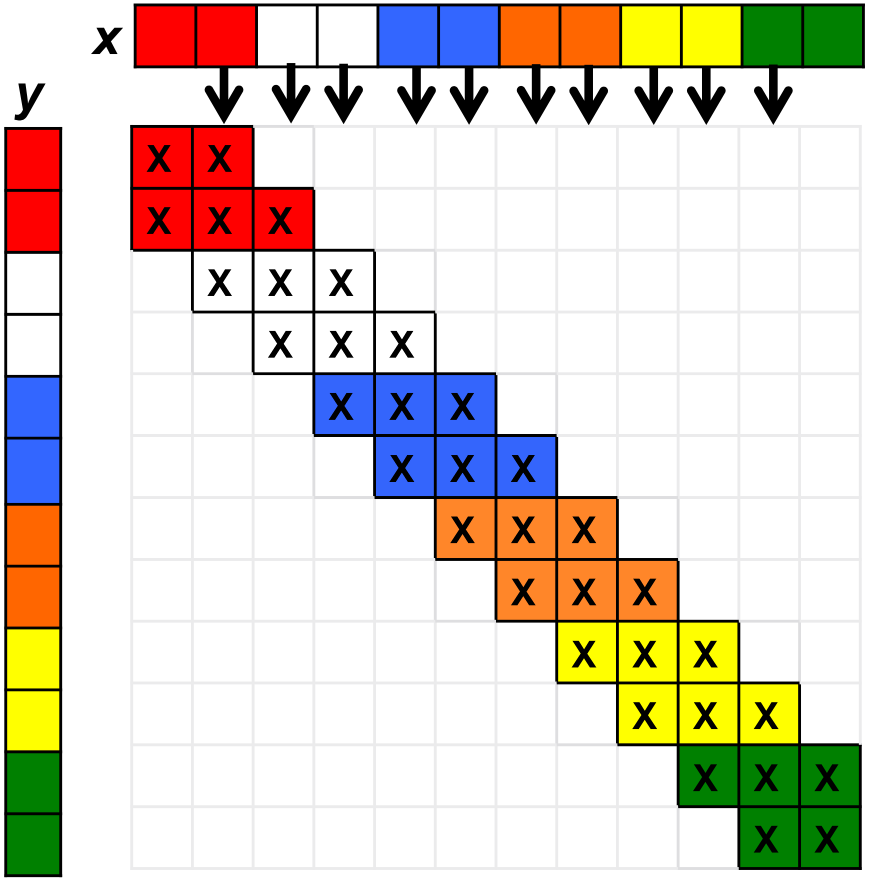
- Four maps needed for SpMV:
  - Row map: Rows of the matrix *A* for which the processor has nonzeros
  - Column map: Columns of *A* for which the processor has nonzeros
  - Domain map: Input vector *x* entries on the processor
  - Range map: Output vector *y* entries on the processor

Rank 2 (Blue)
Row Map = {4, 5}
Column Map = {3, 4, 5, 6}
Range/Domain Map = {4, 5}



1D distributions work well for many physics-based simulations

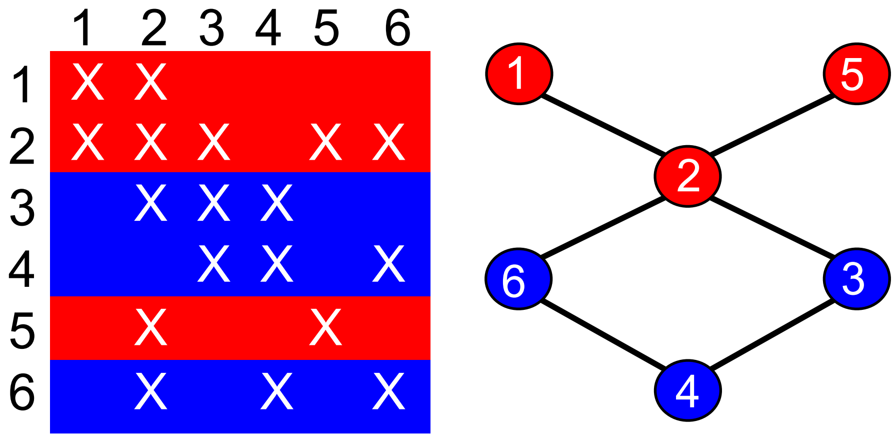
- Data locality in mesh-based applications limits amount of communication needed.
- Several ways to distribute rows:
  - 1D-Block: each processor given block of *N/p* rows
  - 1D-Random: each processor given *N/p* randomly selected rows
  - 1D-GP: assign rows based on output of graph partitioning algorithm



Graph partitioning: 1D-GP

(Kernighan, Lin, Schweikert, Fiduccia, Mattheyes, Simon, Hendrickson, Leland, Kumar, Karypis, et al.)

- Explicitly attempts to minimize communication costs induced by partition
- Represent matrix *A* as a graph:
  - One vertex *j* per row a_j
  - Edge (*i, j*) exists iff $a_{ij} \neq 0$
  - Vertex weights = # nonzeros in row

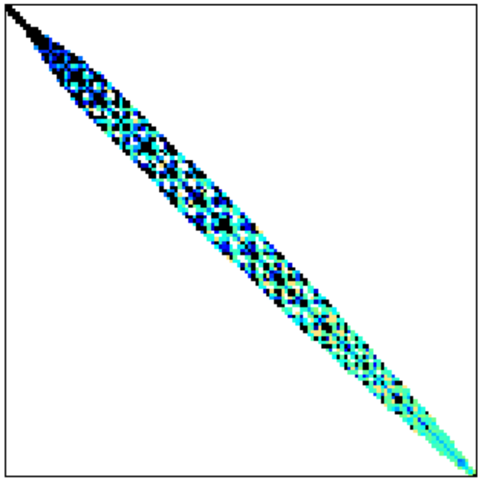


	1	2	3	4	5	6
1	X	X				
2	X	X	X		X	X
3		X	X	X		
4			X	X		X
5		X			X	
6		X		X		X

- Goal: Assign equal vertex weight to parts while minimizing weight of edges between parts (i.e., cut by part boundary)
- Highly effective for mesh-based PDE problems
  - Mostly local connectivity (e.g., local support for basis functions)
  - Regular structure (e.g., dual graph of mesh)
- Many high quality graph partitioners available: Chaco (Sandia), ParMETIS (U.Minn.), Scotch (Inria/Bordeaux), PuLP (Sandia/PennSt)

Example: Finite element matrix

- Structural problem discretizing a gas reservoir with tetrahedral finite elements
- Platform: SNL Redsky cluster
  - 2.93 GHz dual socket/quad core Nehalem X5570 procs
  - 3D torus InfiniBand network
- Graph partition gives 25% reduction in SpMV time relative to 1D-Block
  - Improves load balance
  - Reduces communication volume

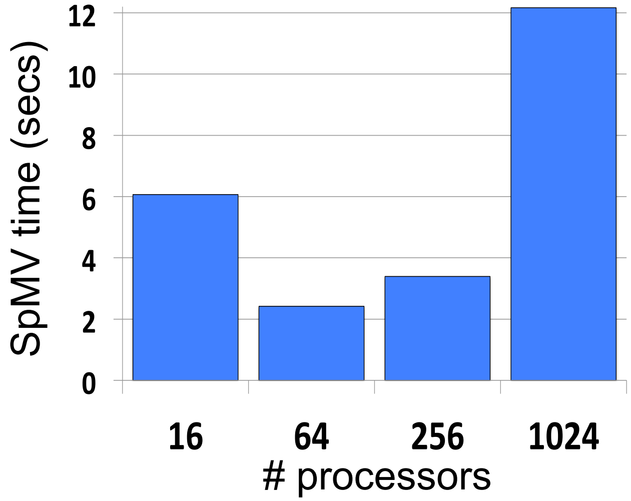


Serena matrix
Janna & Ferronato
U.Florida Sparse Matrix Collection

Serena: 1.4M rows; 65M nonzeros; Max 249 nz/row; Avg 46 nz/row 1024 processes				
Method	Imbalance in nonzeros (Max/Avg per proc)	Max # Messages per SpMV	Comm. Vol. per SpMV (doubles)	100 SpMV time (secs)
1D-Block	1.2	55	4.4M	0.20
1D-Random	1.0	1023	62.1M	13.62
1D-GP	1.1	98	1.1M	0.15

CounterExample: Social network matrix

- Social networks, web graphs, etc., have very *different structure* from PDE discretizations
  - Power-law degree distributions; scale-free properties
- Graph partitioning can reduce SpMV time
  - Reduces imbalance and communication volume
- But large number of messages hurts scaling
  - Nearly all-to-all communication

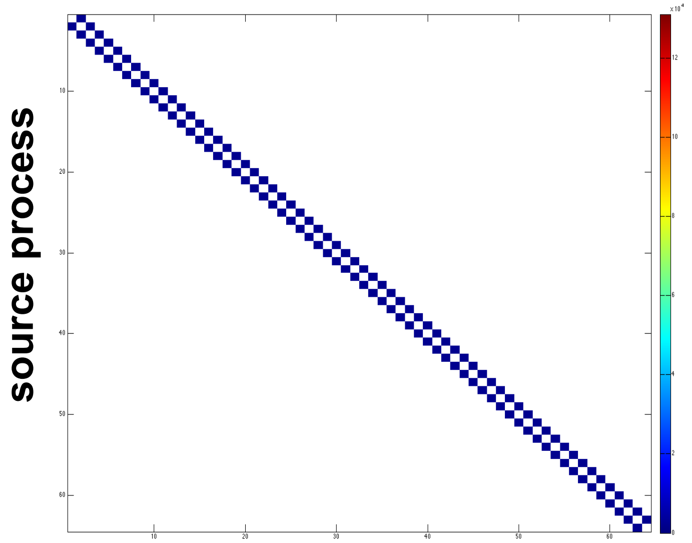


Strong scaling of 1D-GP for com-liveJournal matrix Yang & Leskovec Stanford SNAP collection

com-liveJournal: 4M rows; 73M nonzeros; Max 15K nz/row; Avg 18 nz/row 1024 processes				
Method	Imbalance in nonzeros (Max/Avg per proc)	Max # Messages per SpMV	Comm. Vol. per SpMV (doubles)	100 SpMV time (secs)
1D-Block	12.8	1023	34.5M	14.72
1D-Random	1.3	1023	66.3M	14.00
1D-GP	1.2	1011	18.9M	12.17

In 1D, irregular matrix structure drives greatly increased communication cost

2D Finite Difference (9 point)



P=64 **destination process**

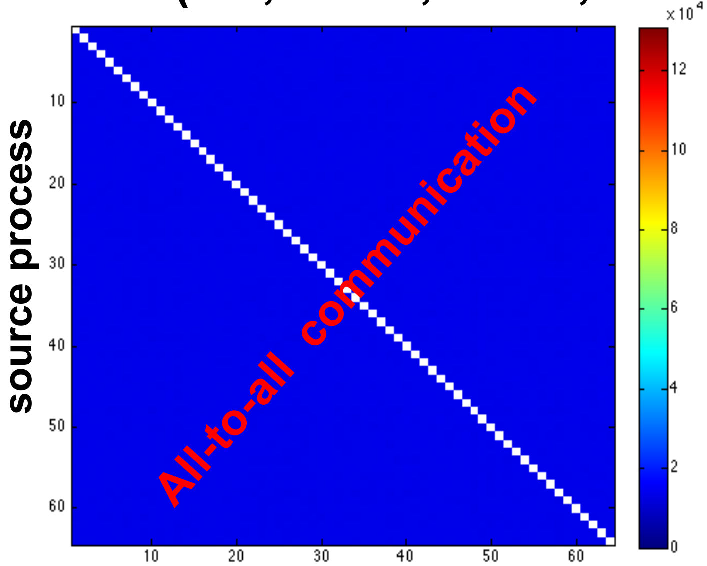
Number of Rows: 2^{23}
Avg. nonzeros/row: 9

NNZ/process
min: 1.17E+06
max: 1.18E+06
avg: 1.18E+06
max/avg: 1.00

# Messages
total: 126
max: 2

Volume
total: 2.58E+05
max: 4.10E+03

R-Mat (0.5, 0.125, 0.125, 0.25)



P=64 **destination process**

Number of Rows: 2^{23}
Nonzeros/Row: 8

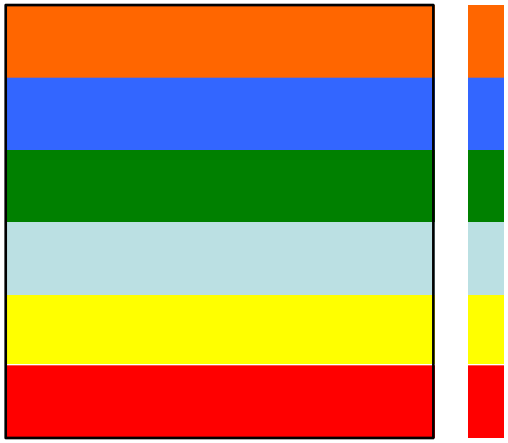
NNZ/process
min: 1.05E+06
max: 1.07E+06
avg: 1.06E+06
max/avg: 1.01

# Messages
total: 4032
max: 63

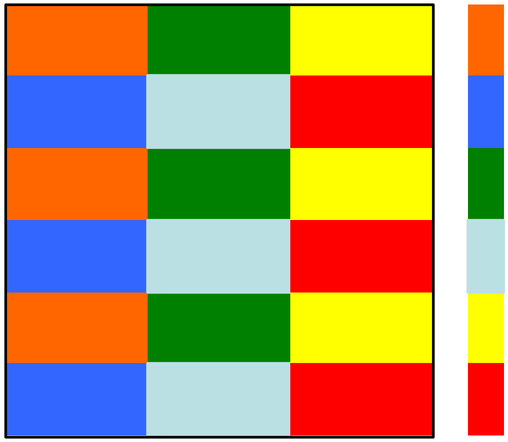
Volume
total: 5.48E+07
max: 8.62E+05

Goal: Reduce number of messages

- 1D distribution:
  - Entire rows (or columns) of matrix assigned to a processor
- 2D distribution:
  - Cartesian methods: Each process owns intersection of some rows & columns
  - Processes are *logically* arranged in a 2D grid
  - Limits max #messages per process to *O(sqrt(#processors))*
  - Long used in parallel dense solvers (ScaLapack)
  - Beneficial also for sparse matrices (Fox et al. ‘88, Lewis & van de Geijn ‘93, Hendrickson et al. ’95)
  - Yoo et al. (SC’11) demonstrated benefit over 1D layouts for eigensolves on scale-free graphs



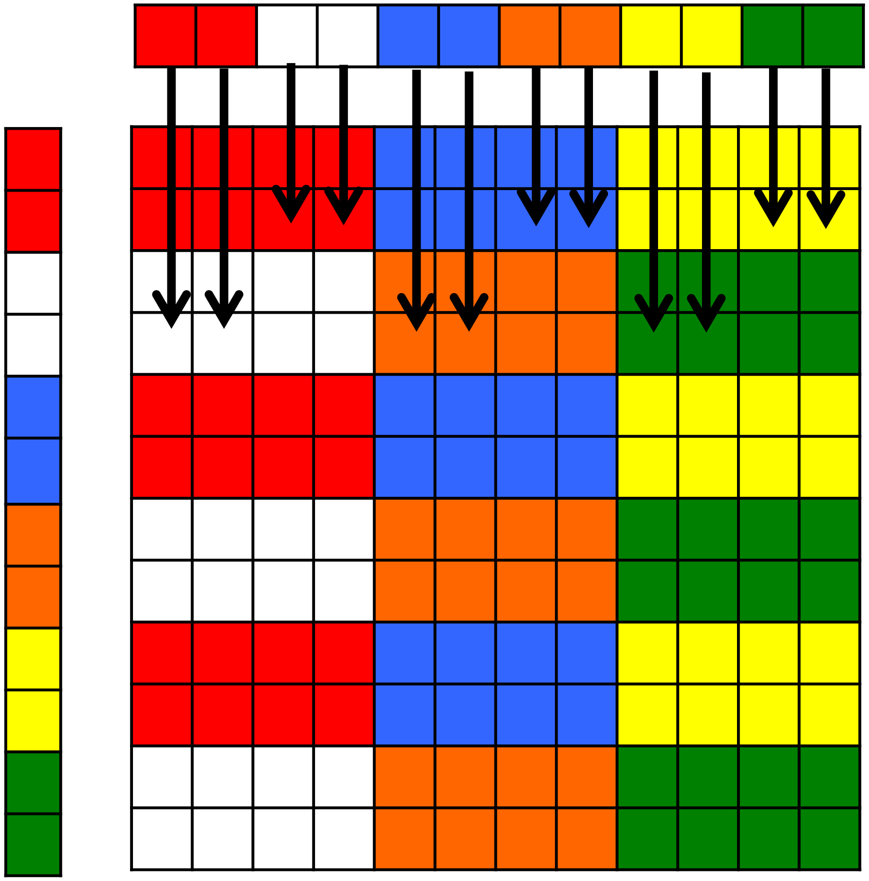
1D row-wise matrix distribution; 6 processes



2D matrix distribution; 6 processes

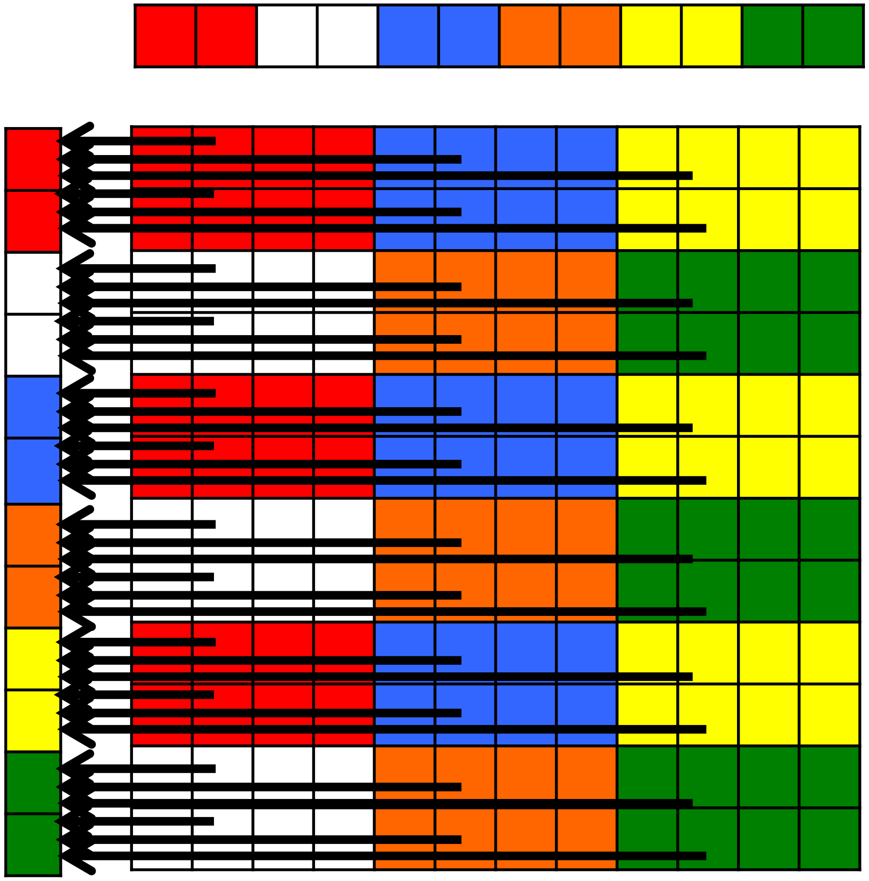
Benefit of 2D Matrix Distribution in SpMV

- During matrix-vector multiplication (y=Ax), communication occurs only along rows or columns of processors.
  - Expand (vertical):
    Vector entries x_j sent to column processors to compute local product $y^p = A^p x$
  - Fold (horizontal):
    Local products y^p summed along row processors; $y = \Sigma y^p$
- In 1D, fold is not needed, but expand may be all-to-all.



Benefit of 2D Matrix Distribution

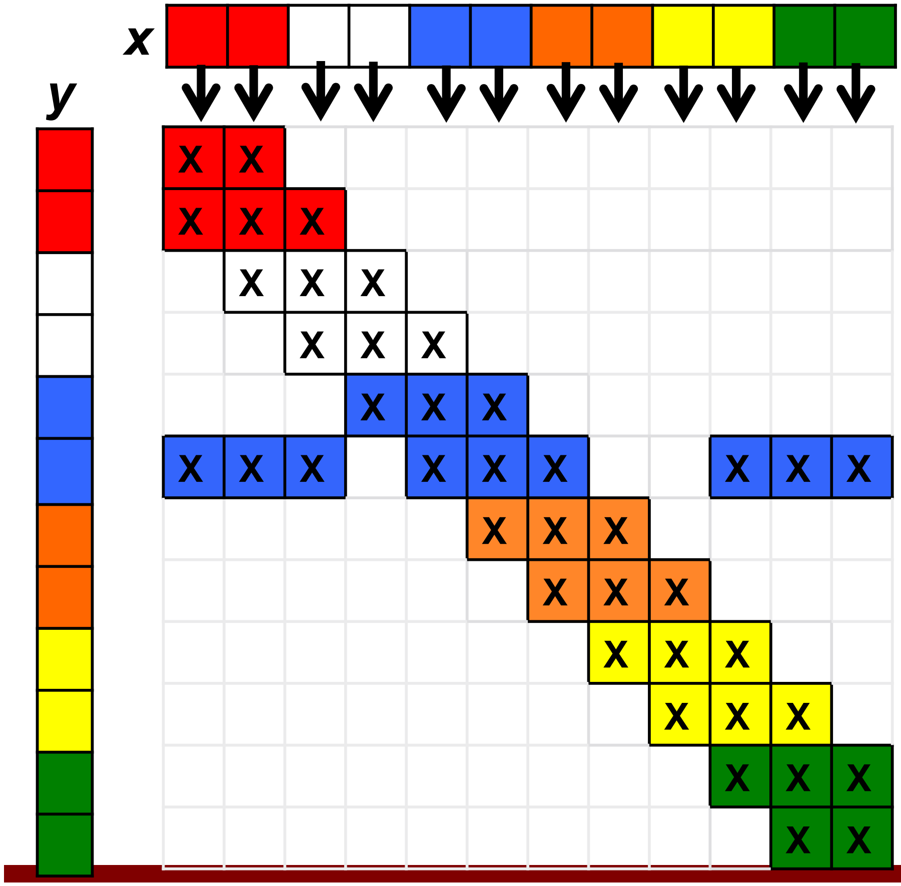
- During matrix-vector multiplication, communication occurs only along rows or columns of processors.
  - Expand (vertical): Vector entries x_j sent to column processors to compute local product $y^p = A^p x$
  - Fold (horizontal): Local products y^p summed along row processors; $y = \Sigma y^p$
- In 1D, fold is not needed, but expand may be all-to-all.



Trilinos’ maps support both 1D and 2D distributions

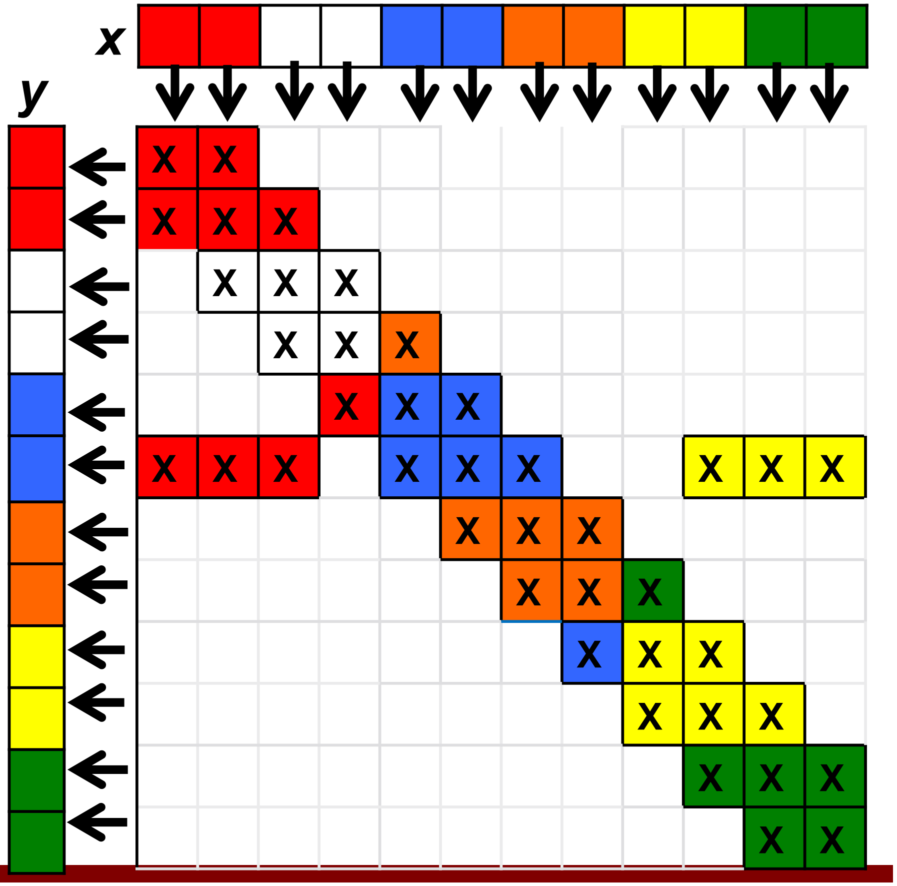
1D Map

Rank 2 (Blue)
Row Map = {4, 5}
Column Map = {0, 1, 2, 3, 4, 5, 6, 9, 10, 11}
Range/Domain Map = {4, 5}



2D Map

Rank 2 (Blue)
Row Map = {4, 5, 8}
Column Map = {4, 5, 6, 7}
Range/Domain Map = {4, 5}



2D Partitioning of Social Network

- Drastic reduction in max number of messages and SpMV time
  - Even with expand & fold, max number of messages is smaller
- Communication volume high with 2D partitions
  - Ignoring the non-zero structure of the matrix.
  - Can we exploit it as we did with 1D-GP?

liveJournal: 4M rows; 73M nonzeros; Max 15K nz/row; Avg 18 nz/row
1024 processes

Method	Imbalance in nonzeros (Max/Avg per proc)	Max # Messages per SpMV	Comm. Vol. per SpMV (doubles)	100 SpMV time (secs)
1D-Block	12.8	1023	34.5M	14.72
1D-Random	1.3	1023	66.3M	14.00
1D-GP	1.2	1011	18.9M	12.17
2D-Block	11.4	62	43.4M	1.31
2D-Random	1.0	62	64.2M	0.97

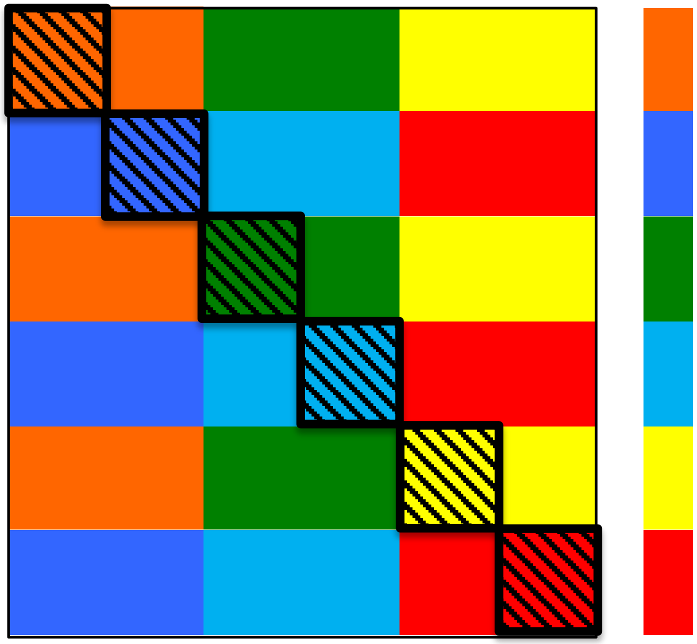
The twist: 2D + Graph Partitioning

- Existing research into direct 2D partitioning of nonzeros (treat nonzeros as graph/hypergraph vertices)
  - Catalyurek & Aykanat; Vastenhouw & Bisseling
  - Much larger problem → very expensive
  - Only serial software available
- Our idea: Apply parallel graph partitioning and 2D distribution together
  - Compute 1D-GP row (vertex) partition of matrix (graph)
  - Apply 2D distribution to the resulting permuted matrix (graph)
- Advantages:
  - Balance the number of nonzeros per process,
  - Exploit structure in the graph to reduce communication volume, AND
  - Reduce the number of messages via 2D distribution
- Don’t optimize a single objective but try do fairly well in all

2D Graph Partitioning (2D-GP)

- Partition rows (vertices) of original matrix (graph) into p parts
  - Using standard graph partitioner
- Implicitly, let $A_{perm} = PAP^T$
  - Where P is permutation from partitioning above
- Assign A_{perm} to processes using Cartesian block 2D layout

Due to partitioning, diagonal blocks of A_{perm} will be denser:



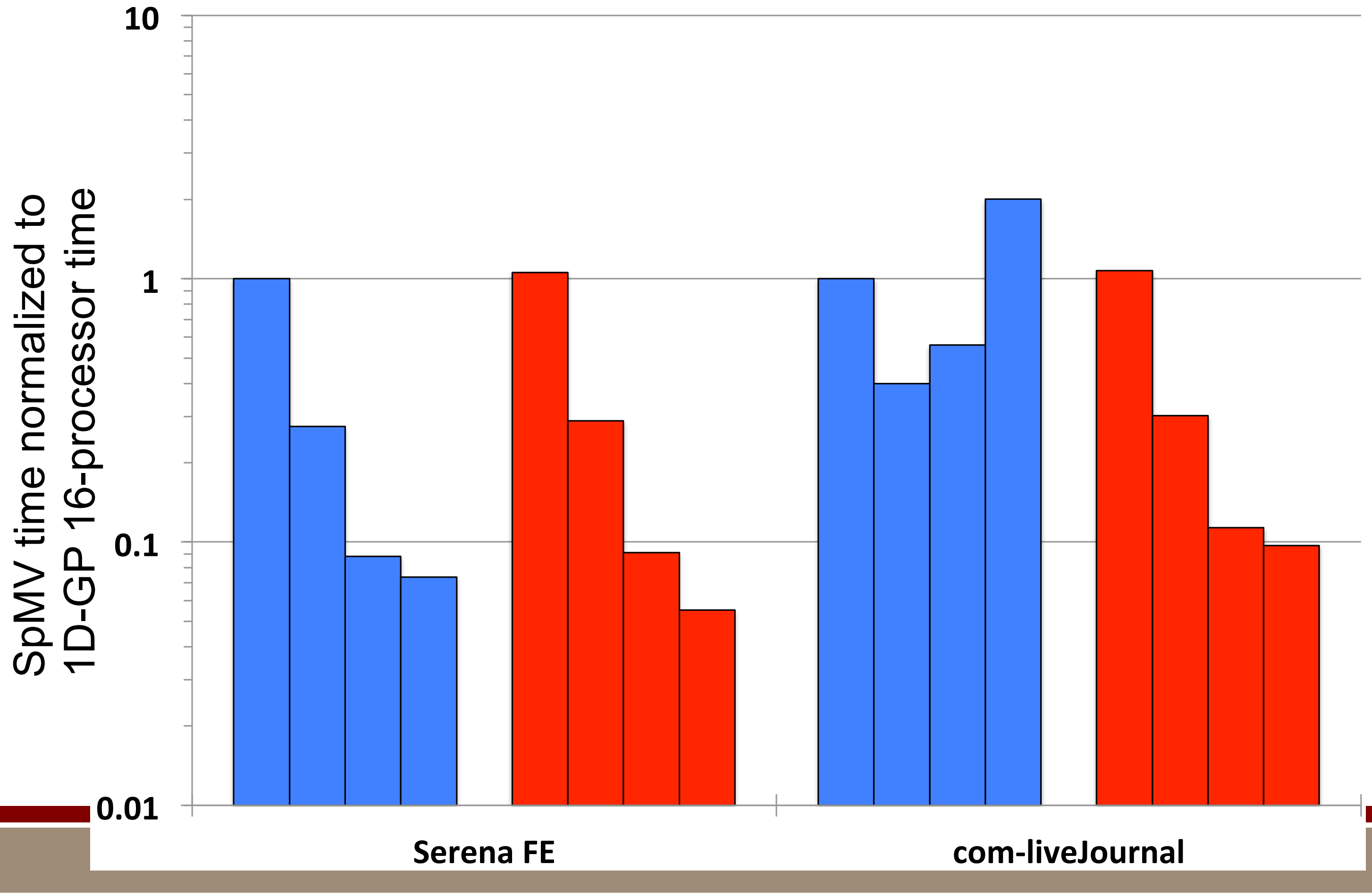
Results 1D vs 2D (Block, Random, GP)

- With 2D-GP,
  - Low number of messages as with 2D-Block, 2D-Random
  - Reduced communication volume due to using structure of matrix
  - Reduced SpMV execution time

liveJournal: 4M rows; 73M nonzeros; Max 15K nz/row; Avg 18 nz/row 1024 processes				
Method	Imbalance in nonzeros (Max/Avg per proc)	Max # Messages per SpMV	Comm. Vol. per SpMV (doubles)	100 SpMV time (secs)
1D-Block	12.8	1023	34.5M	14.72
1D-Random	1.3	1023	66.3M	14.00
1D-GP	1.2	1011	18.9M	12.17
2D-Block	11.4	62	43.4M	1.31
2D-Random	1.0	62	64.2M	0.97
2D-GP	1.4	62	22.4M	0.59

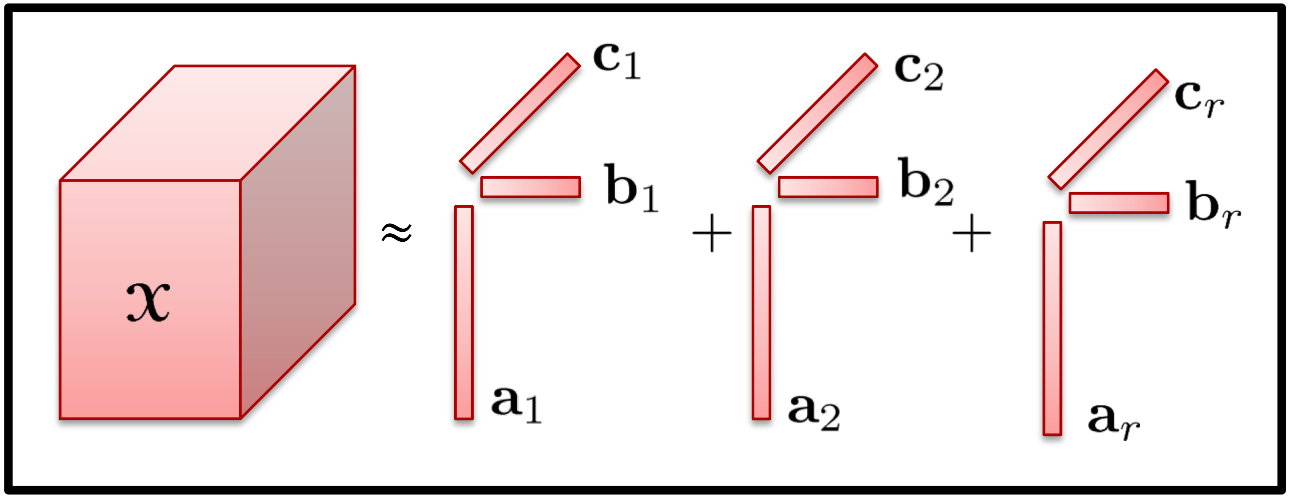
Strong scaling: 1D-GP vs 2D-GP

- Performance for fixed problem as increase number of processors
- For each matrix:
  - **Blue = 1D-GP on 16, 64, 256, 1024 processors (left to right)**
  - **Red = 2D-GP on 16, 64, 256, 1024 processors (left to right)**
  - Times are normalized to the 1D-GP 16-processor runtime



Building New Applications: Parallel Sparse Tensor Decomposition

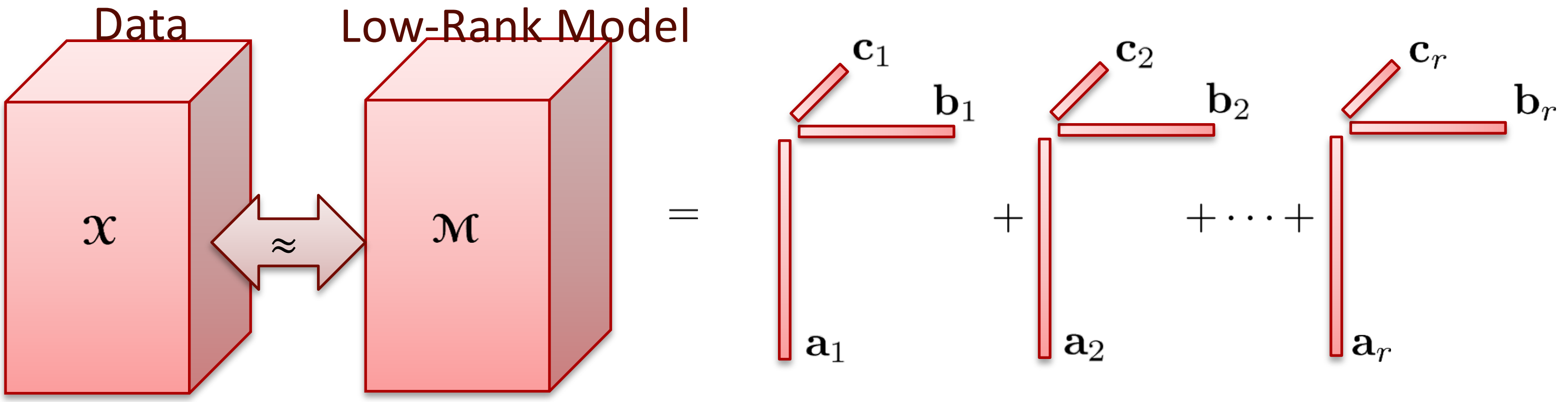
- *Tammy Kolda, Eric Phipps, Karen Devine*
- Goal: Distributed memory parallel sparse tensor decomposition for extremely large tensors



- Approach: Use Trilinos data structures and communication for efficient parallel decomposition

CANDECOMP/PARAFAC Tensor Decomposition

- F. Hitchcock; J.D. Carroll & J-J Chang; R. Harshman
- Seek low-rank approximation of tensor data
- Solve as an optimization problem



$$\underbrace{\mathcal{X} \approx \mathcal{M}}_{m\times n\times p} = \sum_{\alpha=1}^{r} \mathbf{a}_\alpha \circ \mathbf{b}_\alpha \circ \mathbf{c}_\alpha = [\![\underbrace{\mathbf{A}}_{m\times r}, \overset{n\times r}{\mathbf{B}}, \underbrace{\mathbf{C}}_{p\times r}]\!]$$

Optimization Problem: $\min \|\mathcal{X} - \mathcal{M}\|^2 \equiv \sum_i \sum_j \sum_k (x_{ijk} - m_{ijk})^2$

" Tensor Decompositions and Applications," T. Kolda & B. Bader, SIAM Review, 2009

CP-ALS is common solution method

- Solve optimization problem using Alternating Least Squares

$$\min F(\mathbf{A},\mathbf{B},\mathbf{C}) \equiv \tfrac{1}{2}\|\mathcal{X}-\mathcal{M}\|^2 \text{ subject to } \mathcal{M} = [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$$

$$\begin{aligned}
\frac{\partial F}{\partial \mathbf{A}} &= \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B}) - \mathbf{A}(\mathbf{C}^{\intercal}\mathbf{C} * \mathbf{B}^{\intercal}\mathbf{B})\\
\frac{\partial F}{\partial \mathbf{B}} &= \mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A}) - \mathbf{B}(\mathbf{C}^{\intercal}\mathbf{C} * \mathbf{A}^{\intercal}\mathbf{A})\\
\frac{\partial F}{\partial \mathbf{C}} &= \mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A}) - \mathbf{C}(\mathbf{B}^{\intercal}\mathbf{B} * \mathbf{A}^{\intercal}\mathbf{A})
\end{aligned}$$

- Repeat until converged:

$$\begin{aligned}
\mathbf{A} &\leftarrow \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})(\mathbf{C}^{\intercal}\mathbf{C} * \mathbf{B}^{\intercal}\mathbf{B})^{-1}\\
\mathbf{B} &\leftarrow \mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A})(\mathbf{C}^{\intercal}\mathbf{C} * \mathbf{A}^{\intercal}\mathbf{A})^{-1}\\
\mathbf{C} &\leftarrow \underbrace{\mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A})}(\mathbf{B}^{\intercal}\mathbf{B} * \mathbf{A}^{\intercal}\mathbf{A})^{-1}
\end{aligned}$$

Most expensive part of computation:
MTTKRP: Matricized Tensor Times Khatri-Rao Products

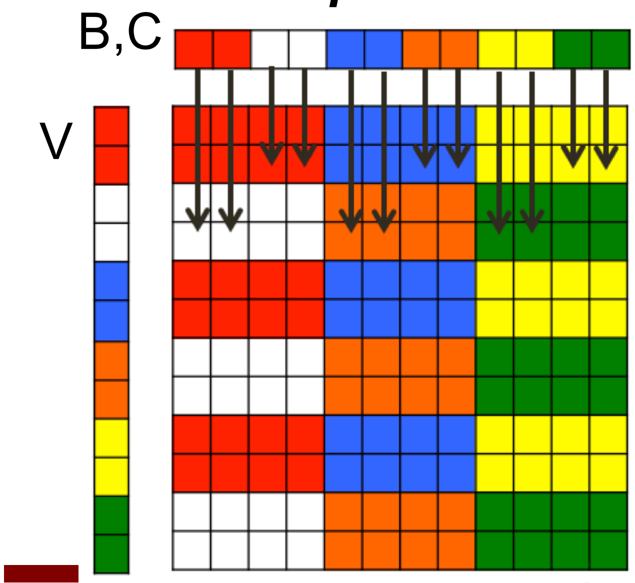
$$\boldsymbol{V} = \mathcal{X}_{(1)}(\boldsymbol{C}\odot\boldsymbol{B}) \implies v(i,l) = \sum_{(i,j,k)\in\mathcal{N}(\mathcal{X})} x(i,j,k)b(j,l)c(k,l), \quad l = 1,\dots,R$$

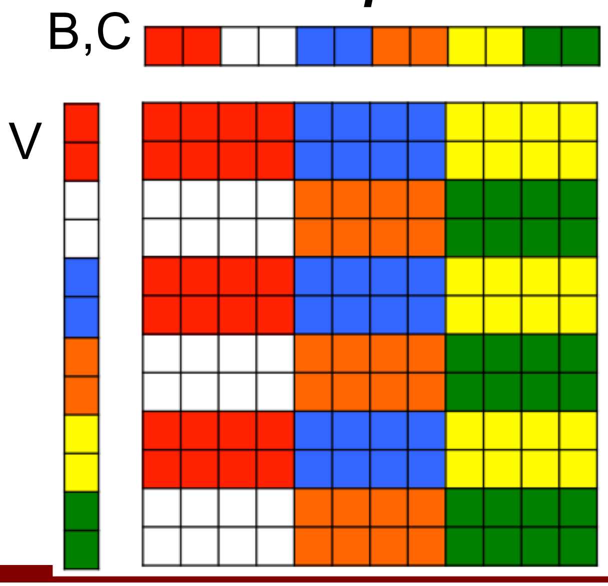
"A Scalable Optimization Approach for Fitting Canonical Tensor Decompositions," E. *Acar, D. Dunlavy, T. Kolda, J. Chemometrics., 2011*

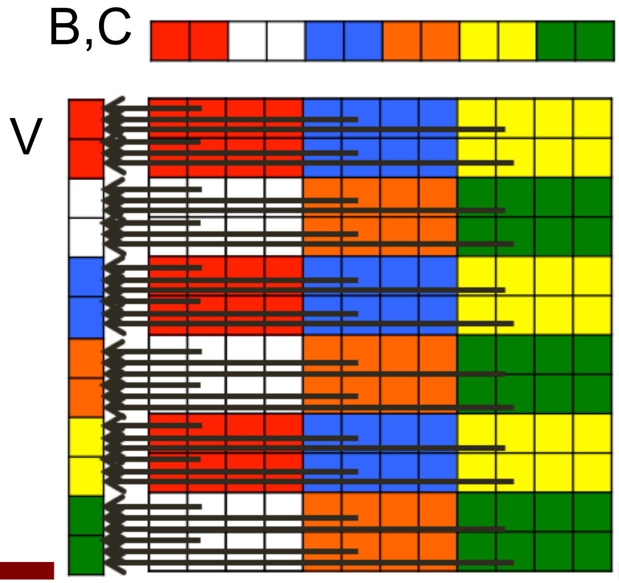
Parallel MTTKRP looks a lot like SpMV

$$\boldsymbol{V} = \boldsymbol{\mathcal{X}}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B}) \implies v(i,l) = \sum_{(i,j,k)\in\mathcal{N}(\boldsymbol{\mathcal{X}})} x(i,j,k)b(j,l)c(k,l), \quad l = 1,\ldots,R$$

- Given:
  - Distribution of tensor (matrix) nonzeros to processors
  - Distribution of factor-matrix (vector) entries to processors
- Expand: Import factor-matrix (vector) entries to processors as needed
- Perform local computation with owned tensor (matrix) nonzeros
- Fold: Export local product values to factor matrix (vector)


Expand


Compute


Fold

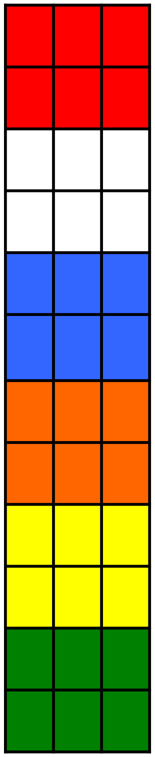
Sparse tensors use Trilinos' Tpetra Maps to describe parallel distribution

- Nonzeros stored in coordinate format
- One Tpetra::Map per tensor mode
  - *Analogous to row/column map in SpMV*
  - Built from indices in coordinate-format storage
  - E.g., nonzero x_{ijk} of 3-mode tensor has entry
i in mode 0 map,
j in mode 1 map,
k in mode 2 map
  - Not necessarily one-to-one
    - Many processors may have a given index *j* in their mode 1 map
- Store only single copy of sparse tensor
  - Each nonzero stored on only one processor

Factor Matrices use Trilinos' Tpetra::MultiVector

- Factor Matrix
  - Dense rectangular *NxR* matrix
  - Entries distributed w.r.t. *N* across all processors
- Tpetra::MultiVector
  - Designed to support Block Krylov linear solvers
  - Class consisting of *R* distributed vectors of length *N*
  - Entries distributed w.r.t. *N*
    - e.g., sub-multivectors assigned to processors
  - Tpetra::Map describes parallel distribution of MultiVector
    - Map is one-to-one; i.e., each MultiVector entry stored on only one processor
  - MultiVector class provides operations (norms, initialization, etc.) needed in tensor decomposition

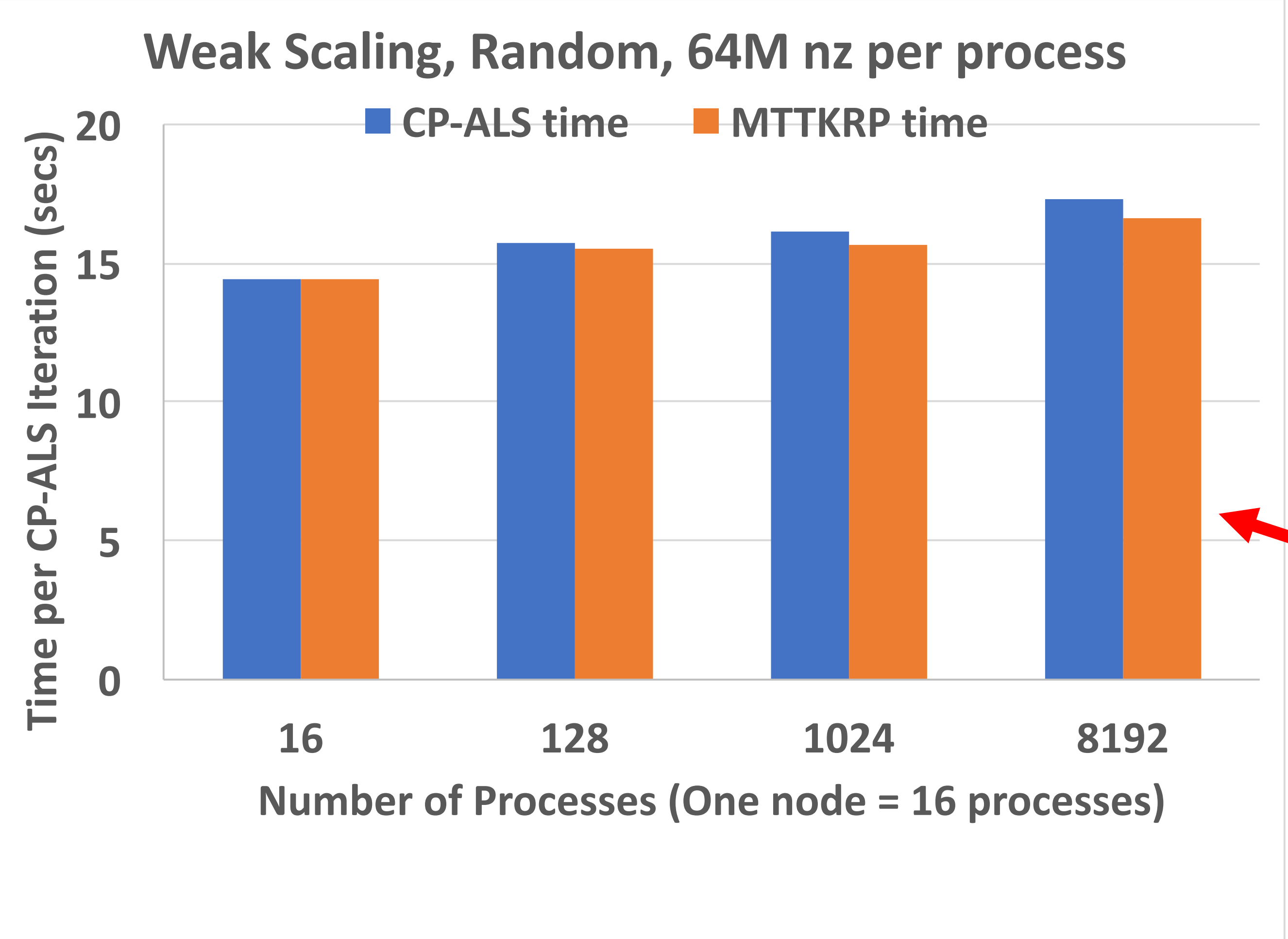
Rank 3, length 12 factor matrix distributed across 6 processors



All communication performed in Trilinos' Tpetra classes

- Communication operations: expand and fold of factor matrix entries are the same as those in SpMV
- Tpetra::Import/Export
  - Built based on two maps: Factor matrix map and corresponding tensor map
  - Point-to-point MPI Isend/Irecv
- All other communication: MPI_Allreduce
  - Convergence tests, contributions to replicated Gramian matrices, etc.

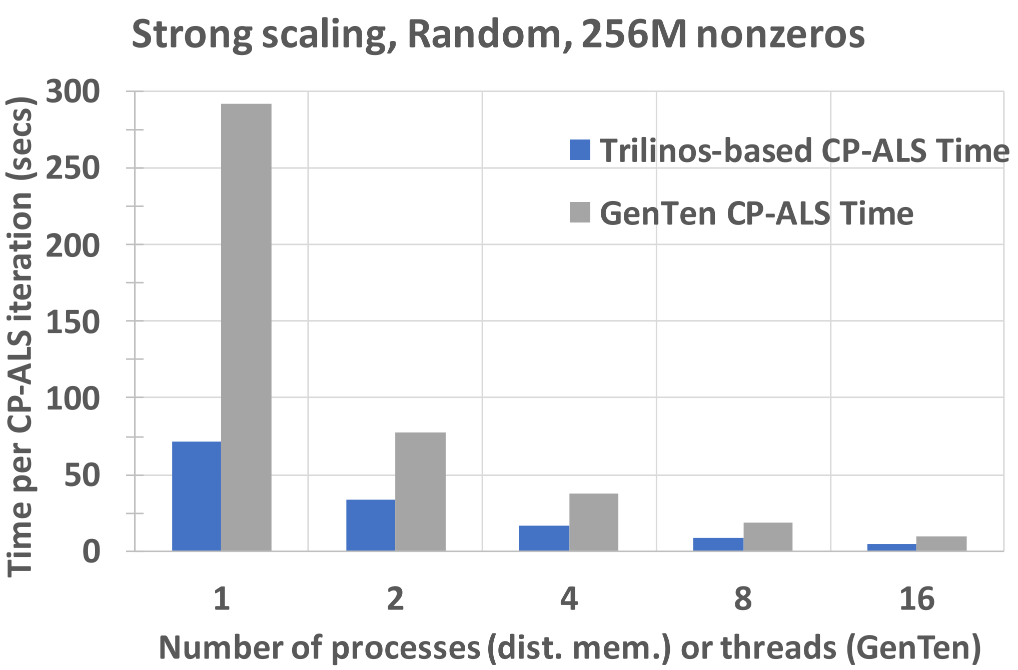
Scalable communication enables decomposition of huge problems



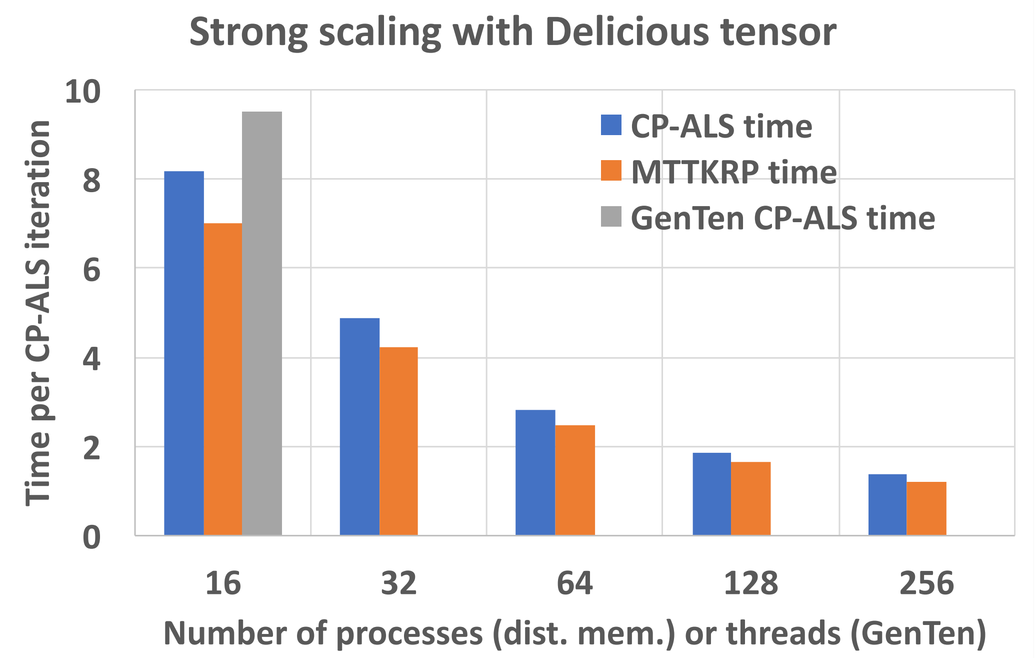
- 4D Random tensor
  - 64M nonzeros per process
  - Constant nonzero density 0.001024
  - SkyBridge cluster (2.6 GHz Intel Sandy Bridge with Infiniband)
  - **12.6 Terabyte** tensor on 8192 MPI processes
    - 524 B nonzeros
    - Four integer indices per nonzero
    - One double value per nonzero

Good strong scaling in MPI-only Trilinos-based CP-ALS

- **Genten** (Phipps, Kolda; SNL) CP-ALS on single Skybridge node
  - Kokkos-based code with OpenMP, CUDA backends
  - **Single implementation runs on CPUs *and GPUs*** *(key advantage)*
- Random tensor
  - 1000 x 1000 x 500 x 500
  - 256M nonzeros; rank=16
- Delicious (FROSTT collection):
  - 533K x 17M x 2.5M x 1.4K
  - 140M nonzeros; rank=10



Scaling not hurt by MPI



Strong scaling extends beyond single node

Conclusions

- Investment in HPC scientific libraries can be leveraged for data sciences
  - Directly to explore new areas (e.g., hypergraph clustering in Grafiki)
  - With a twist (e.g., 2D matrix distributions)
  - Enabling new applications (e.g., sparse tensor decomposition)

- Another whole talk: Kokkos and KokkosKernels
  - Kokkos performance portable data structures and parallel execution constructs (Trott, Edwards, et al.)
  - KokkosKernels graph and matrix operations using Kokkos (Rajamanickam et al.)
  - Single code compiles/executes well on CPUs, KNLs, GPUs
  - Examples:
    - *GenTen sparse tensor decomposition (Phipps, Kolda)*
    - Triangle counting (Wolf, Deveci, Berry, Hammond, Rajamanickam)
    - SpGEMM (Deveci, Trott, Rajamanickam)
  - “High-Performance Portable Data Analytics Software Using Kokkos” Michael Wolf, 2018 Chesapeake Large-Scale Analytics Conference

- Thanks to
  - The organizers for inviting me
  - Erik Boman
  - Mark Hoemmen
  - Tammy Kolda
  - Eric Phipps
  - Chris Siefert
  - Daniel Dunlavy
  - Alicia Klinvex
  - Mauro Perego
  - Siva Rajamanickam
  - Michael Wolf

- Trilinos:
  - https://github.com/trilinos/Trilinos
  - https://trilinos.github.io/

- Karen Devine: kddevin@sandia.gov