SAND2014-19463PE

Exceptional service in the national interest

Sandia National Laboratories

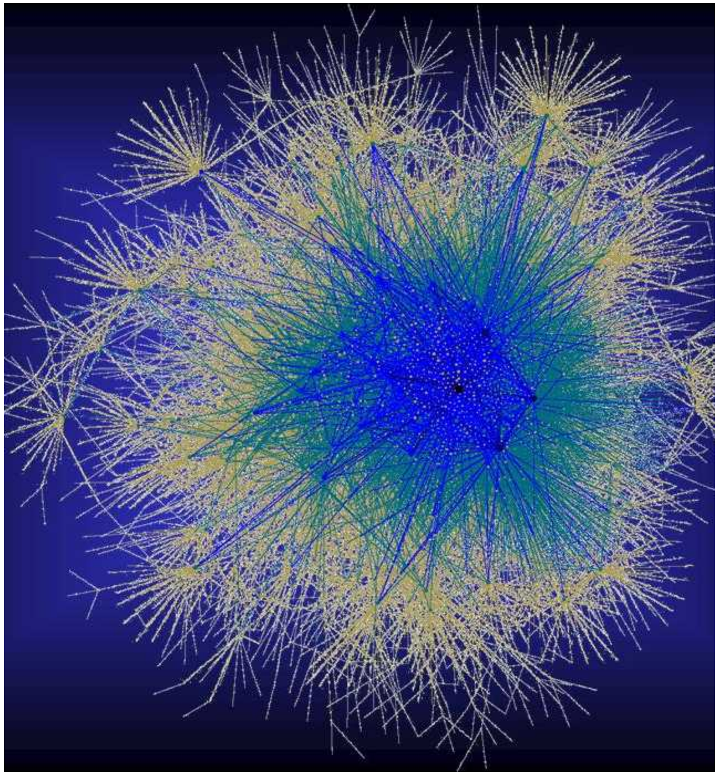
2D Partitioning for Scalable Matrix Computations on Scale-Free Graphs

Erik Boman, Karen Devine, Sivasankaran Rajamanickam
Sandia National Laboratories

Dagstuhl, Nov. 2014

Introduction

- *Big data* is a challenge to HPC
  - Large graphs/networks are pervasive
  - E.g., WWW, social networks
    - Scale-free, small-world
    - Skewed degree distribution
    - Very different from PDE discretizations
- How to do efficient parallel computations?
  - Using distributed-memory computers
  - Data layout is important



BGP graph (credit: Richardson, Chung)
http://math.ucsd.edu/~fan/graphs/gallery

Key Message

- 2D (edge-based) partitioning is important for data analytics.
  - Scale-free, small-world, skewed degree distribution.
  - Long predicted, but demonstrated only recently, and not yet widely appreciated.

- Computing a “good” 2D distribution is no harder than 1D!
  - Still active research area.

Sparse matvec (SpMV) important

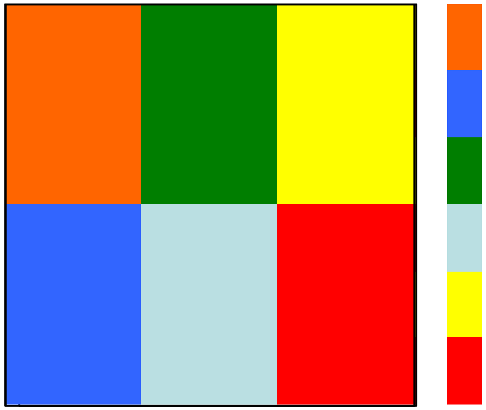
- Linear algebra is a useful analysis tool for graphs
  - Both adjacency matrix and graph Laplacian are of interest
  - Spectral analysis using extreme eigenpairs
  - SpMV is core kernel in iterative methods
- SpMV is bottleneck for scale-free graphs on large distributed-memory computers
  - Example: For a social network (orkut) on 64 cores
    - SpMV took 95% of the compute time in an eigensolver
  - Maximum #messages, over all processes, is typically p-1
    - Due to some very high-degree vertices

1D and 2D Matrix Distributions

We view graphs as sparse matrices.

- 1D (vertex) distribution:
  - Entire rows (or columns) of matrix assigned to a processor
  - Required in most software
- 2D (edge) distribution:
  - Cartesian methods: Each process owns intersection of some rows & columns
  - Processes are *logically* arranged in a 2D grid
  - This limits #messages per process to $O(\sqrt{p})$
  - 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

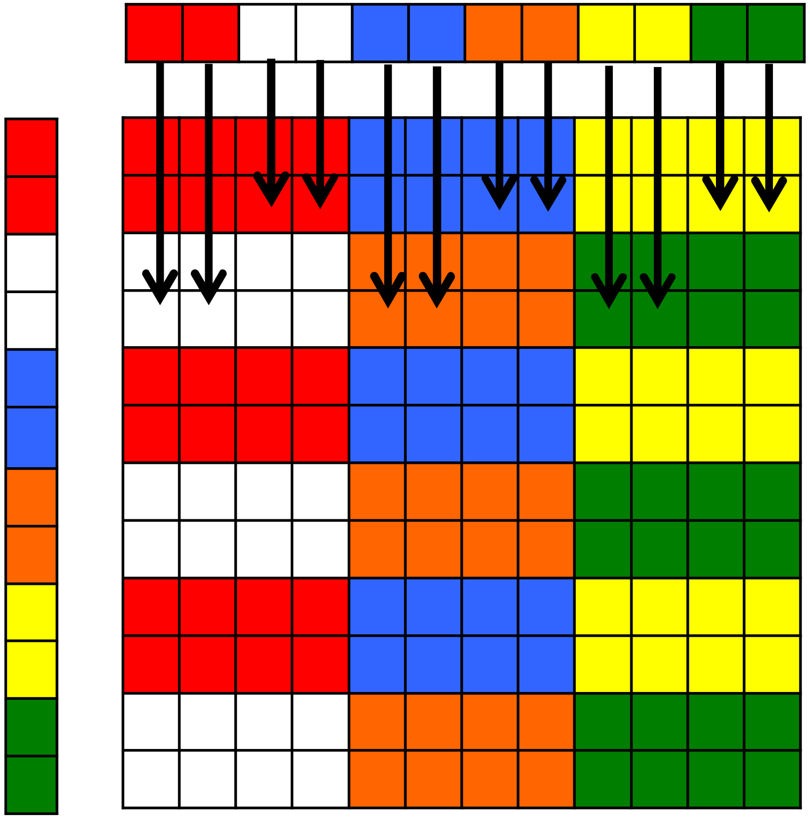
1D row-wise matrix distribution; 6 processes



2D matrix distribution; 6 processes

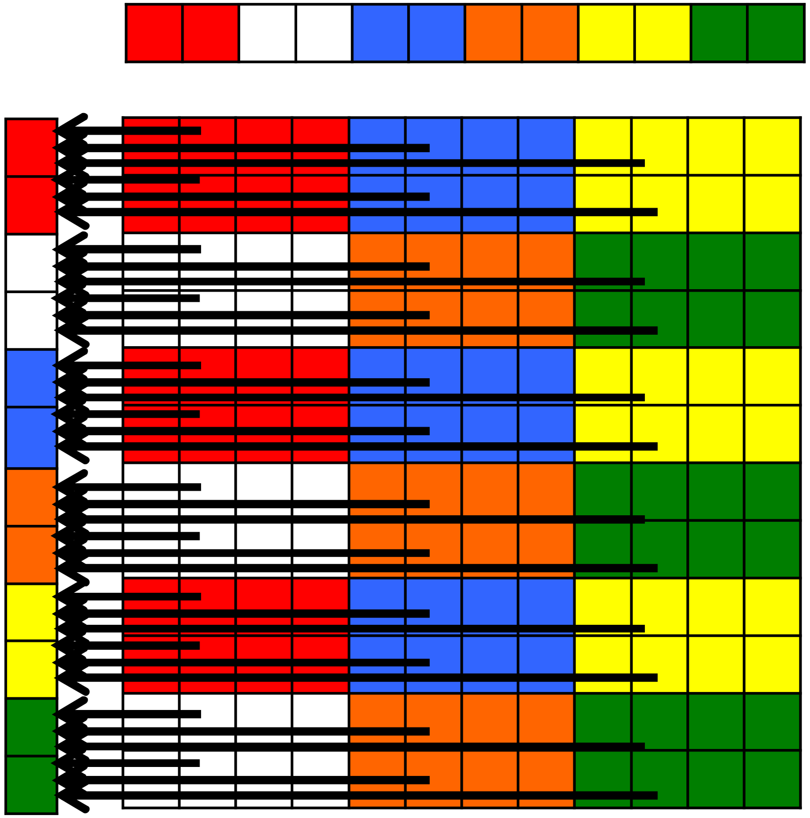
Benefit of 2D Matrix Distribution

- 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.



2D Partitioning Methods

- Cartesian 2D block (Fox et al. '88)
  - Simple/fast to compute but ignores the structure of the graph.
  - Low #messages, but communication volume may be high.
- Coarse-grain hypergraph (Catalyurek & Aykanat '01)
  - Cartesian product, but uses the matrix structure.
  - Requires multiconstraint hypergraph partitioning.
- Fine-grain hypergraph (Catalyurek & Ayk. '01)
  - Assign each nonzero separately
  - Not Cartesian, high #messages
  - Larger hypergraph, impractical for big problems
- Mondriaan (Vastenhouw & Bisseling '05)
  - Recursive bisection, hypergraph partitioning
  - Not Cartesian, no bound on #messages

Trilinos Computational Science Toolkit

- Collection of ~60 packages
  - Heroux et al., Sandia
- Trilinos Capabilities:
  - Scalable Linear & Eigen Solvers
  - Discretizations, Meshes & Load Balancing
  - Nonlinear & Optimization Solvers
  - Software Engineering Technologies & Integration
- In this project, we used
  - Distributed Matrix/Vector classes *Epetra*
  - Partitioning package *Zoltan*
  - Eigensolver package *Anasazi*

Petra Object Model

- Maps describe the distribution of global IDs for rows/columns/vector entries to processors.
- Four maps needed in most general case:
  - Row map for matrix
  - Column map for matrix
  - Range map for vector
  - Domain map for vector
- Implemented in *Epetra* (and Tpetra) packages
- Allows 2D distributions!

Load-Balancing by Randomization

- Simple "block" partitioning balances rows but not nonzeros
- Randomization is a simple but powerful technique
- On input, randomly permute matrix rows/columns
  - Eliminates any inherent structure in input file (e.g., high degree nodes first)
  - Gives better balance in number of nonzeros per processor for 1D and 2D
  - But can drastically increase communication volume

liveJournal matrix (4M rows; 73M nonzeros) on 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	2.14
1D-Random	1.3	1023	55.3M	1.52
2D-Block	11.4	62	43.4M	0.95
2D-Random	1.0	62	64.2M	0.43

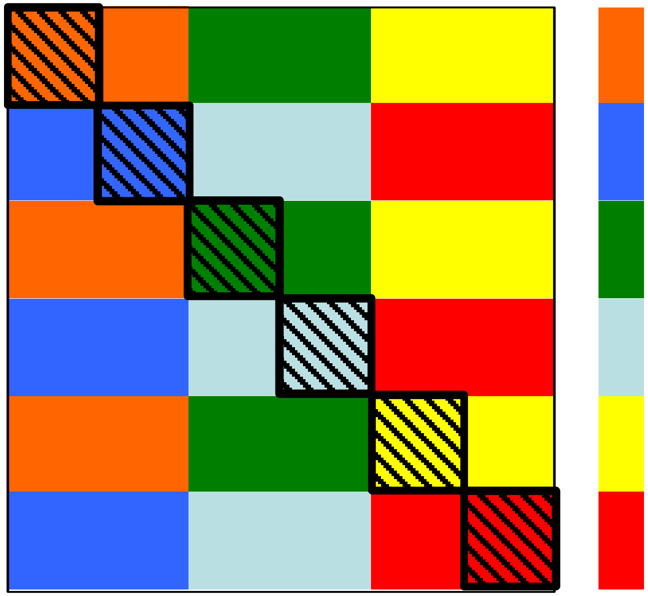
New Method: Graph Partitioning + 2D

- Our idea: Apply (hyper)graph partitioning and 2D distribution together
  - Compute vertex-based partition of graph using ParMETIS or Zoltan
  - Apply 2D distribution to the resulting permuted graph/matrix
- Advantages:
  - Balance the number of nonzeros per process
  - Exploit structure in the graph to reduce communication volume
  - Reduce the number of messages via 2D distribution
- Don’t optimize a single objective but try do fairly well in all

2D (Hyper-)Graph Partitioning (GP/HP)

- Partition vertices of original graph into *p* parts
  - Using standard (hyper)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

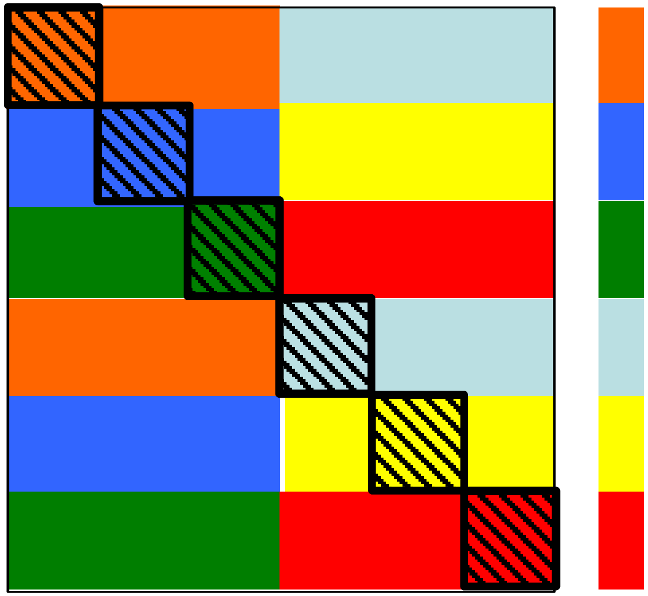
Due to partitioning, diagonal blocks of A_{perm} will be denser:



Observations

- We first partition into *p* parts
  - NOT sqrt(p)
- Many choices for Cartesian 2d layout in second step
  - Fast method: just use (I,j) indices, ignore structure
  - Future: Pick “best” option based on structure

Example: Another possible 2d layout



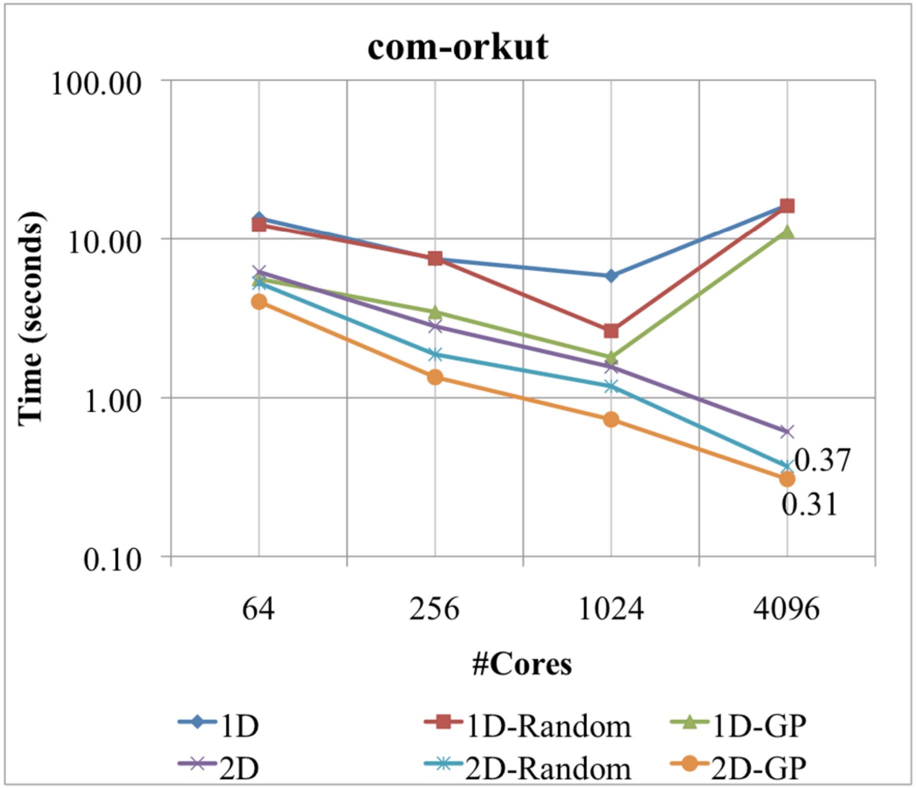
Results 1D vs 2D (Block, Random, GP)

Sandia National Laboratories

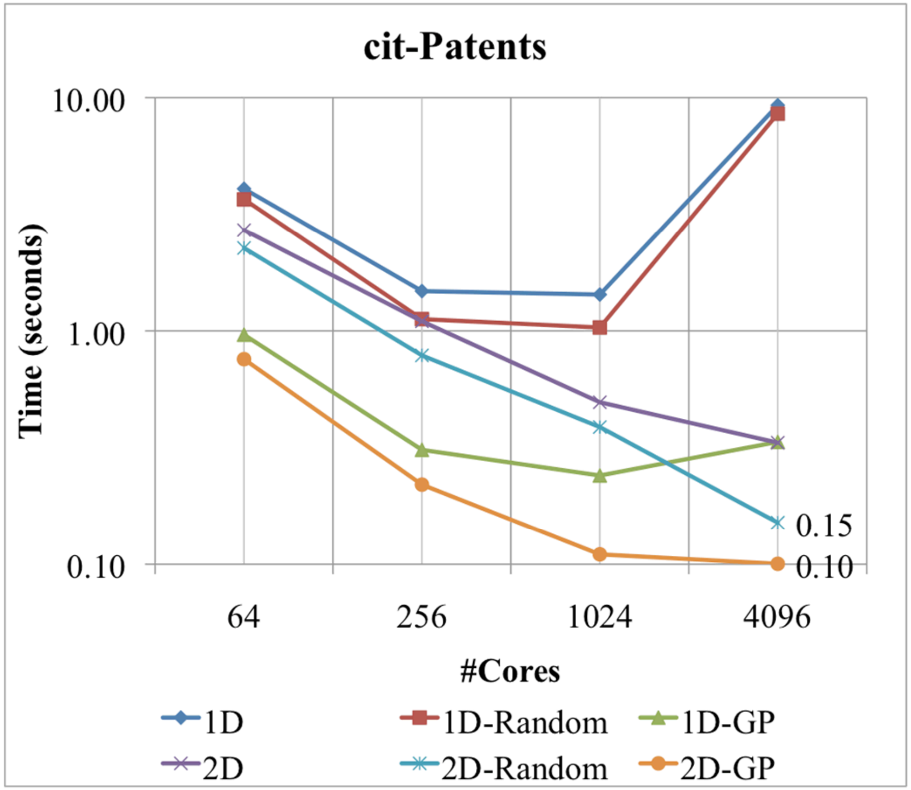
Platform: cab cluster at LLNL (1200 Intel Xeon E5 16-core nodes operating at 2.6 GHz, 32 GB memory/node, Infiniband)
All matrices from UF collection (some originally from SNAP, etc.)

liveJournal matrix (4M rows; 73M nonzeros) on 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	2.14
1D-Random	1.3	1023	55.3M	1.52
1D-GP	1.2	1011	18.9M	0.53
2D-Block	11.4	62	43.4M	0.95
2D-Random	1.0	62	64.2M	0.43
2D-GP	1.4	62	22.4M	0.22

Strong scaling



Orkut social network
3.1M rows; 237M nonzeros
Max nonzeros/row = 33K

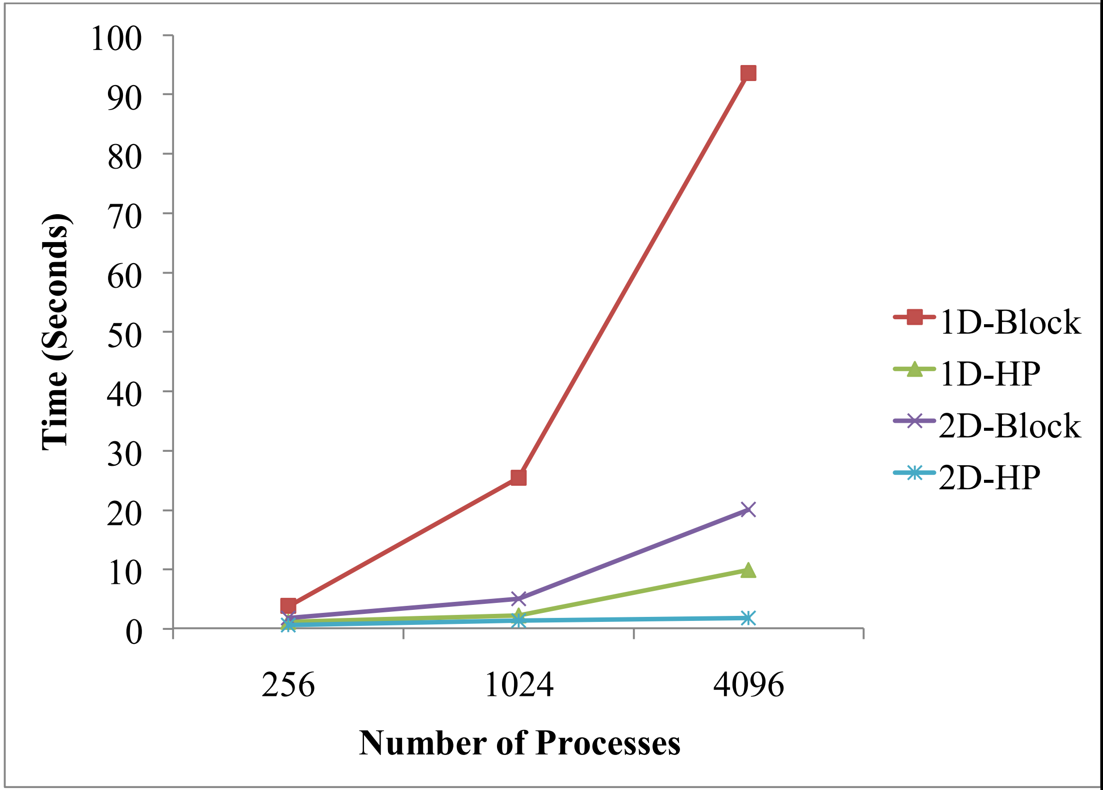


Patent citations network
3.8M rows; 37M nonzeros
Max nonzeros/row = 1K

1D stops scaling around 1024 processes due to high communication cost.

“Weak Scaling”

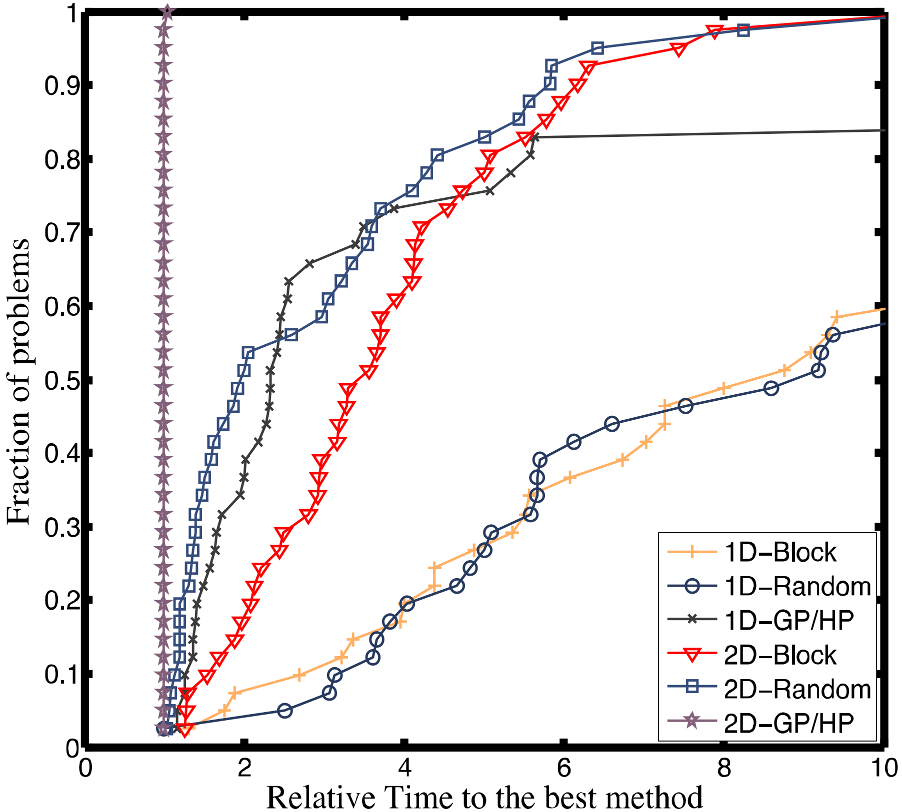
- R-MAT matrices (Chakrabarti et al., 2004) with Graph-500 parameters (a=0.57; b=c=0.19; d=0.05)
  - rmat_22 on 256 procs
    - 4.2M vertices
    - 38M edges
  - rmat_24 on 1024 procs
    - 16.8M vertices
    - 151M edges
  - rmat_26 on 4096 procs
    - 67.1M vertices
    - 604M edges
- Times for 100 SpMV
- 2D-HP maintains best weak scaling.



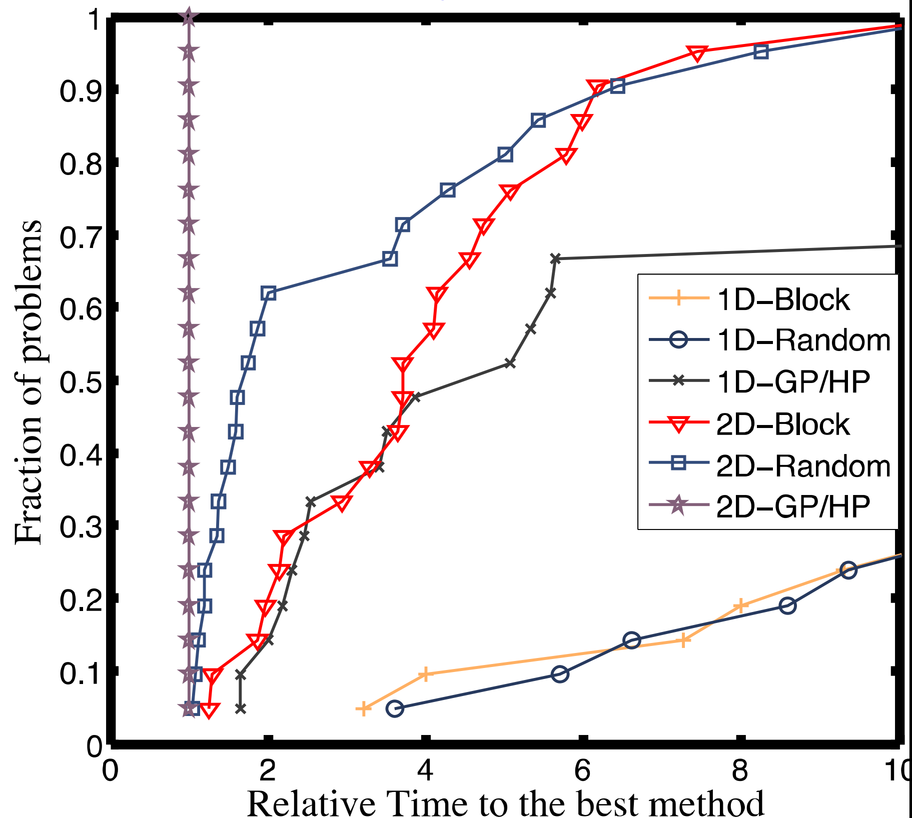
Performance Profile

- 10 matrices: 1.1M - 67.5M rows; 36M-1.6B nonzeros
- 2D-GP/HP best in all but one experiment
- Benefit of 2D even greater for large numbers of processes

All experiments: 64-4096 procs

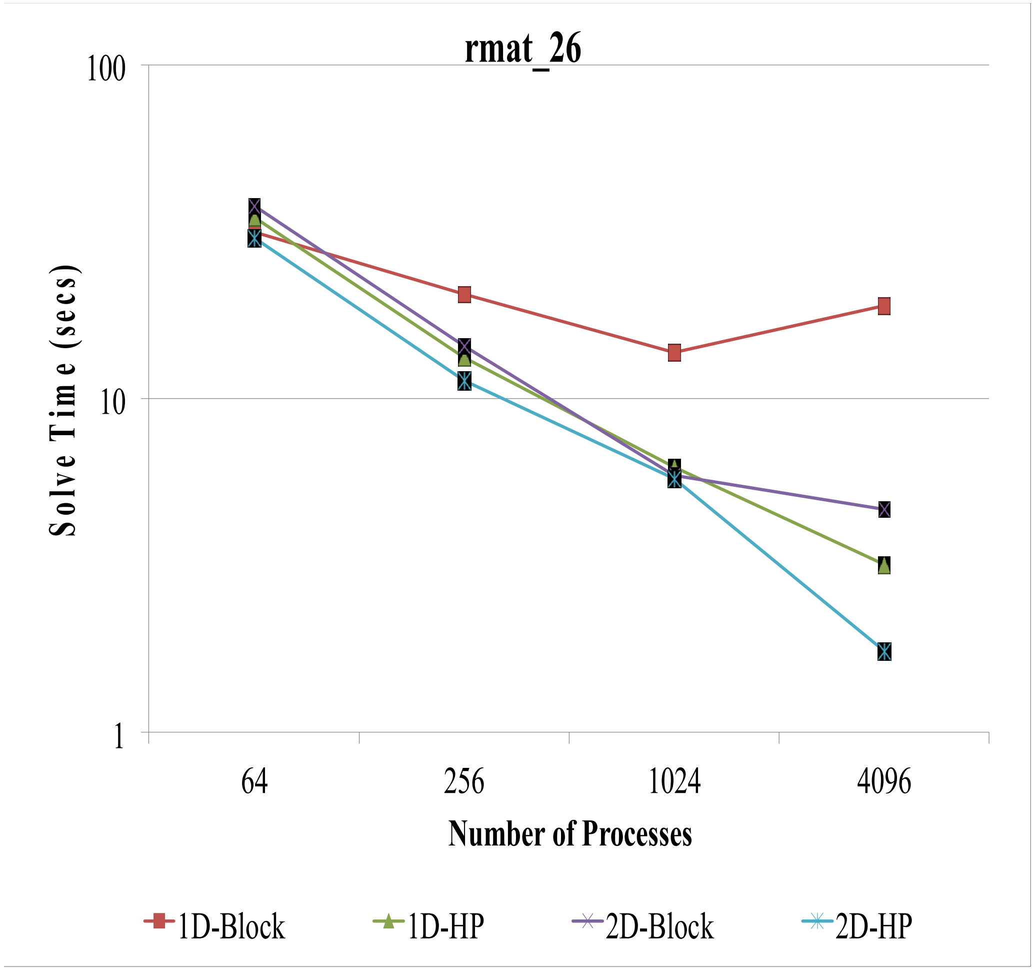


Large runs only: 1024-4096 procs



Eigensolver Experiments

- Anasazi Toolkit in Trilinos
  - Baker, Hetmaniuk, Lehoucq, Thornquist; ACM TOMS 2009
  - Block-based eigensolvers: Solve AX = XΛ or AX = BXΛ
- Experiment:
  - Find 10 largest eigenvalues of Laplacian using Block Krylov-Schur (BKS) solver
  - rmat_26 matrix: 67.1M rows; 604M nonzeros



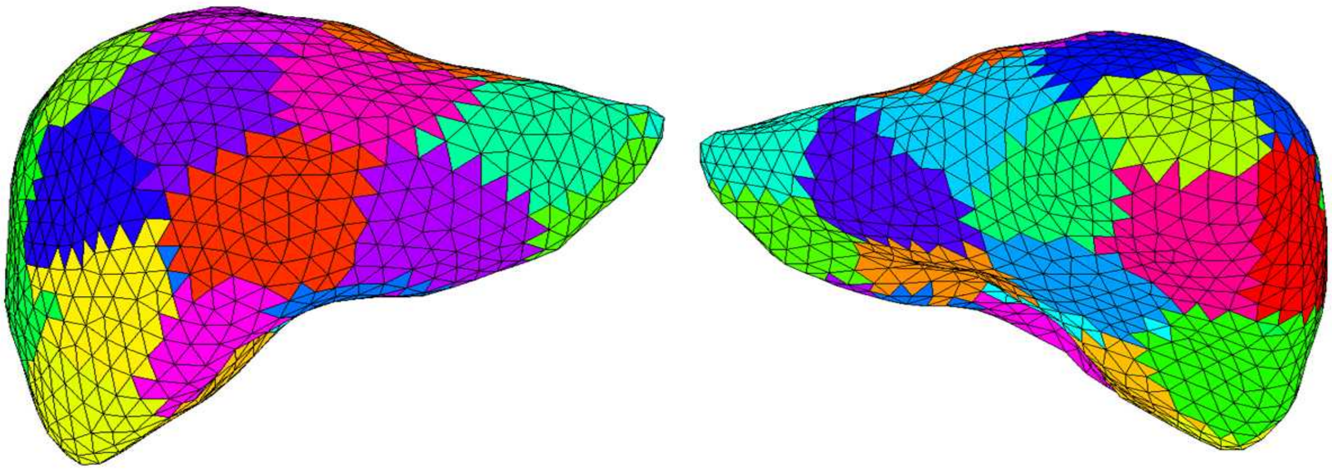
Conclusions

- 2D distributions allow scalable parallel computations for small-world (scale-free) graphs.
  - For 1D, must use “vertex delegates” or “disaggregation”
- 1D (hyper)graph partitioning is effective on scale-free graphs for moderate number of processes.
  - Good load balance, low communication volume
- Combining 2D distribution with (hyper)graph partitioning gives best results.
  - Low number of messages, low communication volume, low imbalance.
  - Allows reuse of existing partitioning software.
- Ongoing/future work:
  - Compare to other 2D partitioning methods.
  - Use faster partitioning method in 1st step (e.g., PULP)
  - Optimize 2nd step in algorithm (Cartesian layout)

Extra Slides

Data Partitioning

- (Hyper-)graph partitioning generally reduces communication for SpMV
- Software tools (e.g., Metis, Scotch, Zoltan) were designed for meshes and PDE discretizations
  - Not optimized for scale-free graphs
  - Focus has been on cut edges and communication volume
    - We also wish to reduce #messages



Test Matrices & Platform

- Compare times for 100 matrix-vector products with 1D and 2D distributions
- Platform: cab cluster at LLNL (1200 Intel Xeon E5 16-core nodes operating at 2.6 GHz, 32 GB memory/node, Infiniband)
- Matrices from the University of Florida matrix collection. (Symmetrized, if needed)

Name	Description	Number of Rows	Number of Nonzeros
Hollywood-2009	Hollywood movie actor network (Boldi, Rosa, Santini, Vigna)	1.1M	113M
Wikipedia-20070206	Links between wikipedia pages (Gleich)	3.5M	85M
Ljournal-2008	LiveJournal social network (Boldi, Rosa, Santini, Vigna)	5.6M	99M
Wb-edu	Links between *.edu webpages (Gleich)	8.9M	88M
Cit-Patents	Citation network among US patents (Hall, Jaffe, Trajtenberg)	3.8M	33M

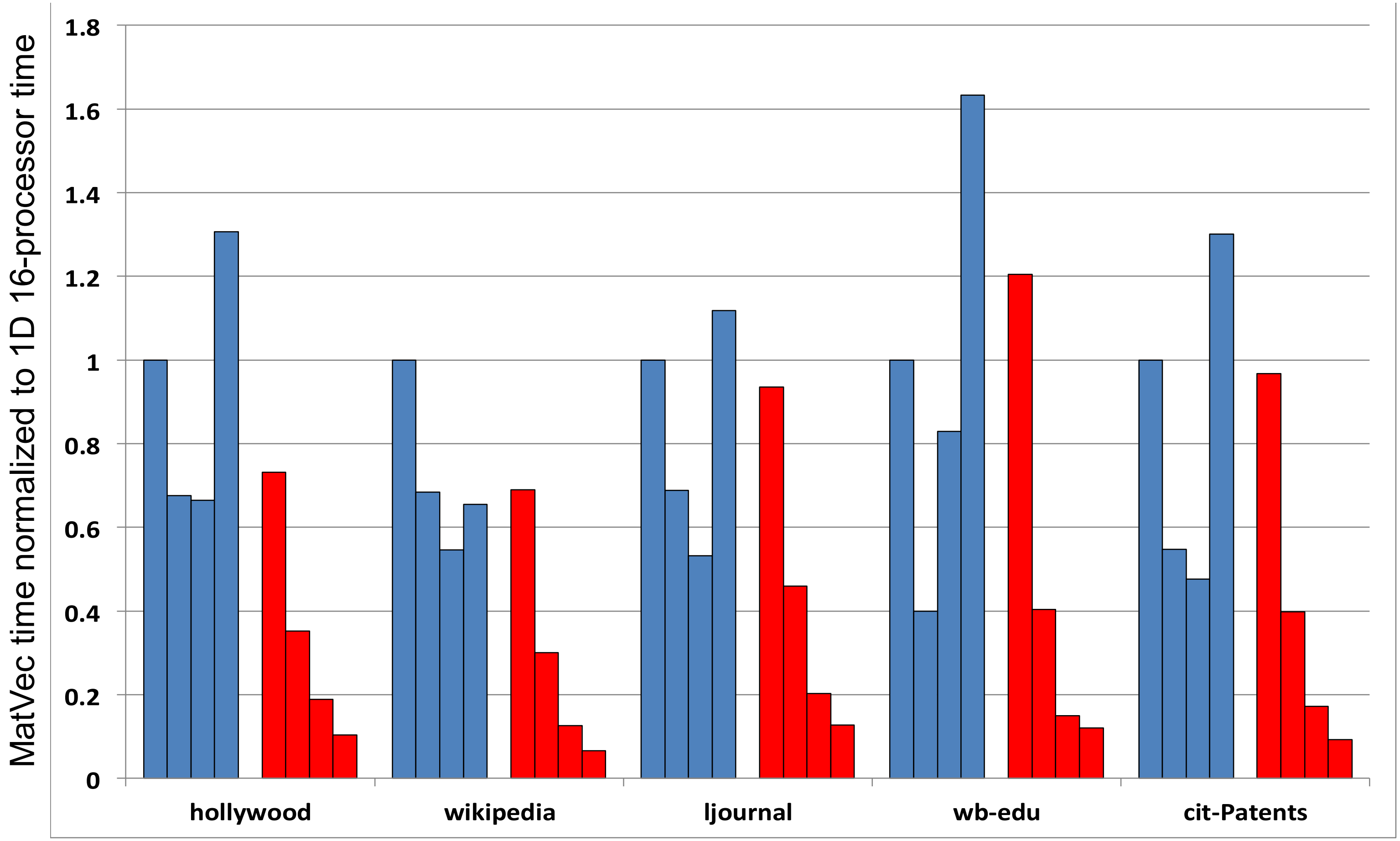
1D vs 2D Strong Scaling experiments

For each matrix:

Blue = Trilinos 1D Matrix Distribution on 16, 64, 256, 1024 processors (left to right)

Red = Trilinos 2D Matrix Distribution on 16, 64, 256, 1024 processors (left to right)

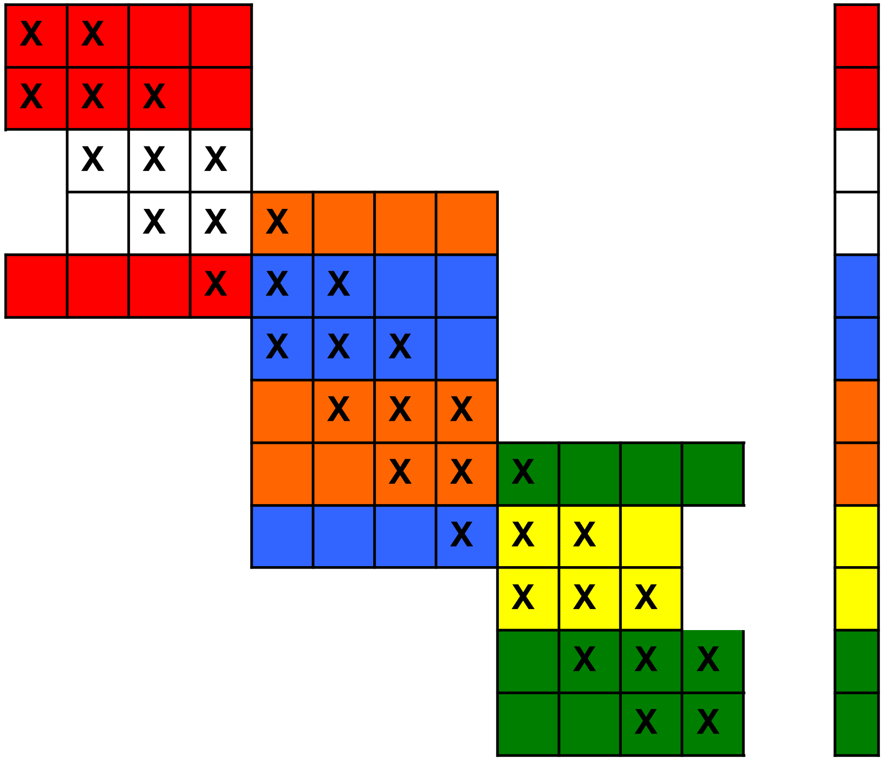
Times are normalized to the 1D 16-processor runtime for each matrix.



Trilinos: Petra Object Model

- Maps describe the distribution of global IDs for rows/columns/vector entries to processors.
- Four maps needed in most general case:
  - Row map for matrix
  - Column map for matrix
  - Range map for vector
  - Domain map for vector
- Implemented in *Epetra* (and Tpetra) packages
- Allows 2D distributions!

Rank 3 (Blue)
Row Map = {4, 5, 8}
Column Map = {4, 5, 6, 7}
Range/Domain Map = {4, 5}



Eigensolver Experiments

- Anasazi Toolkit in Trilinos
  - Baker, Hetmaniuk, Lehoucq, Thornquist; ACM TOMS 2009
  - Block-based eigensolvers: Solve $AX = X\Lambda$ or $AX = BX\Lambda$
- Experiment:
  - Find 10 largest eigenvalues of Laplacian using Block Krylov-Schur (BKS) solver
  - rmat_26 matrix: 67.1M rows; 604M nonzeros
  - HP = Hypergraph partitioning in Zoltan