Hierarchical Sparse Graph Computations on Multicore Platforms

Humayun Kabir, Kamesh Madduri

The Pennsylvania State University

{hzk134,madduri}@cse.psu.edu

October 8, 2017

Our contributions

- **PKC** parallel k-core decomposition algorithm; uses less memory and fewer atomic operations; on average 1.90× faster than state-of-the-art ParK
- **PKT** parallel k-truss decomposition algorithm; uses less memory and carefully designed data structure; on average relative speedup achieved is 9.68×
- **Graph analysis** using above techniques; they are used for coarsening, sparsification, vertex reordering, partitioning and community detection; reordering decreases number of bits needed to compress a graph by over 22% and number of non-empty blocks decrease by over 28% compared to SlashBurn.
- Developed **CSR-k** for sparse matrix-vector multiplication (SpMV); *CSR-k* is 3.82× and 2.12× faster than *MKL* and pOSKI respectively
- Sparse triangular solution (STS) using *CSR-k* is 2.44× and 4.21× faster than coloring and level-set respectively

Sparse Graphs properties

- Big
  - Facebook - 2.01B+ users and 0.3T+ friendships
  - Internet - 51B+ pages index by Google (http://www.worldwidewebsize.com/)
- Large scale sparse graph computations have challenges
  - The memory access pattern of sparse algorithms is irregular
  - Degree distribution is skewed; time taken to process vertices varies
  - Sparse matrix computations do less number floating point operations per memory load

Thesis scope

Design high performance sparse algorithms on share memory systems

- Hierarchical Graph Algorithms
  - Design hierarchical sparse graph algorithms on shared memory systems
  - Use the algorithms to analyze large sparse graphs
- Sparse Matrix Algorithms
  - Design sparse matrix algorithms that enhances locality in memory accesses
  - Design algorithms to decrease work load imbalance

k-core Decomposition

- k-core: a maximal subgraph such that each vertex has degree at least k
- coreness of a vertex: the maximum k-core it belongs to
- k-core decomposition: find the coreness value of each vertex
- Usage: k-core decomposition is used for
  - Graph visualization, graph clustering
  - analysis of internet map; preprocessing for finding maximum clique

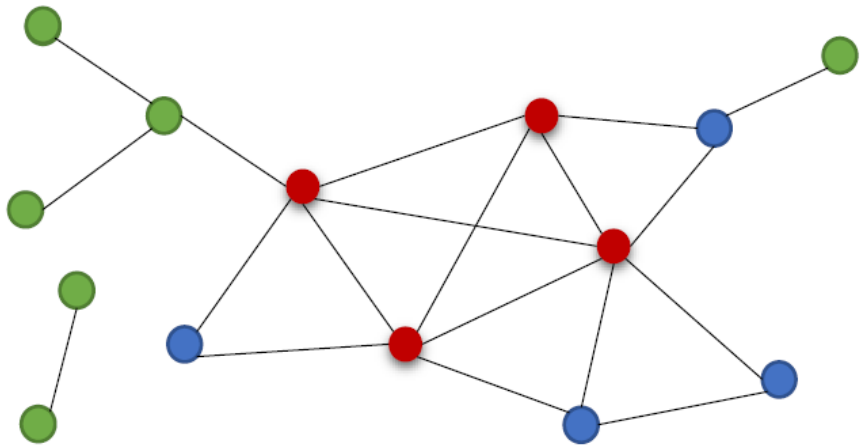

Figure: An example graph to illustrate k-core decomposition. Vertex colors indicate coreness values (1: green, 2: blue, 3: red).

PKC: Parallel k-core Decomposition Algorithm

- k-core decomposition is computed in a level by level fashion
- Initially, an array d stores the degree of each vertex
- to find vertices with coreness k
  - scan d and add all vertices with degree k to local *buffer*
  - process local *buffer* until it becomes empty
  - processing decreases degree of a vertex
  - if degree of a vertex becomes k, add the vertex to local *buffer*
- switch to a small graph when a significant fraction of vertices are done processing
- increment k and process until the whole graph is done

PKC: performance

- On average 1.90× faster than ParK and 5.33× faster than MPM
- We report speedup of PKC compared to ParK on 24 cores

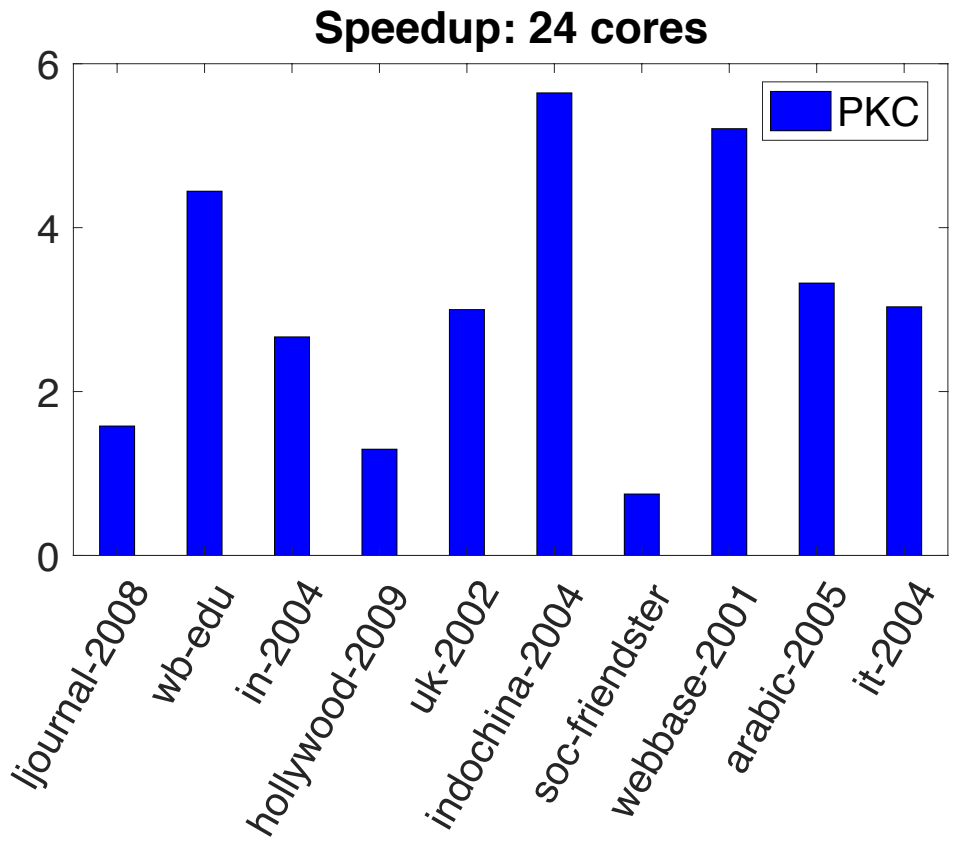

Figure: PKC speedup on 24 cores relative to ParK.

k-truss Decomposition

- k-truss: a non-trivial, one component subgraph such that each edge is contained in at least $(k-2)$ triangles
- trussness of an edge: the maximum k-truss that it belongs to
- k-truss decomposition: find the trussness value of each edge
- k-truss decomposition is used for
  - graph clustering; preprocessing for finding maximum clique

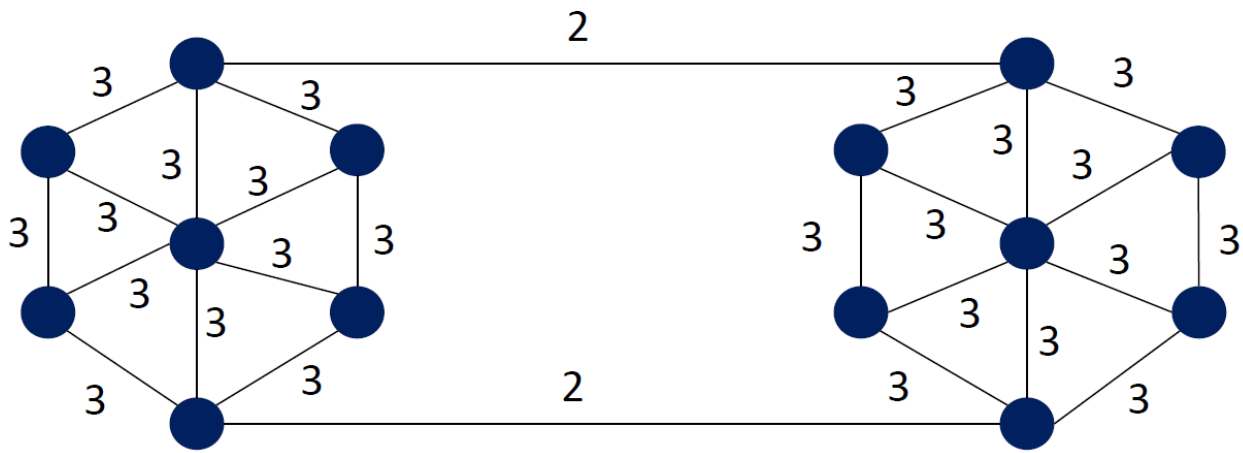

Figure: An example graph showing k-core and k-truss decomposition. Two edges have trussness 2 and rest of the edges have trussness 3.

PKT: Parallel k-truss Decomposition Algorithm

- Use the best triangle counting algorithm for support computation
- to find edges with trussnes k
  - find initial edges with trussness value k
  - process these edges in parallel
  - this may find new edges with trussness k
  - process until no more edges can be added

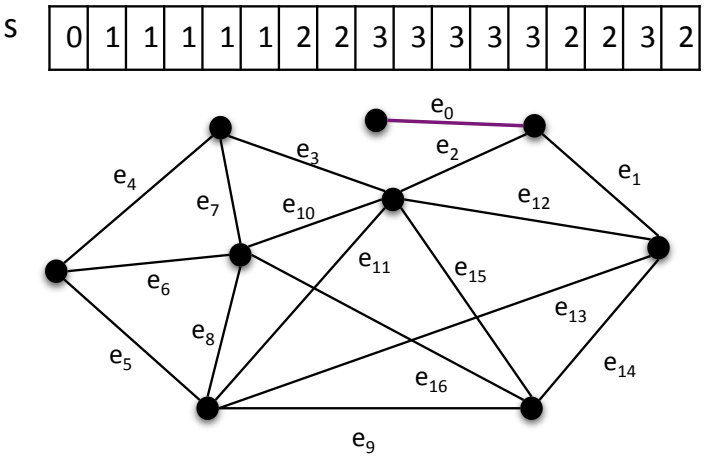

Figure: **Iteration 1**: Finds edges with trussness 2.

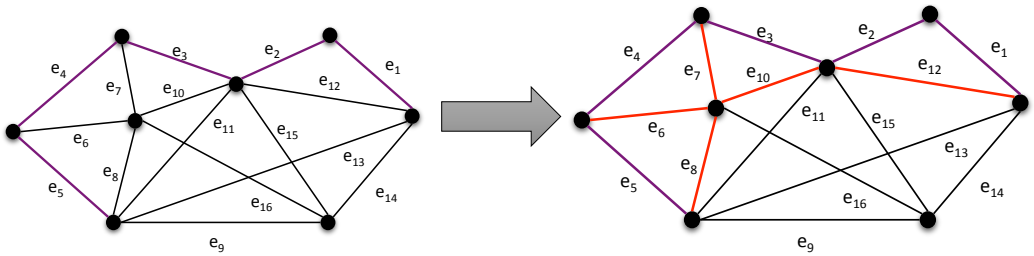

Figure: **Iteration 2**: Finds edges with trussness 3.

PKT: Performance

- On 24 cores, PKT achieves a relative speedup of 9.68×
- On 24 cores, truss decomposition takes 8.5 minutes for Friendster

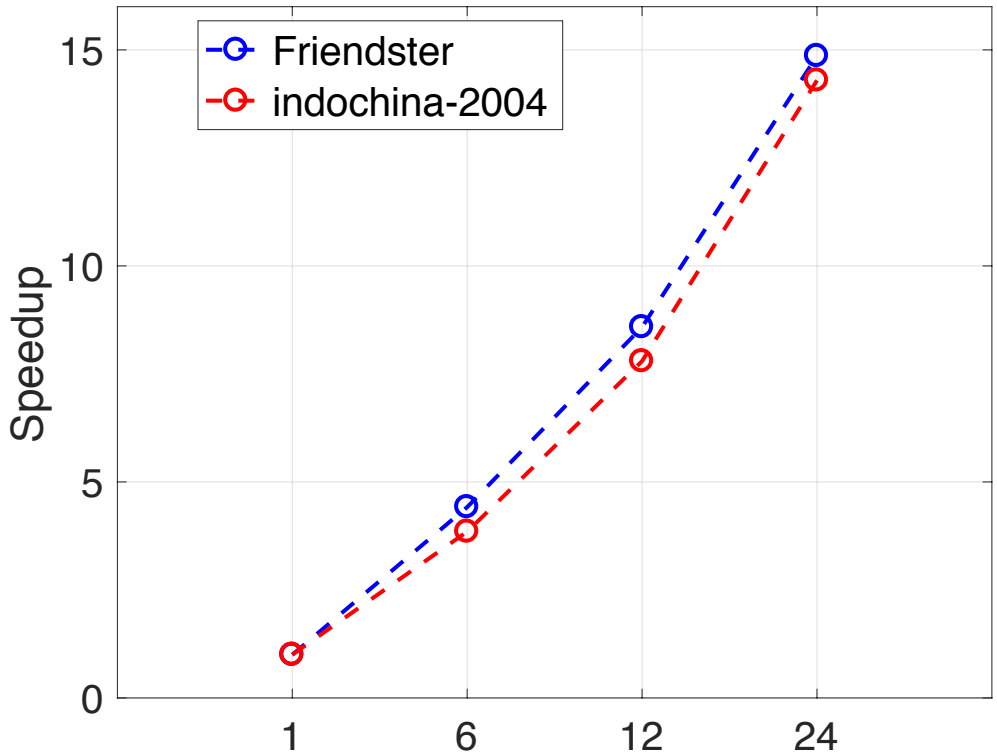

Figure: Relative speedup of PKT for Friendster and indochina-2004.

Graph Analysis

- Hierarchical decomposition algorithms (k-core and k-truss) are used for graph coarsening, sparsification, vertex reordering, partitioning and community detection.
- SlashBurn: SB(k) reordering algorithm repeatedly deletes k vertices with highest centrality
- SB decreases bits needed for compression and decreases number of non-empty block
- Our SB-core (SB-truss) reordering based on SlashBurn repeatedly removing vertices in the maximum k-core (k-truss)

Graph Analysis Results

- SB-core (SB-truss) decreases number of bits needed to compress each edge by more than 22% compared to SB
- SB-core (SB-truss) decreases number of nonempty blocks in the reordered graph by more than 28% compared to SB

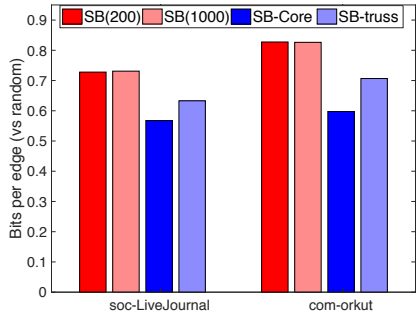

Figure: Bits needed per edge (vs random) to compress graph.

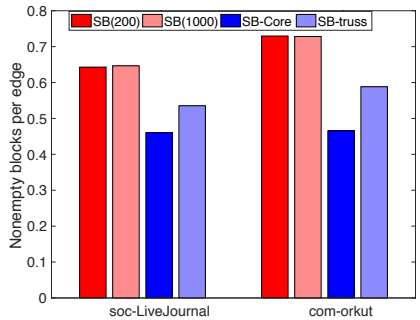

Figure: Number of non-empty blocks per edge (vs random).

SpMV : Sparse Matrix-Vector Multiplication

- $y = A * x$, multiply A with x, where
  - A is a sparse matrix
  - x is a dense vector
- SpMV is used in
  - scientific computing
  - pagerank computation

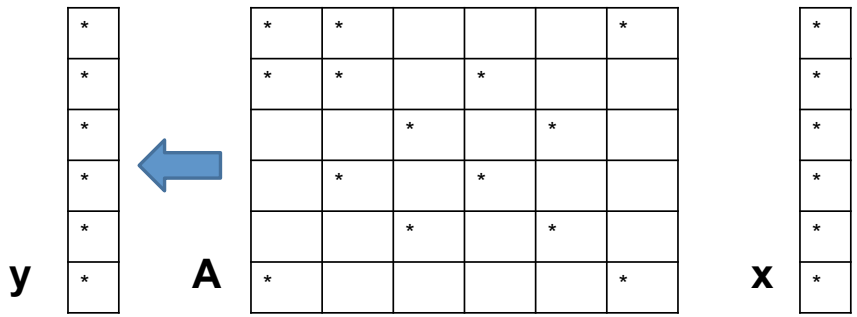

Figure: Sparse matrix-vector multiplication ($y = A * x$).

CSR-k: Multilevel CSR

- **Graph coarsening** is used to represent matrix in *CSR-k*
- **Decrease bandwidth:** RCM on the coarsened graph; induces ordering on original graph
- **SpMV** is performed by using the coarsened graph; threads compute y for all rows in a super-row.

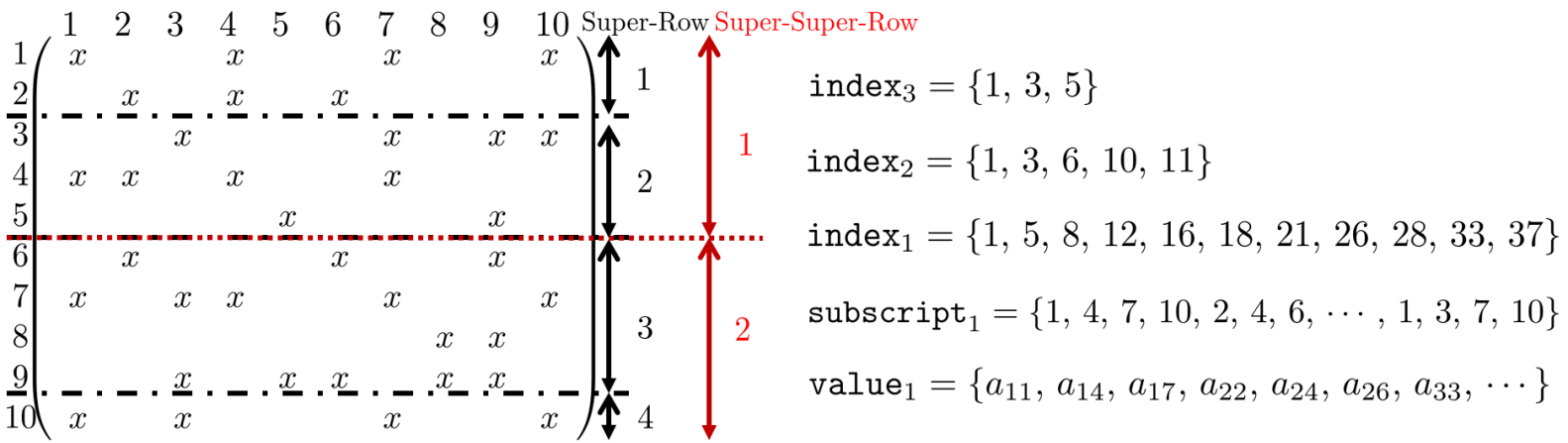

Figure: An example *CSR-k* representation for $k = 3$.

SpMV performance

- **SpMV** using *CSR-k* is 3.82× and 2.12× faster than **MKL** and **pOSKI**
- We report the speedup of *CSR-k* and pOSKI relative to *MKL* on 24 cores.

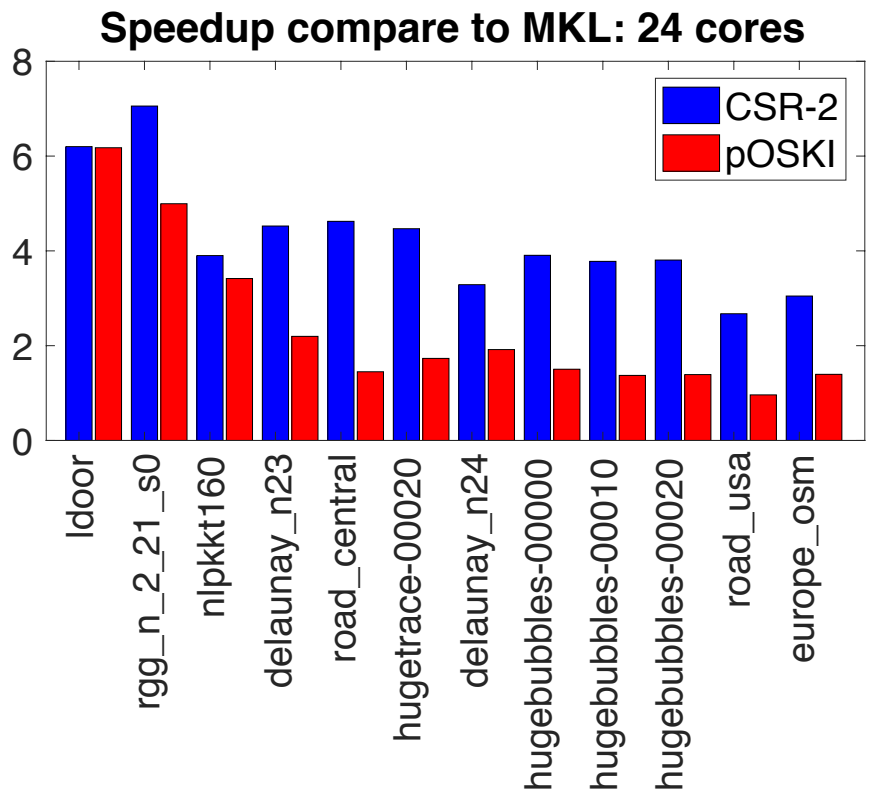

Figure: Speedup of *CSR-k* and pOSKI relative *MKL* on 24 cores.

STS: Sparse Triangular Solution

- $L * x = b$, solve for x, where
  - L is a sparse and lower triangular matrix
- STS is an important kernel in
  - Linear system of equations
  - Least squares problems

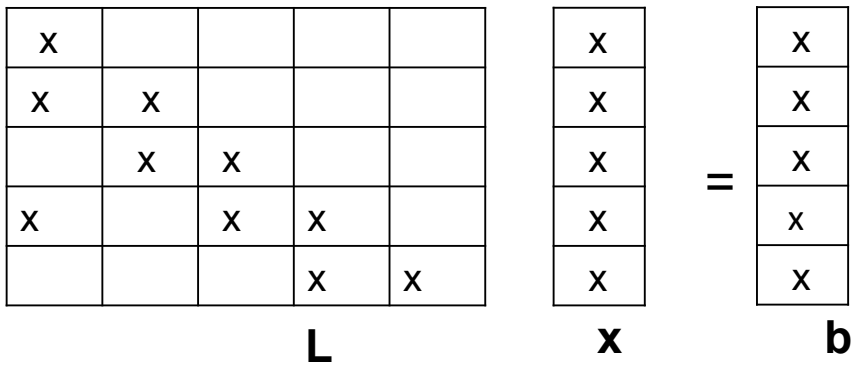

Figure: Sparse triangular solution ($L * x = b$).

STS using *CSR-k*

- Color coarsened graph; solve each color in parallel
- Graph coarsening increases spatial locality; putting connected super-rows near by increases temporal locality

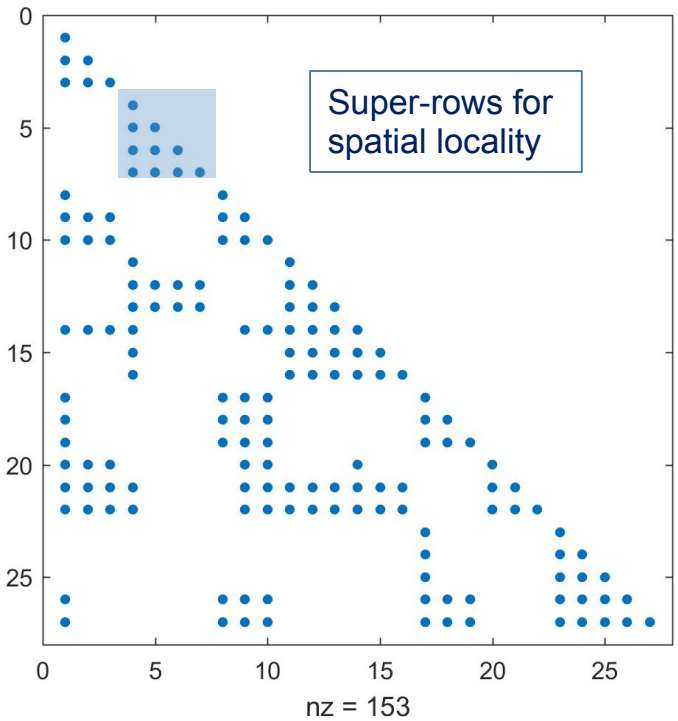

Figure: STS using *CSR-k*; super rows increase spatial locality.

STS performance

- On 24 cores, STS using *CSR-k* takes 0.04 sec for the matrix europe_osm.
- On 24 cores, *CSR-k* achieves a geometric mean speedup of 7.72× for a test suite with 12 matrices.

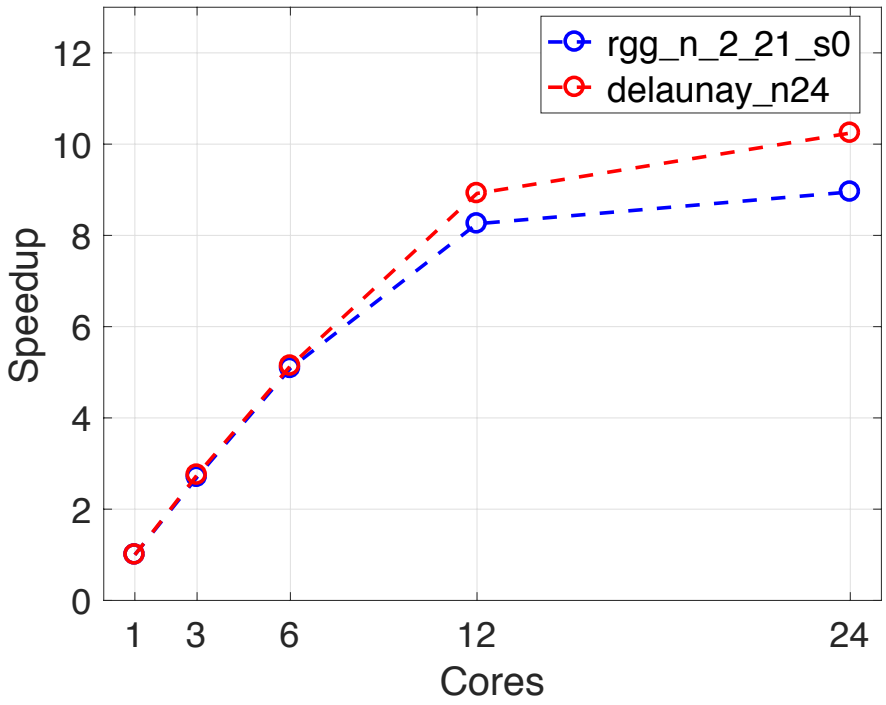

Figure: Relative speedup of STS-3.

Conclusions and Future work

- Sparse graphs are big and complex; hard to design effective parallel algorithms
- we have developed hierarchical sparse graph algorithms (k-core, k-truss) on shared memory systems; used them for graph analysis
- we have also developed parallel algorithms for sparse matrix computations (SpMV , STS)
- Future work
  - Develop sparse graph algorithms for larger graphs and larger systems
  - Develop algorithms on many core system (GPU, Xeon Phi) and distributed memory cluster

I would like to thank Padma Raghavan, Joshua D. Booth, Guillaume Aupy, Anne Benoit, Yves Robert. This work is supported by the US National Science Foundation grants ACI-1253881 and CCF-1439057.