Difficult Problems in High-Performance Computation on Large Graphs

John R. Gilbert
University of California at Santa Barbara

DOE / DOD Workshop on Emerging High Performance Architectures and Applications

November 29, 2007

Graphs

- Very large graphs appear in many HPC applications.

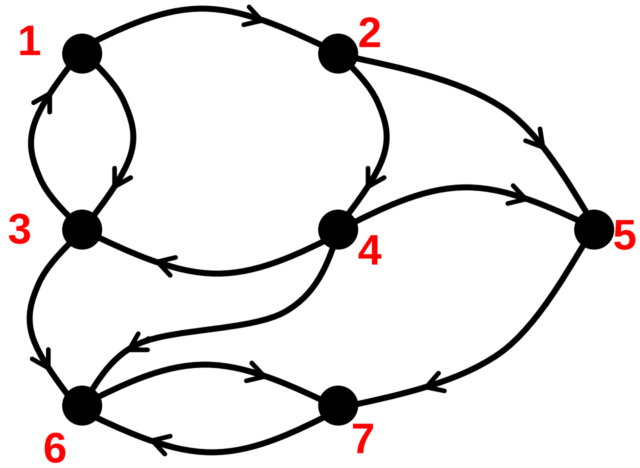
- Indeed, large graph applications are rapidly becoming more and more common.

- Computational biology, informatics and analytics, web search, network theory, dynamical systems, sparse matrix computation, geometric modeling, ….

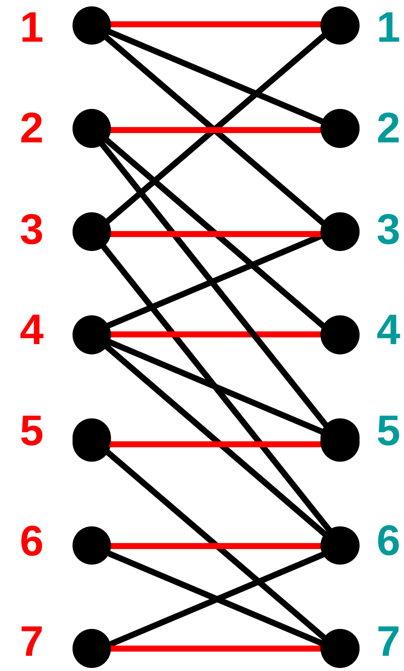
UCSB

Graph view and matrix view

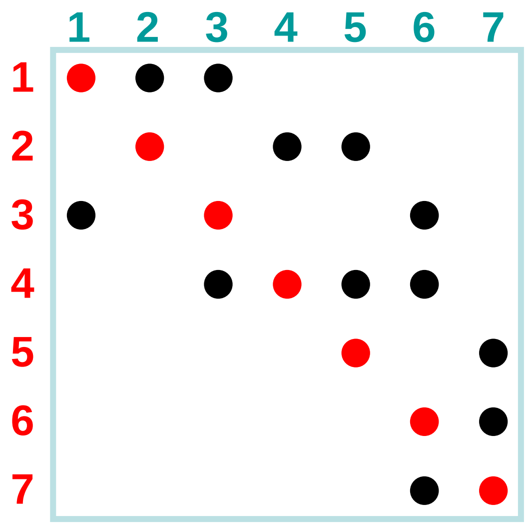
Directed graph



Bipartite graph

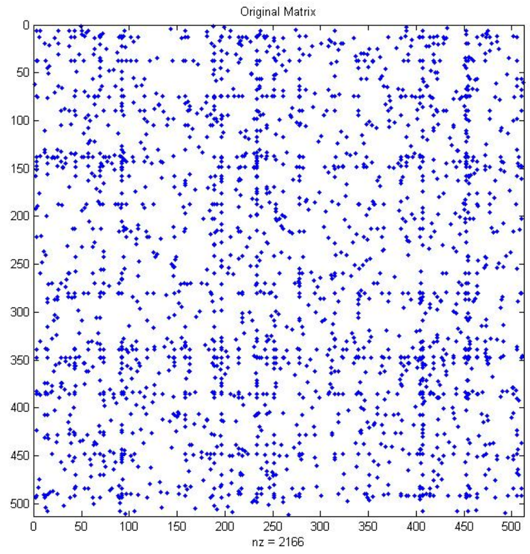


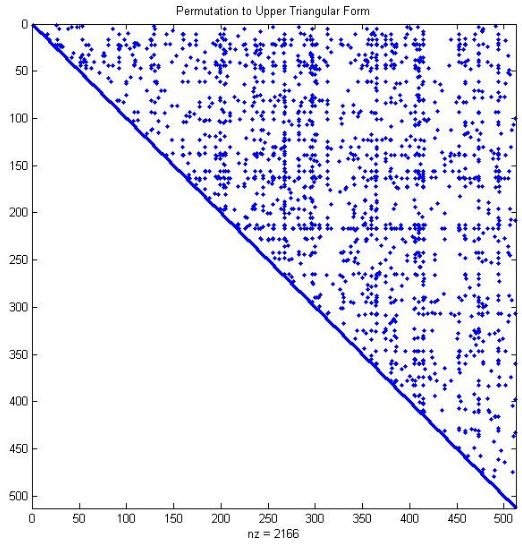
Adjacency matrix



UCSB

Kernel: Sort permuted triangular matrix





Original matrix

Permuted to upper triangular form

- Used in sparse linear solvers (e.g. Matlab's)
- Simple kernel abstracts many other graph operations (see next)
- <u>Sequential:</u> linear time; greedy topological sort; no locality
- <u>Parallel:</u> very unbalanced; one DAG level per step;
  possible long sequential dependencies

UCSB

Graph k-core

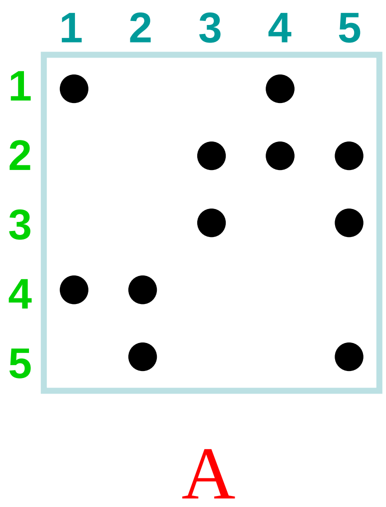
- Delete all vertices of degree less than k
- Repeat until no such vertices remain

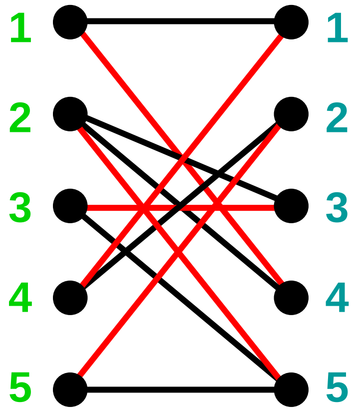
- Used (originally in biological applications) to find "essential" or "strongly related" subgraphs of a graph

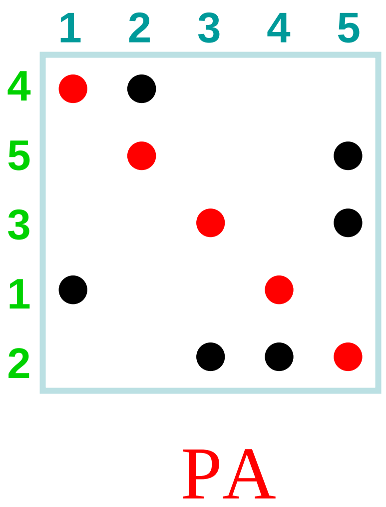
- Triangular matrix algorithm is 2-core of a bipartite graph
- k-core has similar issues in parallel

UCSB

Matching in bipartite graph

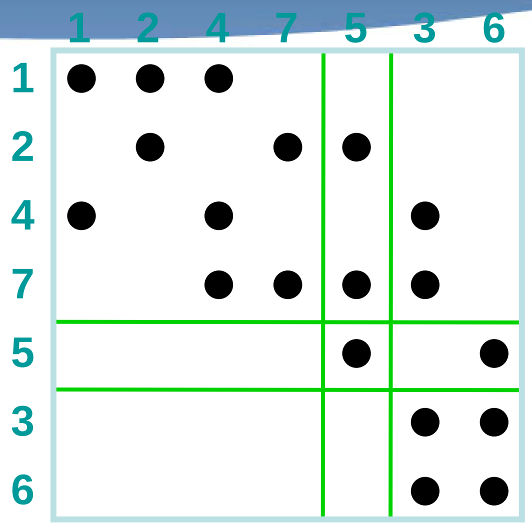




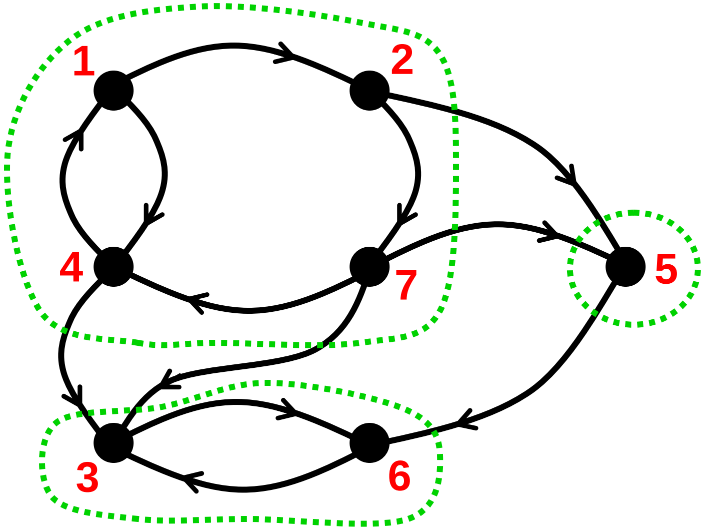


- Perfect matching: set of edges that hits each vertex exactly once
- Matrix permutation to put nonzeros on diagonal
- Variant: Maximum-weight matching

Strongly connected components



PAP^T

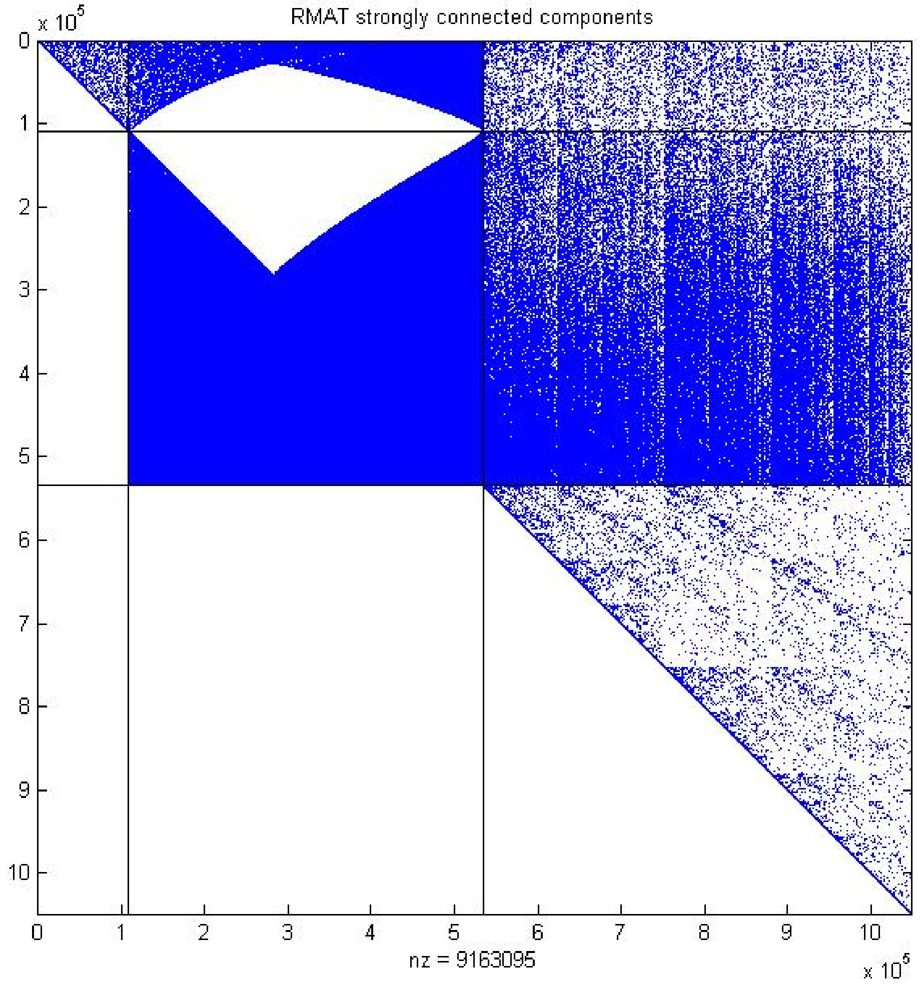


G(A)

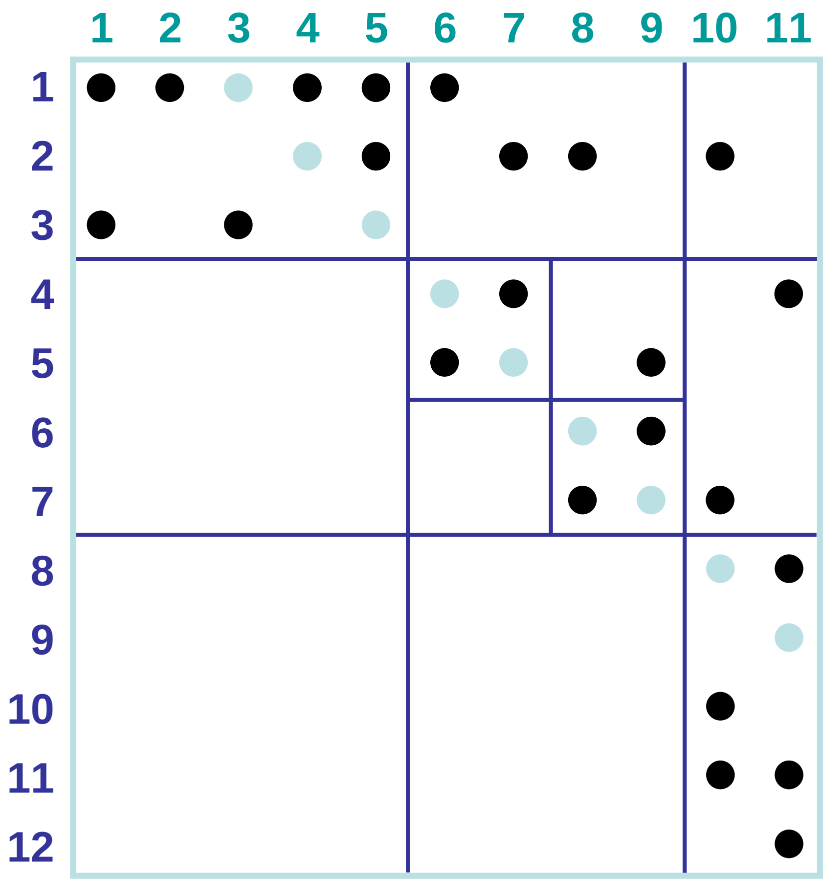
- Symmetric permutation to block triangular form
- Diagonal blocks are strong Hall (irreducible / strongly connected)
- Sequential: linear time by depth-first search [Tarjan]
- Parallel: divide & conquer algorithm, performance depends on input [Fleischer, Hendrickson, Pinar]

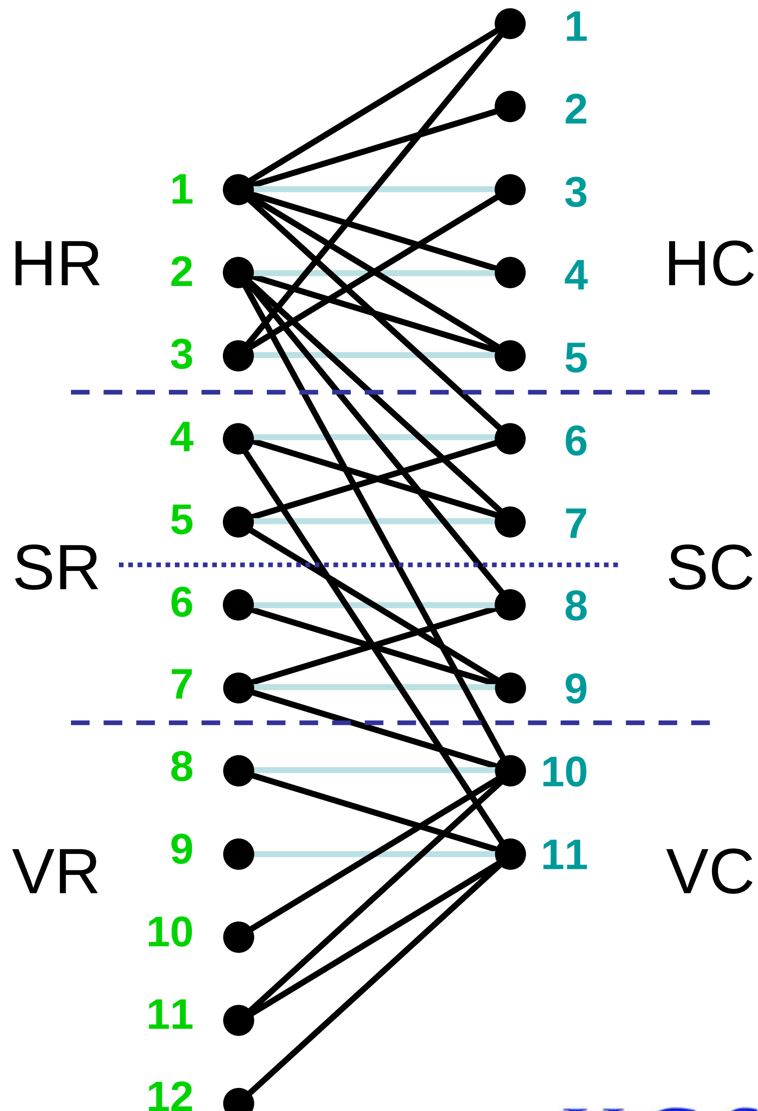
UCSB

Strongly connected components



Dulmage-Mendelsohn decomposition





Applications of D-M decomposition

- Permutation to block triangular form for Ax=b
- Connected components of undirected graphs
- Strongly connected components of directed graphs
- Minimum-size vertex cover of bipartite graphs
- Extracting vertex separators from edge cuts for arbitrary graphs
- For strong Hall matrices, several upper bounds in nonzero structure prediction are best possible

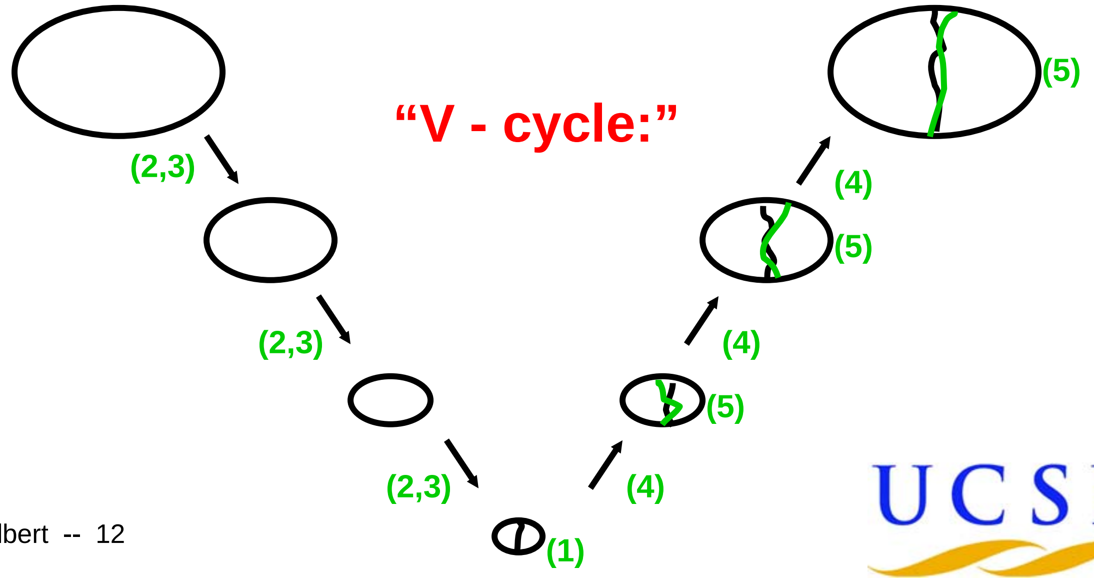
UCSB

Graph partitioning

- Graph partitioning heuristics have been studied for many years, often motivated by partitioning for parallel computation.
- Best results (and best theory) for graphs from PDE problems.
- Some approaches:
  - Iterative swapping (Kernighan-Lin, Fiduccia-Matheysses)
  - Spectral partitioning (eigenvectors of graph Laplacian)
  - Geometric partitioning (for meshes with coordinates)
  - Breadth-first search (fast but poor performance)
- Modern codes (Metis, Chaco) use multilevel iterative swapping.
- Parallel versions exist (e.g. ParMetis) but don't work as well.
- Partitioning for non-PDE problems is poorly understood in general.

Multilevel partitioning sketch

(N+,N-) = Multilevel_Partition(N, E)
… recursive partitioning routine returns N+ and N- where N = N+ U N-
if |N| is small
(1) Partition G = (N,E) directly to get N = N+ U N-
Return (N+, N-)
else
(2) Coarsen G to get an approximation $G_C = (N_C, E_C)$
(3) (N_C+ , N_C-) = Multilevel_Partition(N_C, E_C)
(4) Expand (N_C+ , N_C-) to a partition (N+ , N-) of N
(5) Improve the partition (N+ , N-)
Return (N+ , N-)
endif



Slide courtesy of Kathy Yelick

Some other key graph kernels

- Graph contraction
- Connected components
- s-t connectivity
- Shortest paths
- Subgraph isomorphism

Many studied by Berry, Hendrickson, and others on MTA architecture

UCSB

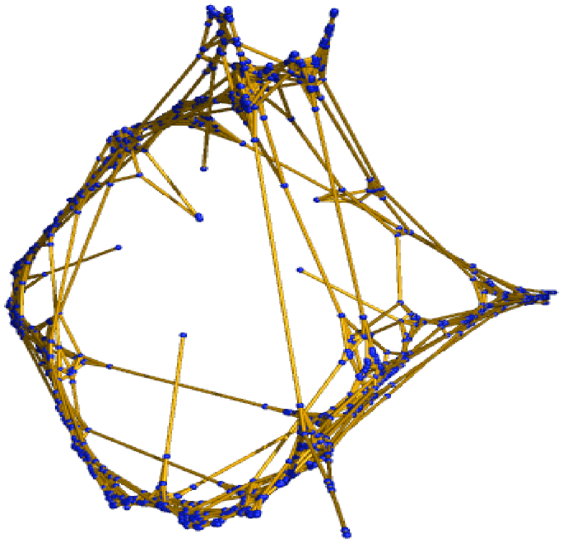
Sparse Matrix times Sparse Matrix

- A primitive in many array-based graph algorithms:
  - Parallel breadth-first search
  - Shortest paths
  - Graph contraction
  - Subgraph / submatrix indexing
  - Etc.

- Graphs are often *not* mesh-like, i.e. geometric locality and good separators.

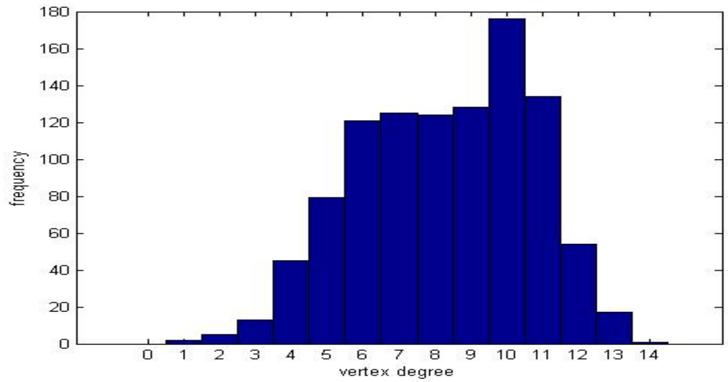
- Do *not* want to optimize for one repeated operation, as in matvec for iterative methods

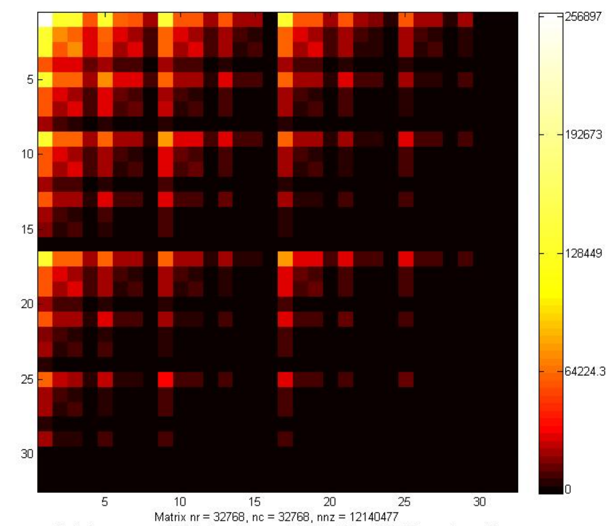
Toolbox for Graph Analysis and Pattern Discovery



Layer 1: Graph Theoretic Tools

- Graph operations
- Global structure of graphs
- Graph partitioning and clustering
- Graph generators
- Visualization and graphics
- Scan and combining operations
- Utilities





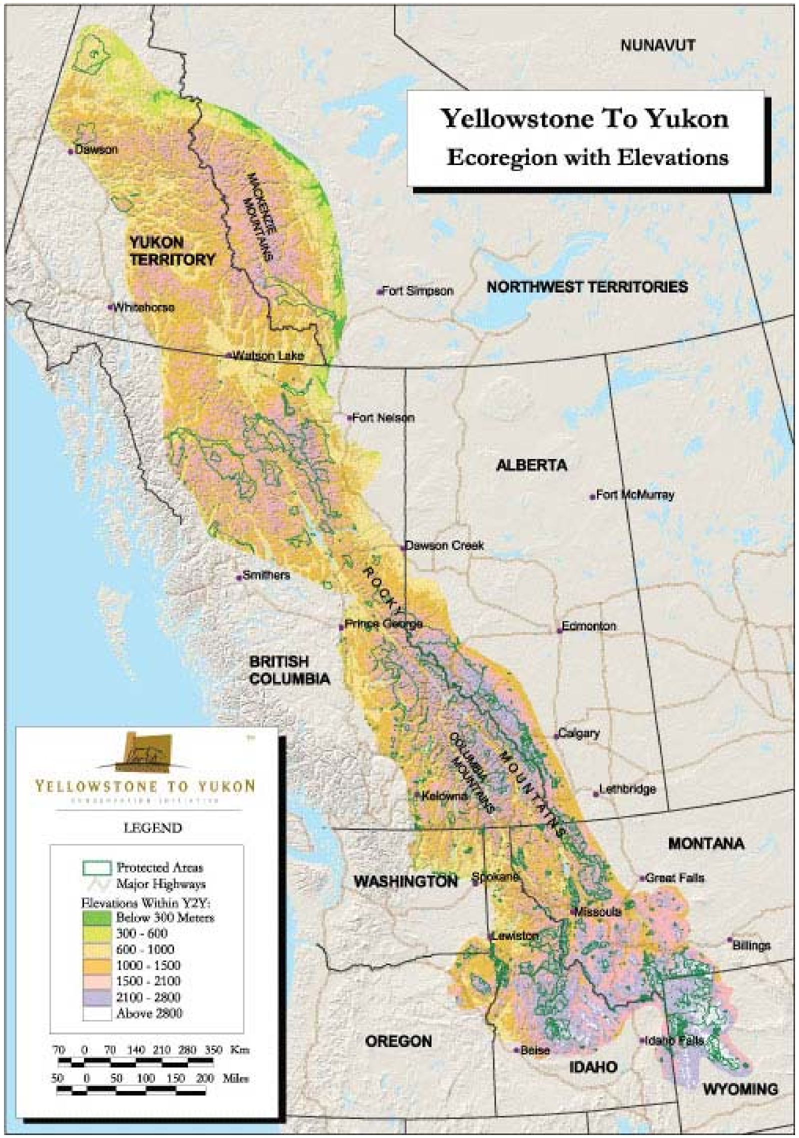
Application stack for combinatorics + numerics

Computational ecology, CFD, data exploration **Applications**
CG, BiCGStab, etc. + combinatorial preconditioners (AMG, Vaidya) **Preconditioned Iterative Methods**
Graph querying & manipulation, connectivity, spanning trees, geometric partitioning, nested dissection, NNMF, . . . **Graph Analysis & PD Toolbox**
Arithmetic, matrix multiplication, indexing, solvers (\, eigs) **Distributed Sparse Arrays**

UCSB

Landscape connectivity modeling

- Habitat quality, gene flow, corridor identification, conservation planning
- Pumas in southern California: 12 million nodes, < 1 hour
- Targeting larger problems: Yellowstone-to-Yukon corridor



Figures courtesy of Brad McRae, NCEAS

UCSB

Issues in (many) large graph applications

- Multiple simultaneous queries to same graph
  - Graph may be fixed, or slowly changing
  - Throughput and response time both important
- Dynamic subsetting
  - User needs to solve problem on "their own" version of the main graph
  - E.g. landscape data masked by geographic location, filtered by obstruction type, resolved by species of interest

UCSB

Productivity

Raw performance isn’t always the only criterion.

Other factors include:

- Seamless scaling from desktop to HPC
- Interactive response for data exploration and viz
- Rapid prototyping
- Just plain programmability

UCSB

Some approaches to HPC graph programming

- Visitor-based multithreaded – MTGL + XMT
  - \+ search templates natural for many algorithms
  - \+ relatively simple load balancing
  - – complex thread interactions, race conditions
- Array-based data parallel – GAPDT + parallel Matlab
  - \+ relatively simple control structure
  - \+ user-friendly interface
  - – some algorithms hard to express naturally
  - – load balancing not as simple
- Scan-based vectorized – NESL: something of a wild card
- We don't really know the right set of primitives yet!

UCSB