STAR-P: High Productivity Parallel Computing

David Cheng, Ron Choy, Alan Edelman
Massachusetts Institute of Technology

John R. Gilbert and Viral Shah
University of California at Santa Barbara

Graph Algorithms and Sparse Matrix Land

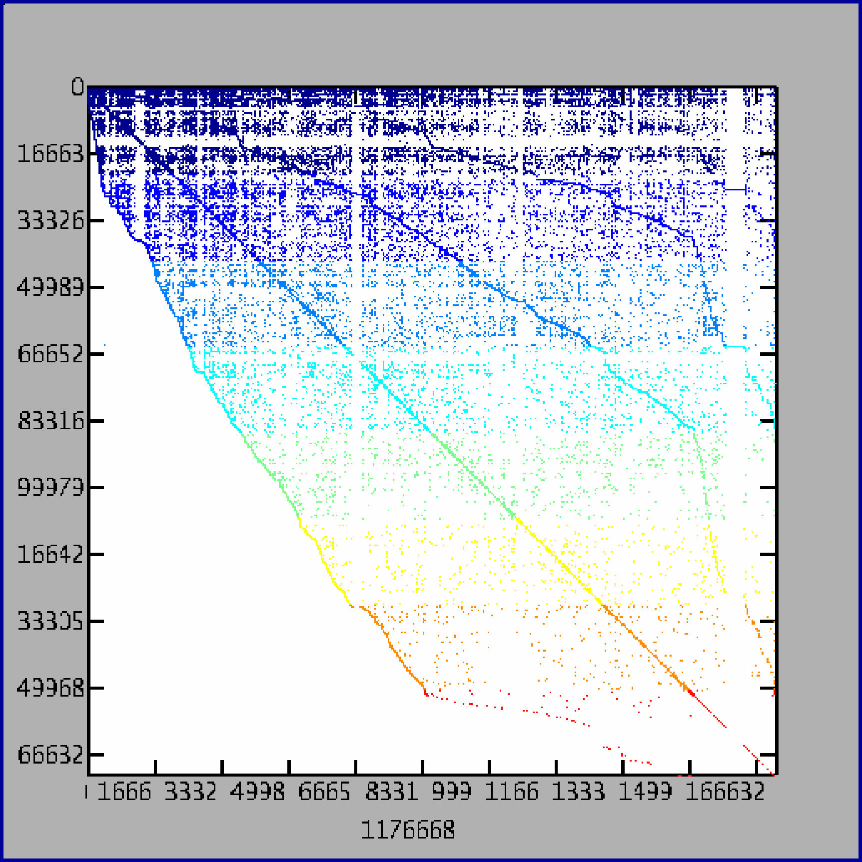
Birth of Interactive Supercomputing

- Dream of taking academic software commercial

Star-P

- Interactive Parallel Computing Environment
- Parallel Client/Server Architecture
- Main goal: parallel computing easier on the human user
- Academic Front End: MATLAB
- Four parallel approaches interacting:
  - Embarrassingly Parallel
  - Message Passing
  - Backend Support (insert *p)
  - Compiling
- Integrates several packages into one easy to use software

Page Rank Matrix



- Web crawl of 170,000 pages from mit.edu
- Matlab*P spy plot of the matrix of the graph

Clock

- c=mm(‘clock’);
- std(c);

- Simple example shows two modes interacting

Pieces of Pi

```
>> quad('4./(1+x.^2)', 0, 1);
ans = 3.14159270703219

>> a = (0:3*p) / 4
a = ddense object: 1-by-4

>> a(:)
ans =
                  0
   0.25000000000000
   0.50000000000000
   0.75000000000000

>> b = a+.25;

>> c = mm('quad','4./(1+x.^2)', a, b);   % Should be “feval”!
c = ddense object: 1-by-4

>> sum(c(:))
ans = 3.14159265358979
```

FFT2 in four lines

```
>>  A = randn(4096, 4096*p)
A = ddense object: 4096-by-4096
>> tic;

>> B = mm('fft', A);
>> C = B.';
>> D = mm('fft', C);
>> F = D.';

>> toc
elapsed_time = 73.50

>>a = A(:,:);
>>  tic; g = fft2(a); toc
elapsed_time = 202.95
```

… we have FFTW installed as well!

Matlab sparse matrix design principles

- All operations should give the same results for sparse and full matrices (almost all)
- Sparse matrices are never created automatically, but once created they propagate
- Performance is important -- but usability, simplicity, completeness, and robustness are more important
- Storage for a sparse matrix should be O(nonzeros)
- Time for a sparse operation should be O(flops) (as nearly as possible)

Matlab sparse matrix design principles

- All operations should give the same results for sparse and full matrices (almost all)
- Sparse matrices are never created automatically, but once created they propagate
- Performance is important -- but usability, simplicity, completeness, and robustness are more important
- Storage for a sparse matrix should be O(nonzeros)
- Time for a sparse operation should be O(flops) (as nearly as possible)

Matlab*P dsparse matrices: same principles, but some different tradeoffs

Sparse matrix operations

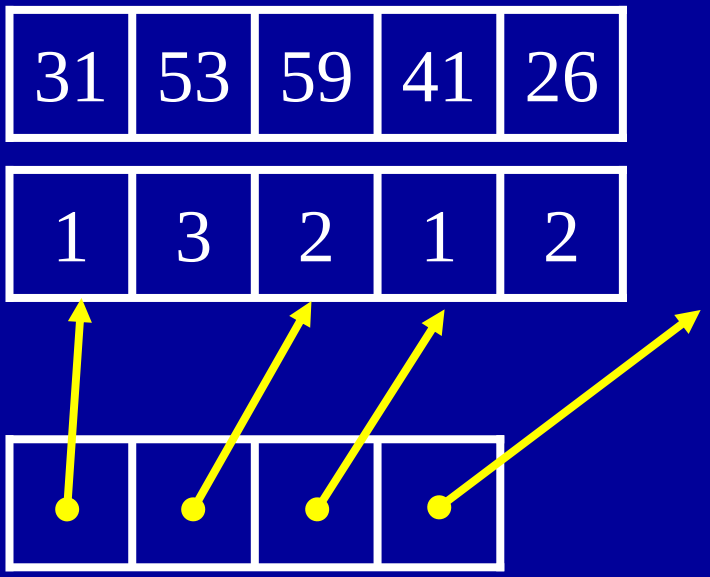
- dsparse layout, same semantics as ddense
- For now, only row distribution
- Matrix operators: +, -, max, etc.
- Matrix indexing and concatenation

  A (1:3, [4 5 2]) = [B(:, 7) C] ;

- A \ b by direct methods
- Conjugate gradients

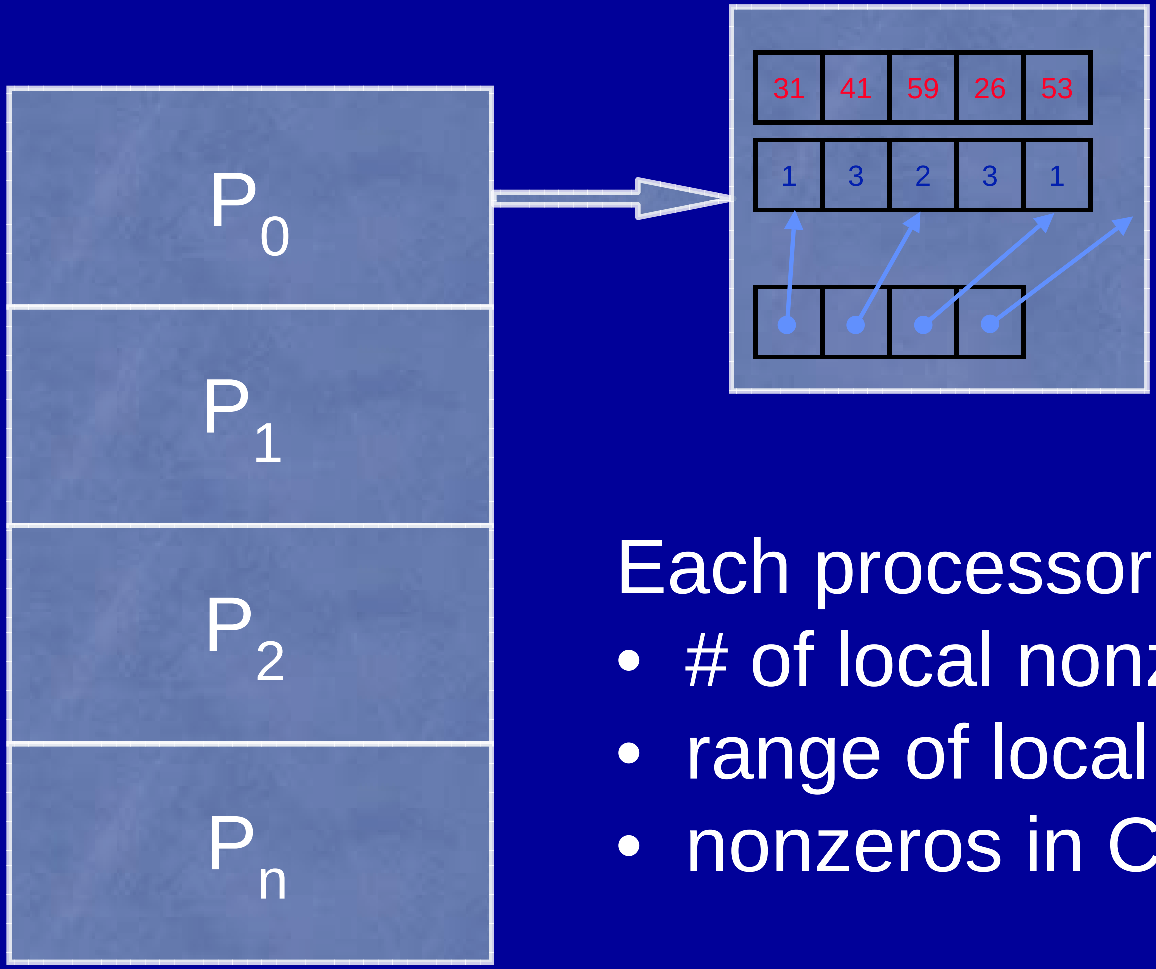
Sparse data structure

31	0	53
0	59	0
41	26	0



- Full:
  - 2-dimensional array of real or complex numbers
  - (nrows*ncols) memory
- Sparse:
  - compressed row storage
  - about (1.5*nzs + .5*nrows) memory

Distributed sparse data structure



Each processor stores:

- # of local nonzeros
- range of local rows
- nonzeros in CSR form

Sparse matrix times dense vector

- y = A * x
- The first call to matvec caches a communication schedule for matrix A. Later calls to multiply any vector by A use the cached schedule.
- Communication and computation overlap.
- Can use a tuned sequential matvec kernel on each processor.

Sparse linear systems

- x = A \ b
- Matrix division uses MPI-based direct solvers:
  - SuperLU_dist: nonsymmetric static pivoting
  - MUMPS: nonsymmetric multifrontal
  - PSPASES: Cholesky

```
ppsetoption('SparseDirectSolver','SUPERLU')
```

- Iterative solvers implemented in Matlab*P
- Some preconditioners; ongoing work

Application: Fluid dynamics

- Modeling density-driven instabilities in miscible fluids (Goyal, Meiburg)
- Groundwater modeling, oil recovery, etc.
- Mixed finite difference & spectral method
- Large sparse generalized eigenvalue problem

```
function lambda = peigs (A, B,
  sigma, iter, tol)

  [m n] = size (A);
  C = A - sigma * B;
  y = rand (m, 1);

  for k = 1:iter
    q = y ./ norm (y);
    v = B * q;
    y = C \ v;
    theta = dot (q, y);
    res = norm (y - theta*q);
    if res <= tol
      break;
    end;
  end;

  lambda = 1 / theta;
```

Combinatorial algorithms in Matlab*P

- Sparse matrices are a good start on primitives for combinatorial scientific computing.
  - Random-access indexing: `A(i,j)`
  - Neighbor sequencing: `find (A(i,:))`
  - Sparse table construction: `sparse (I, J, V)`
- What else do we need?

Sorting in Matlab*P

- **`[V, perm] = sort (V)`**

- Common primitive for many sparse matrix and array algorithms: sparse(), indexing, transpose

- Matlab*P uses a parallel sample sort

Sample sort

- (Perform a random permutation)
- Select p-1 “splitters” to form p buckets
- Route each element to the correct bucket
- Sort each bucket locally
- “Starch” the result to match the distribution of the input vector

Sample sort example

Initial data (after randomizing)

3	6	8	1	5	4	7	2	9

Choose splitters (2 and 6)

1	2	3	6	5	4	8	7	9

Sort local data

1	2	3	4	5	6	7	8	9

Starch

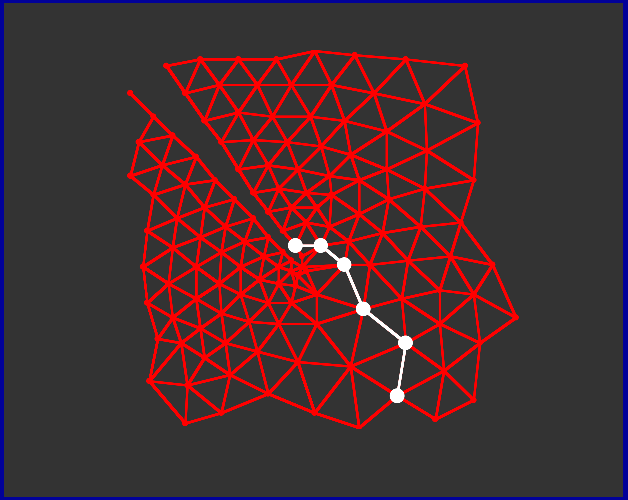
1	2	3	4	5	6	7	8	9

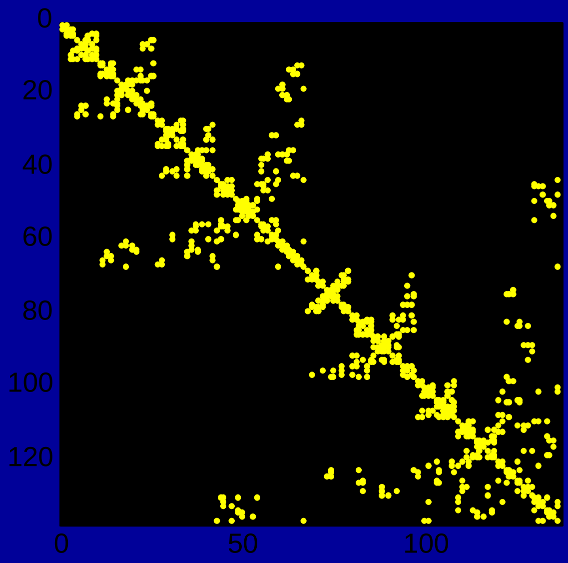
How sparse( ) works

- **`A = sparse (I, J, V)`**
- Input: ddense vectors I, J, V (optionally, also dimensions and distribution info)
- Sort triples (i, j, v) by (i, j)
- Starch the vectors for desired row distribution
- Locally convert to compressed row indices
- Sum values with duplicate indices

Graph / mesh partitioning

- Reduce communication in matvec and other parallel computations
- Reordering for sparse GE
- PARMETIS
- Parts of G/Teng Matlab meshpart toolbox





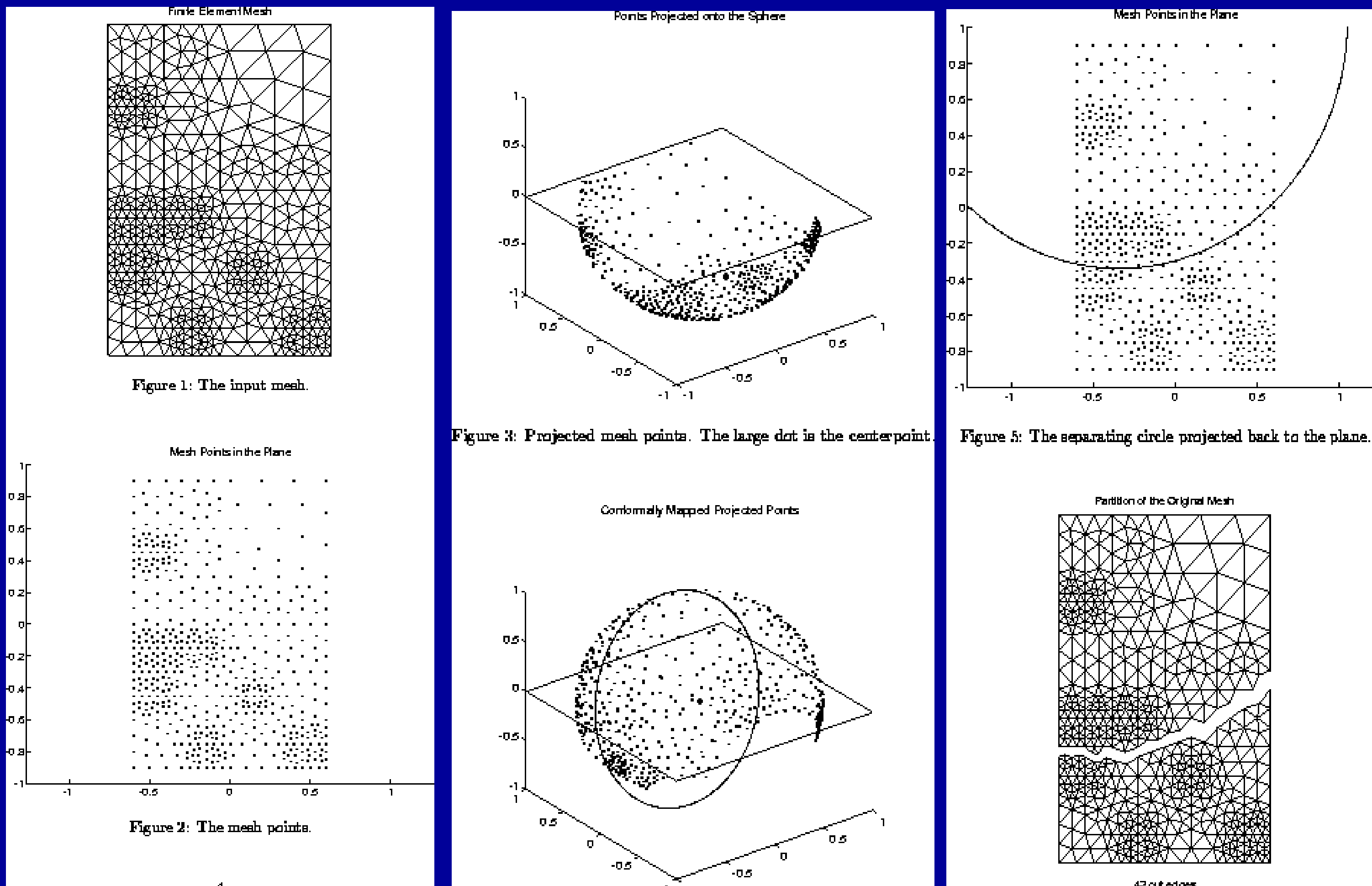
Geometric mesh partitioning

- Algorithm of Miller, Teng, Thurston, Vavasis
- Partitions irregular finite element meshes into equal-size pieces with few connecting edges
- Guaranteed quality partitions for well-shaped meshes, often very good results in practice
- Existing implementation in sequential Matlab
- Code runs in Matlab*P with very minor changes

Outline of algorithm

1. Project points stereographically from R^d to R^{d+1}
2. Find "centerpoint" (generalized median)
3. Conformal map: Rotate and dilate
4. Find great circle
5. Unmap and project down
6. Convert circle to separator

Geometric mesh partitioning



Matching and depth-first search in Matlab

- dmperm: Dulmage-Mendelsohn decomposition

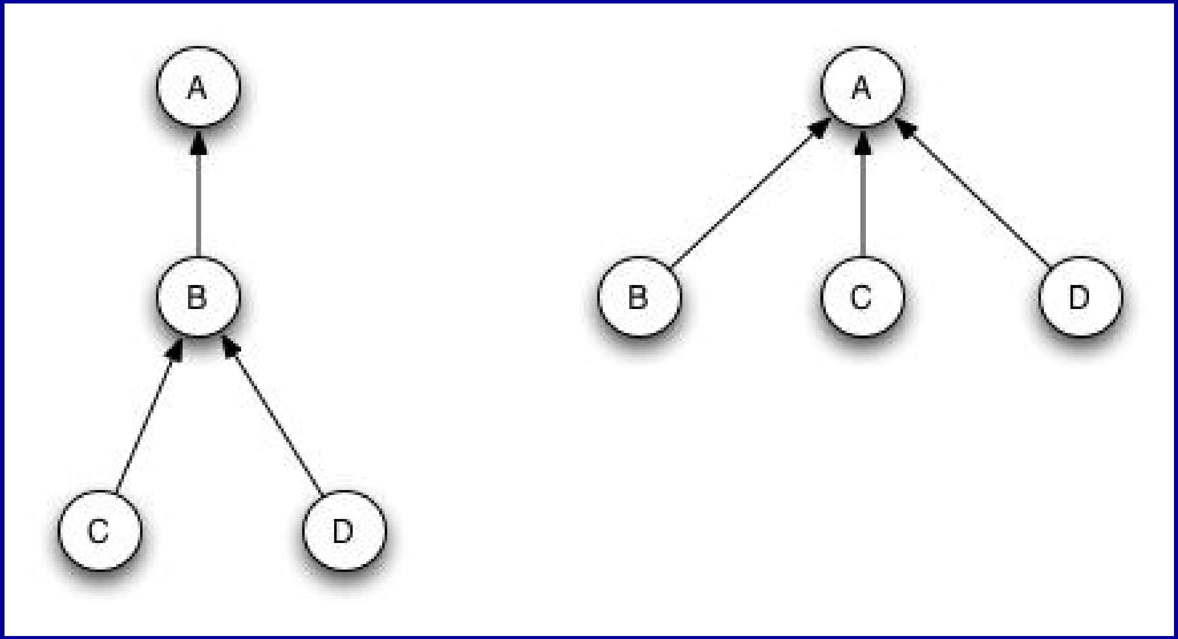
- Square, full rank A:
  - [p, q, r] = dmperm(A);
  - A(p,q) is block upper triangular with nonzero diagonal
  - also, strongly connected components of a directed graph
  - also, connected components of an undirected graph

- Arbitrary A:
  - [p, q, r, s] = dmperm(A);
  - maximum-size matching in a bipartite graph
  - minimum-size vertex cover in a bipartite graph
  - decomposition into strong Hall blocks

Connected components

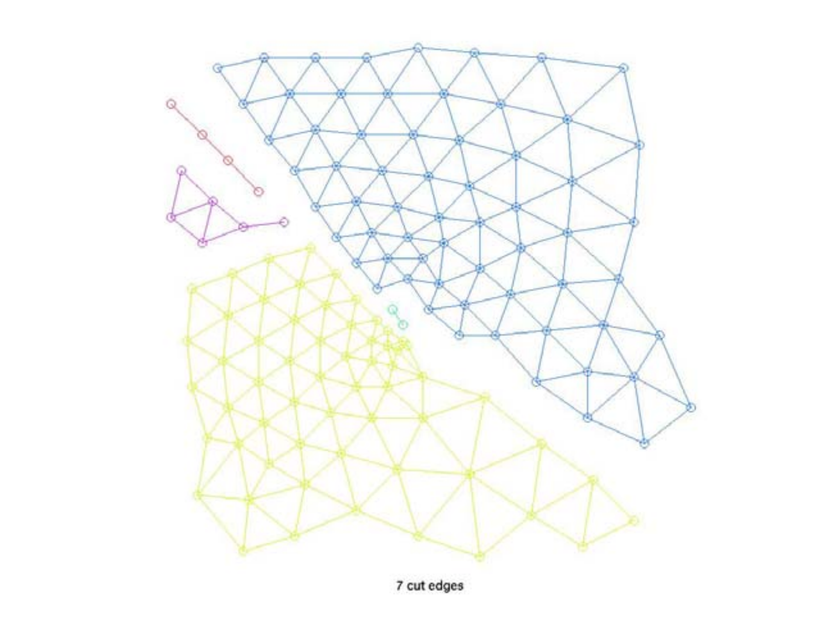
- Sequential Matlab uses depth-first search (`dmperm`), which doesn't parallelize well
- Shiloach-Vishkin algorithm:
  - repeat
    - Link every (super)vertex to a random neighbor
    - Shrink each tree to a supervertex by pointer jumping
  - until no further change
- Originally a processor-efficient PRAM algorithm
- Matlab*P code looks much like the PRAM code

Pointer jumping



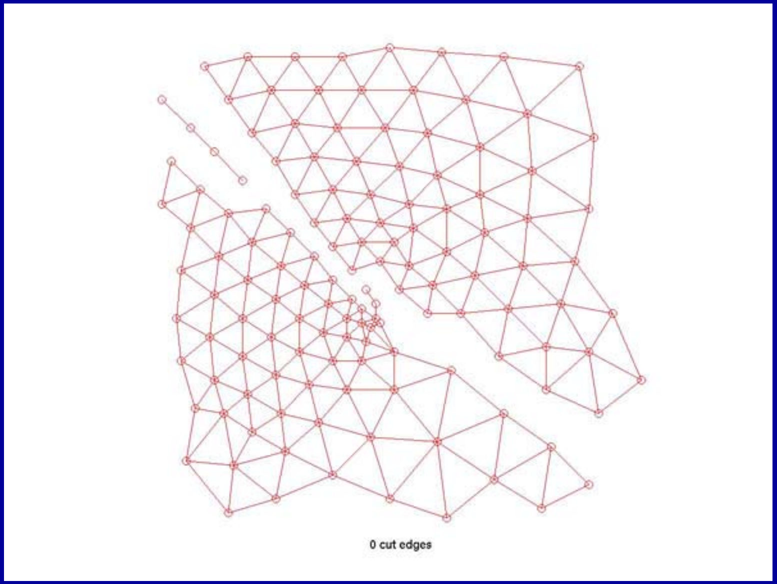

```
while ~all( C(myrows) == C(C(myrows)) )
  C(myrows) = C(C(myrows));
end
C(myrows) = min (C(myrows), C(C(myrows)));
```

Example of execution



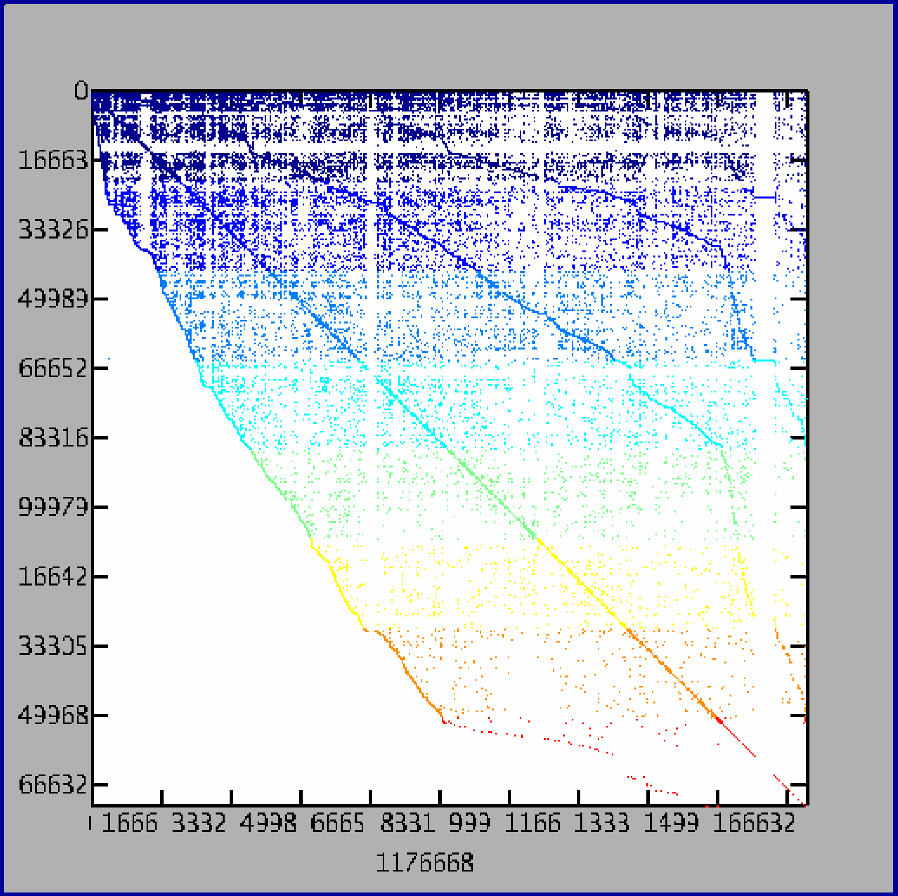
After first iteration

Final components



Page Rank

- Importance ranking of web pages
- Stationary distribution of a Markov chain
- Power method: matvec and vector arithmetic
- Matlab*P page ranking demo (from SC’03) on a web crawl of mit.edu (170,000 pages)



Remarks

- Easy-to-use interactive programming environment
- Interface to existing parallel packages
- Combinatorial methods toolbox being built on parallel sparse matrix infrastructure
  - Much to be done: spanning trees, searches, etc.
- A few issues for ongoing work
  - Dynamic resource management
  - Fault management
  - Programming in the large