Sparse Matrices and Optimized Parallel Implementations

Stan Tomov

Innovative Computing Laboratory
Computer Science Department
The University of Tennessee

April 11, 2012

Topics

Projection in
Scientific Computing

Sparse matrices,
parallel implementations

PDEs, Numerical
solution, Tools, etc.

Iterative Methods

Outline

- Part I
  - **Discussion**
- Part II
  - **Sparse matrix computations**
- Part III
  - **Reordering algorithms and parallelization**

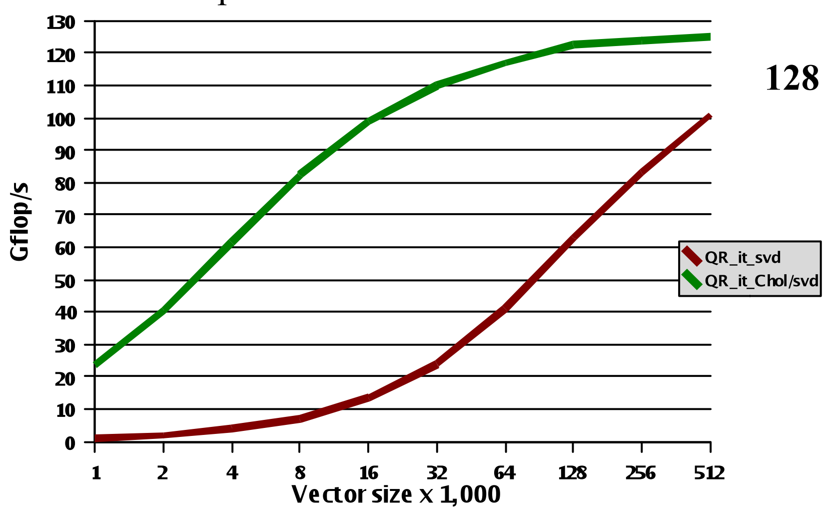
Part I

Discussion

Orthogonalization

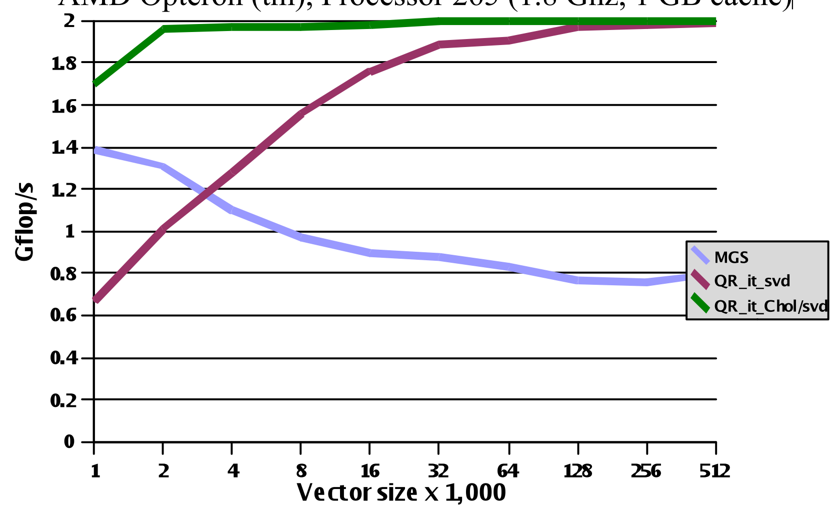
- We can orthonormalize non-orthogonal basis. **How?**
  Other approaches: QR using Householder transformation (as in LAPACK), Cholesky, or/and SVD on normal equations (**as in homeworks 9 and 10**)

Hybrid CPU-GPU (NVIDIA Quadro FX 5600)
computation as in Homework #9



128 vectors

CPU computation
AMD Opteron (tm), Processor 265 (1.8 Ghz, 1 GB cache)



What if the basis is not orthonormal?

- If we do not want to orthonormalize:

$$\mathbf{u} \approx \mathbf{P\,u} = \mathbf{c_1\,x_1 + c_2\,x_2 + \,...\, + c_m\,x_m} \qquad \text{/ 'Multiply' by } x_1, ..., x_m \text{ to get}$$

$$\begin{aligned}
&\mathbf{(u\,,x_1) = c_1\,(x_1\,,x_1) + c_2\,(x_2\,,x_1) + \,...\, + c_m\,(x_m\,,x_1)}\\
&\mathbf{...}\\
&\mathbf{(u,\,x_m) = c_1\,(x_1\,,x_m) + c_2\,(x_2\,,x_m) + \,...\, + c_m\,(x_m\,,x_m)}
\end{aligned}$$

- These are the so called **Petrov-Galerkin conditions**
- We saw examples of their use in

  * optimization, and

  * PDE discretization, e.g. FEM

What if the basis is not orthonormal?

- If we do not want to orthonormalize, e.g. **in FEM**

$$\mathbf{u} \approx \mathbf{P\,u} = \mathbf{c_1\,\phi_1 + c_2\,\phi_2 + \;.\,.\,.\; + c_7\,\phi_7}$$

/ 'Multiply' by ϕ_1, ..., ϕ_7 to get a 7x7 system

$$\mathbf{a(\,c_1\,\phi_1 + c_2\,\phi_2 + \;.\,.\,.\; + c_7\,\phi_7\,,\,\phi_i) = F(\phi_i) \quad for\; i = 1, \ldots, 7}$$

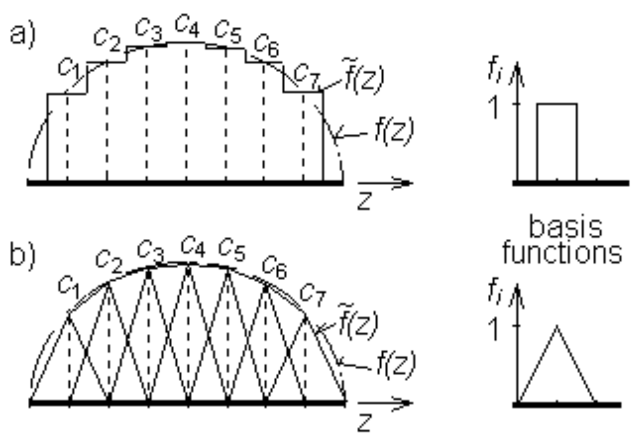


Fig. 4.1B.3 *Multi-basis approximations*
a) piece-wise constant
b) piece-wise linear

- Two examples of basis functions ϕ_i
- **The more ϕ_i overlap, the denser the resulting matrix**
- Spectral element methods (high-order FEM)

(Image taken from http://www.urel.feec.vutbr.cz/~raida)

Stencil Computations

- K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shaft, K. Yelick, "*Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors*", SIAM Review, 2008.

  http://bebop.cs.berkeley.edu/pubs/datta2008-stencil-sirev.pdf

Part II

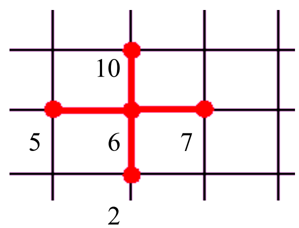
Sparse matrix computations

Sparse matrices

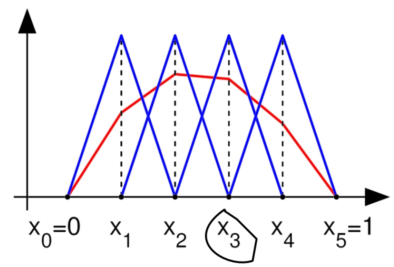
- Sparse matrix: substantial part of the coefficients is zero
- Naturally arise from PDE discretizations
  - finite differences, FEM, etc.; we saw examples in the

5-point finite difference operator

Vertices are indexed, e.g.



1-D piece-wise linear FEM



Row 6 will have 5 non-zero elements:

$A_{6,2}$, $A_{6,5}$, $A_{6,6}$, $A_{6,7}$, and $A_{6,10}$

Row 3, for example, will have 3 non-zeros

$A_{3,2}$, $A_{3,3}$, $A_{3,4}$

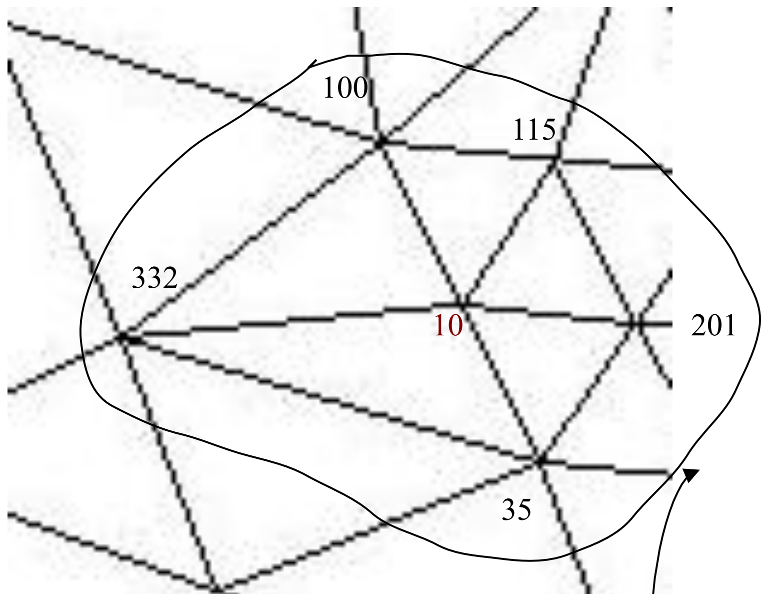
Sparse matrices

- In general :

* Degrees of freedom (DOF), associated for ex. with vertices (or edges, faces, etc.), are indexed

* A basis function is associated with every DOF (unknown)

* A **Petrov-Galerkin condition** (equation) is derived for every basis function, representing a row in the resulting system



* Only 'a few' elements per row will be nonzero as the basis functions have local support
  - eg. row 10, using continuous piecewise linear FEM, will have 6 nonzeroes: $A_{10,10}$, $A_{10,35}$, $A_{10,100}$, $A_{10,332}$, $A_{10,115}$, $A_{10,201}$
  - physical intuition behind: PDEs describe changes in physical processes; describing/discretizing these changes numerically, based only on local/neighbouring information, results in sparse matrices

eg. what happens at '10' is described by the physical state at '10' and the neighbouring 35, 201, 115, 100, and 332.

Sparse matrices

- Can we take advantage of this sparse structure?
  - To solve for example very large problems
  - To solve them efficiently

- Yes! There are algorithms
  - Linear solvers and preconditioners (to cover some in the last 2 lectures)
  - Efficient data storage and implementation (next ...)

Sparse matrix formats

- It pays to avoid storing the zeros!
- Common sparse storage formats:
  - AIJ
  - Compressed row/column storage (**CRS**/CCS)
  - Compressed diagonal storage (CDS)
    * for more see the 'Templates' book
    http://www.netlib.org/linalg/html_templates/node90.html#SECTION00931000000000000000
  - Blocked versions (why?)

AIJ

- Stored in 3 arrays
  - The same length
  - No order implied

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 0 & 4 & 0 & 0 \\ 0 & 5 & 0 & 6 & 0 \\ 0 & 0 & 7 & 0 & 8 \end{bmatrix}$$

I	**J**	**AIJ**
1	1	1
1	2	2
2	1	3
2	3	4
3	2	5
3	4	6
4	3	7
4	5	8

CRS

- Stored in 3 arrays
  - J and AIJ the same length
  - I (representing rows) is compressed

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 0 & 4 & 0 & 0 \\ 0 & 5 & 0 & 6 & 0 \\ 0 & 0 & 7 & 0 & 8 \end{bmatrix}$$

I	**J**	**AIJ**
1	1	1
3	2	2
5	1	3
7	3	4
9	2	5
	4	6
	3	7
	5	8

array I : think of it as pointers to where next row starts

CCS : similar but J is compressed

CDS

- For matrices with non-zeros along sub-diagonals

Sub-diagonal index

sub-diagonals

Sub-diagonal index						
-1	0	3	7	8	9	2
0	10	9	8	7	9	-1
1	-3	6	7	5	13	0

$$A = \begin{pmatrix} 10 & -3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 0 & 0 & 8 & 7 & 5 & 0 \\ 0 & 0 & 0 & 9 & 9 & 13 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{pmatrix}.$$

Performance (Mat-vec product)

- **Notoriously bad** for running at just a fraction of the performance peak!
- Why ?

Consider mat-vec product for matrix in CRS:

```
for i = 1, n
   for j = I[i], I[i+1]-1
       y[i] += AIJ [j]  *  x [  J[j] ]
```

Performance (Mat-vec product)

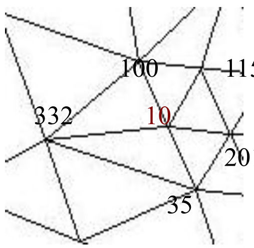
- **Notoriously bad** for running at just a fraction of the performance peak!
- Why ?

Consider mat-vec product for matrix in CRS:

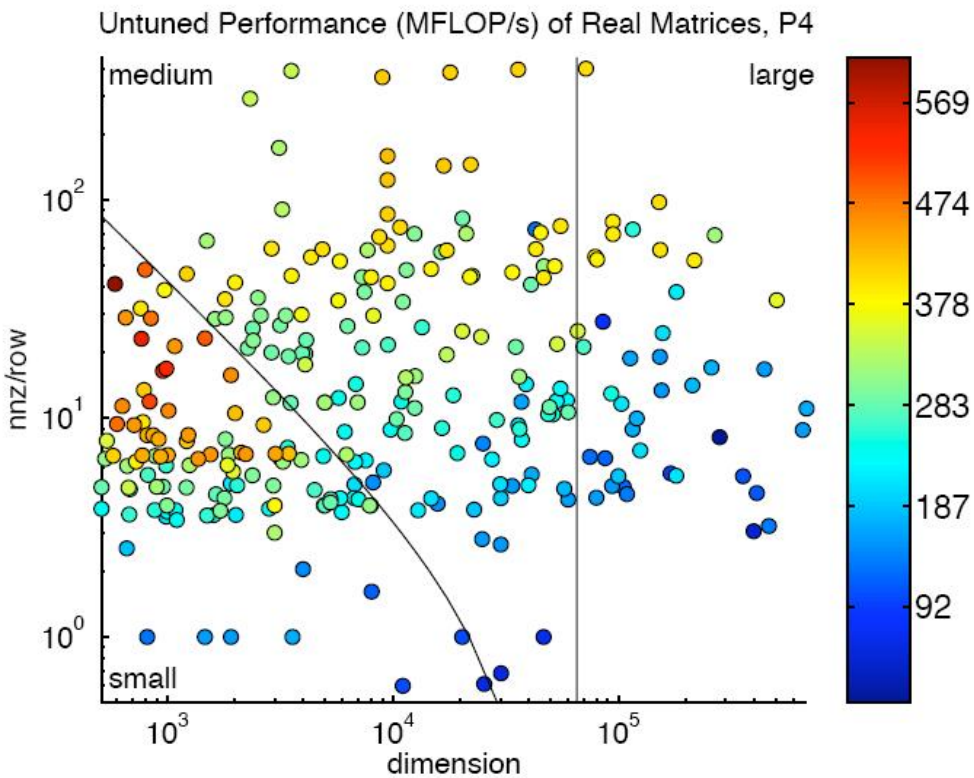
```
for i = 1, n
   for j = I[i], I[i+1]-1
      y[i] += AIJ [j]  *  x [  J[j] ]
```

* Irregular indirect memory access for x
  - result in cache trashing

* performance often
  <10% peak



Performance (Mat-vec product)



(a) Untuned SpMV performance

* Performance of mat-vec products of various sizes on a 2.4 GHz Pentium 4
* An example from Gahvari et.al.:
  http://bebop.cs.berkeley.edu/pubs/gahvari2007-spmvbench-spec.pdf

Performance (Mat-vec product)

- How to improve the performance?
  - A common technique

    (as done for dense linear algebra)

    is **blocking** (register, cache: next ...)
  - **Index reordering** (in Part II)
  - Exploit special matrix structure (e.g., symmetry, bands, other structures)

Block Compressed Row Storage (BCRS)

- Example of using 2x2 blocks

BI	BJ	AIJ
1	1	1
3	2	0
4	3	0
		4
		2
		3
		5
		0
		6
		7
		8
		9

$$\begin{bmatrix} 1 & 0 & 2 & 3 & 0 & 0 \\ 0 & 4 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 7 \\ 0 & 0 & 0 & 0 & 8 & 9 \end{bmatrix}$$

* Reduced storage for indexes

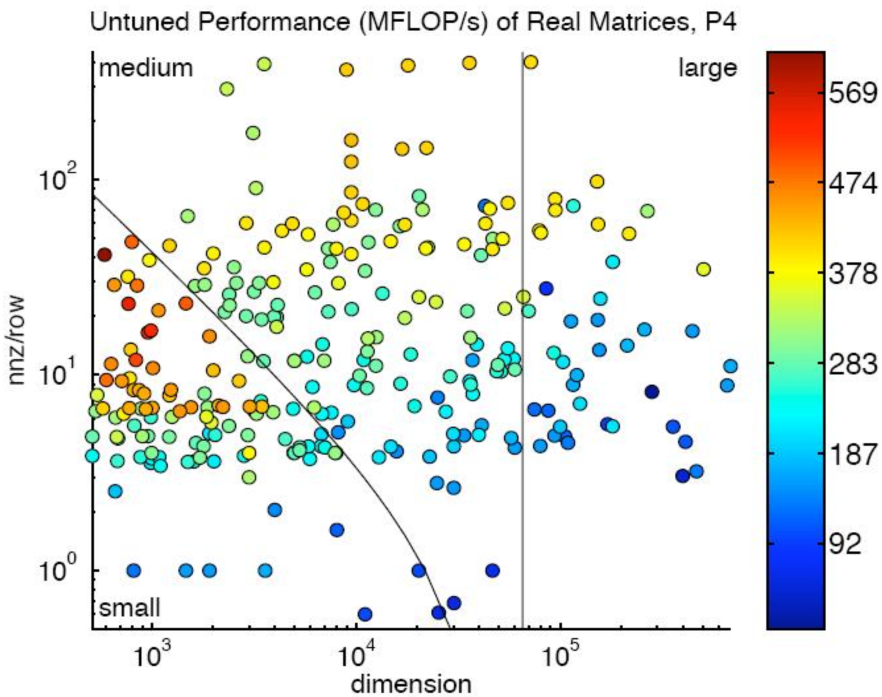
* Drawback: add 0s

* What block size to choose?

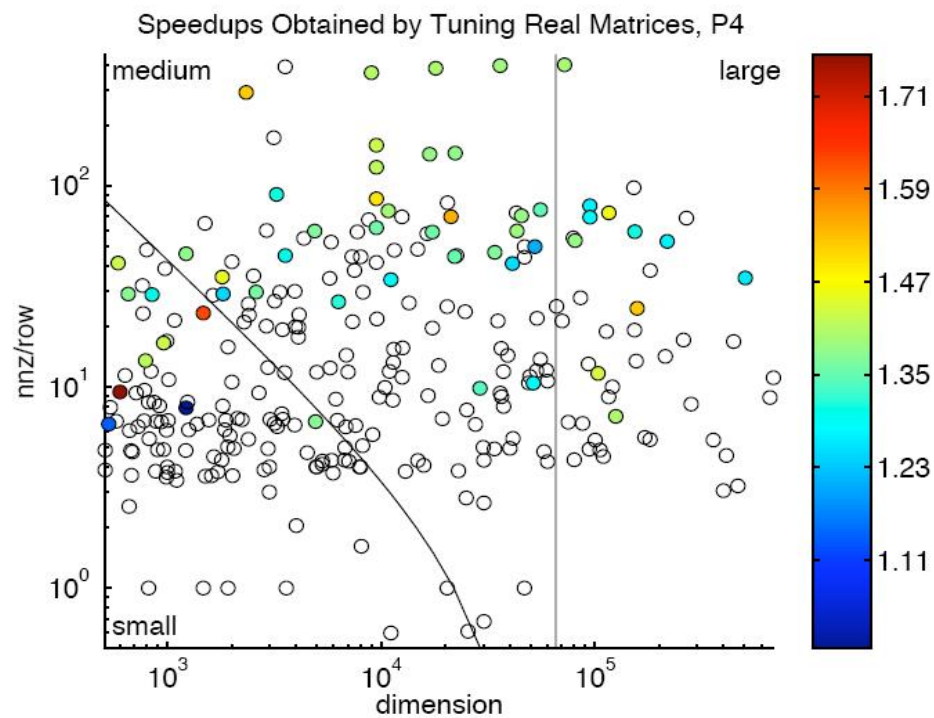
* BCRS for register blocking

* Discussion?

BCRS



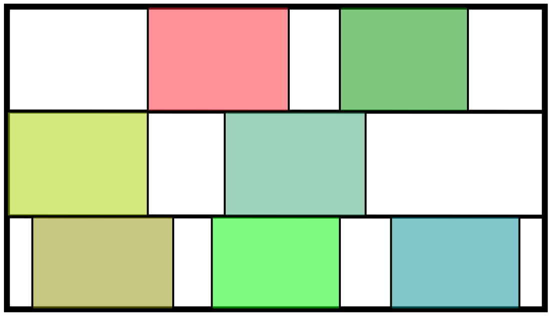
(a) Untuned SpMV performance



(b) Speedups obtained from tuning

Cache blocking

- Improve cache reuse for x in Ax by splitting A into a set of sparse matrices, e.g.



Sparse matrix and its splitting

For more info check:

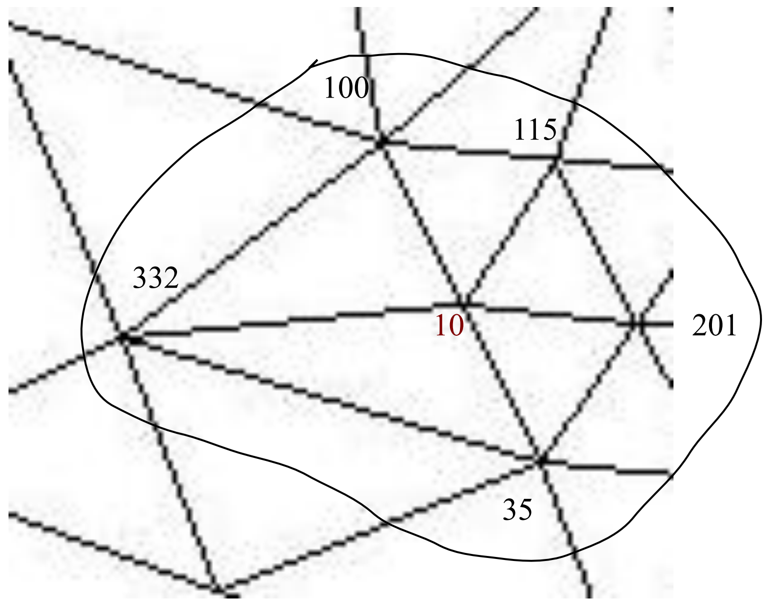
SPARSITY: An Optimization Framework for Sparse Matrix Kernels (International Journal of High Performance Computing Applications, 18 (1), pp. 135-158, February 2004.

Eun-Jin Im, K. Yelick, R. Vuduc

Part III

Reordering algorithms and Parallelization

Reorder to preserve locality

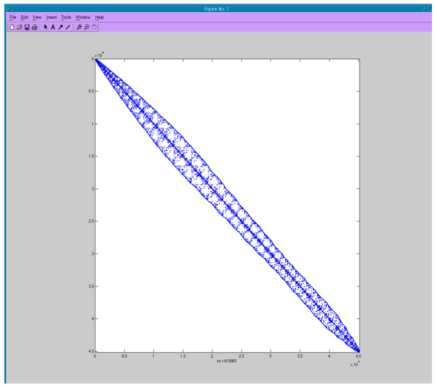


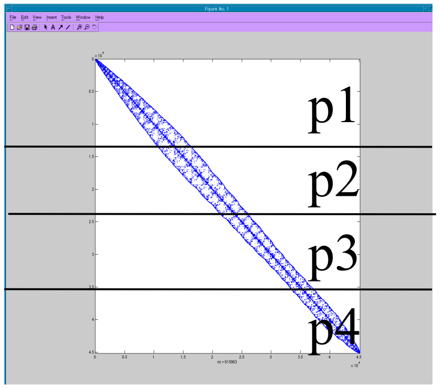
e.g., **Cuthill-McKee Ordering**: start from arbitrary node, say '10' and reorder

* '10' becomes 0
* neighbours are ordered next to become 1, 2, 3, 4, 5, denote this as level 1
* neighbours to level 1 nodes are next consecutively reordered, and so on until end

Cuthill-McKee Ordering

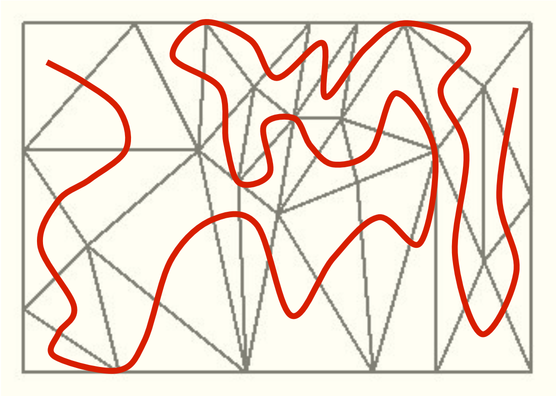
- Reversing the ordering (RCM) results in ordering that is better for sparse LU
- Reduces matrix bandwidth (see example)
- Improves cache performance
- Can be used as partitioner (**parallelization**) but in general does not reduce edge cut





Self-Avoiding Walks (SAW)

- Enumeration of mesh elements through 'consecutive elements' (sharing face, edge, vertex, etc.)



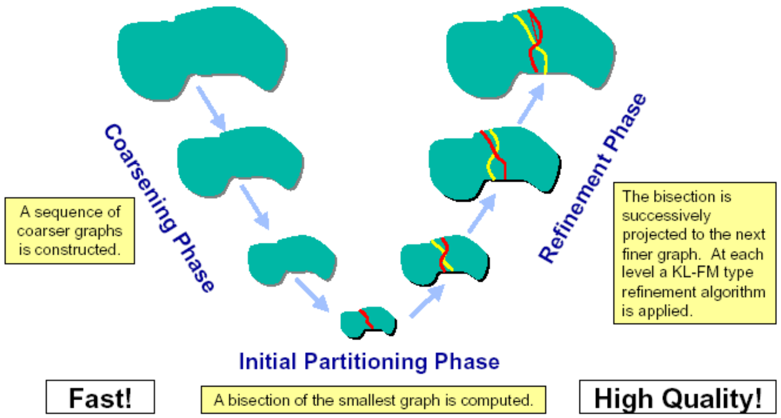
* similar to **space-filling curves** but for unstructured meshes

* improves cache reuse

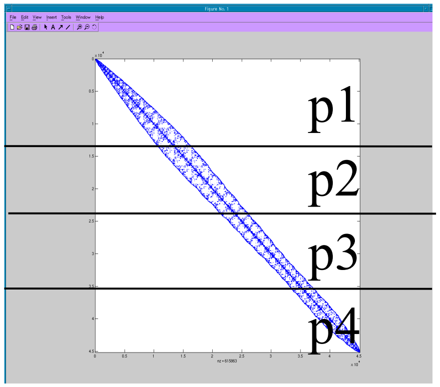
* can be used as partitioner with good load balance but in general does not reduce edge cut

Graph partitioning

- Refer back to Lecture #9, Part II
  Mesh Generation and Load Balancing
- Can be used for reordering
- Metis/ParMetis:
  - multilevel partitioning
  - Good load balance and minimize edge cut

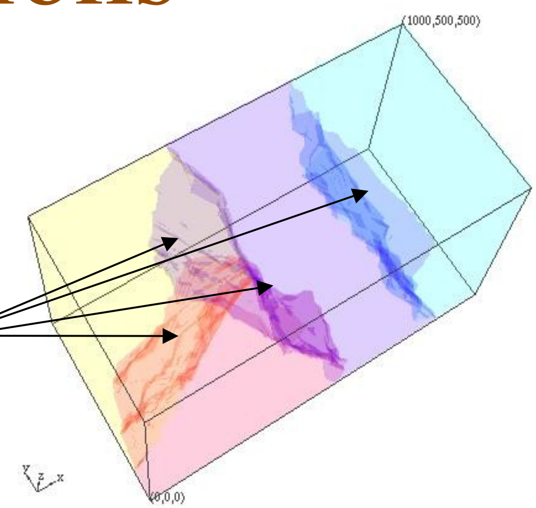


Parallel Mat-Vec Product



- Easiest way:
  - 1D partitioning
  - May lead to load unbalance (why?)
  - May need a lot of communication for x
- Can use any of the just mentioned techniques
- Most promising seems to be spectral multilevel methods (as in Metis/ParMetis)

Possible optimizations



- Block communication
  - To send the min. required part of x
  - e.g., pre-compute blocks of interfaces
- Load balance, minimize edge cut
  - eg. a good partitioner would do it
- Reordering
- Advantage of additional structure (symmetry, bands, etc)

Comparison

Distributed memory implementation

(by X. Li, L. Oliker, G. Heber, R. Biswas)

P	Avg. Cache Misses (10^6)				Avg. Comm (10^6 bytes)			
	ORIG	MeTiS	RCM	SAW	ORIG	MeTiS	RCM	SAW
8	3.684	3.034	3.749	2.004	3.228	0.011	0.031	0.049
16	2.007	1.330	1.905	0.971	2.364	0.011	0.032	0.036
32	1.060	0.658	1.017	0.507	1.492	0.009	0.032	0.030
64	0.601	0.358	0.515	0.290	0.828	0.008	0.032	0.023

- ORIG ordering has large edge cut (interprocessor comm) and poor locality (high number of cache misses)
- MeTiS minimizes edge cut, while SAW minimizes cache misses

Learning Goals

- Efficient sparse computations are challenging!
- Computational challenges and issues related to sparse matrices
  - Data formats
  - Optimization
    - Blocking
    - Reordering
    - Other
- Parallel sparse Mat-Vec product
  - Code optimization opportunities