Graph Ordering

Lecture 16

CSCI 4974/6971

27 Oct 2016

Today's Biz

1. **Reminders**
2. Review
3. Distributed Graph Processing

Reminders

- Project Update Presentation: In class November 3rd
- Assignment 4: due date TBD (early November, probably 10th)
  - Setting up and running on CCI clusters
- Assignment 5: due date TBD (before Thanksgiving break, probably 22nd)
- Assignment 6: due date TBD (early December)
- Office hours: Tuesday & Wednesday 14:00-16:00 Lally 317
  - Or email me for other availability

Today's Biz

1. Reminders
2. **Review**
3. Graph vertex ordering

Quick Review

Distributed Graph Processing

1. Can't store full graph on every node
2. Efficiently store local information - owned vertices / ghost vertices
   - Arrays for days - hashing is slow, not memory optimal
   - **Relabel vertex identifiers**
3. Vertex block, edge block, random, other partitioning strategies
4. **Partitioning strategy important for performance!!!**

Today's Biz

1. Reminders
2. Review
3. **Graph vertex ordering**

Vertex Ordering

- Idea: improve cache utilization by re-organizing adjacency list
- Idea comes from linear solvers
  - Reorder matrix for fill reduction, etc.
  - Efficient cache performance is secondary
- Many many methods, but what to optimize for?

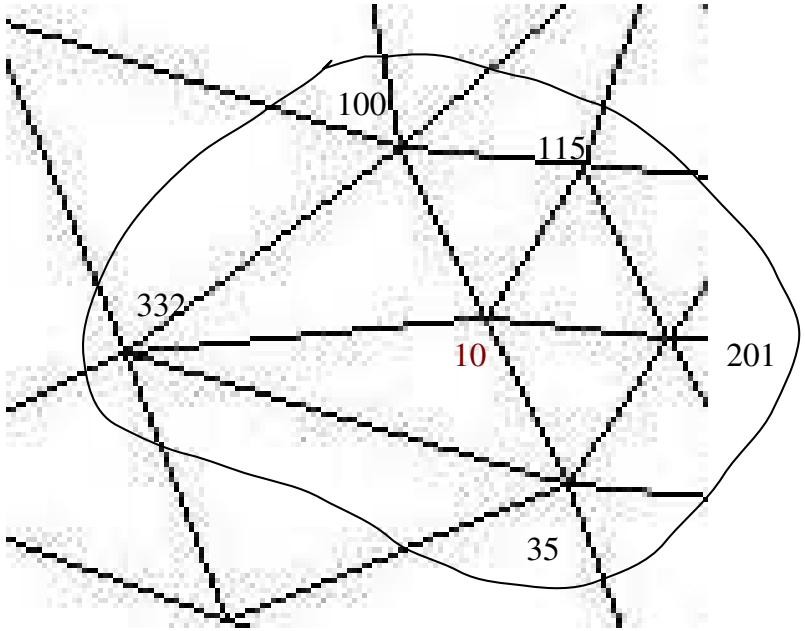
Sparse Matrices and Optimized Parallel Implementations

Slides from Stan Tomov, University of Tennessee

Part III

Reordering algorithms and Parallelization

Reorder to preserve locality

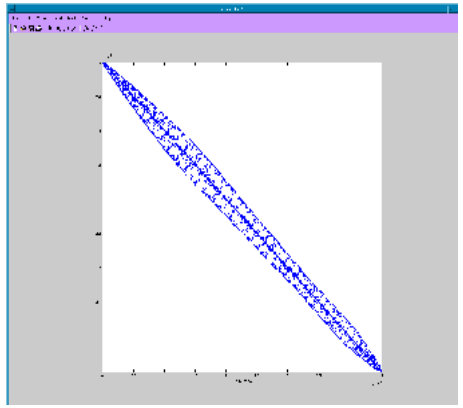


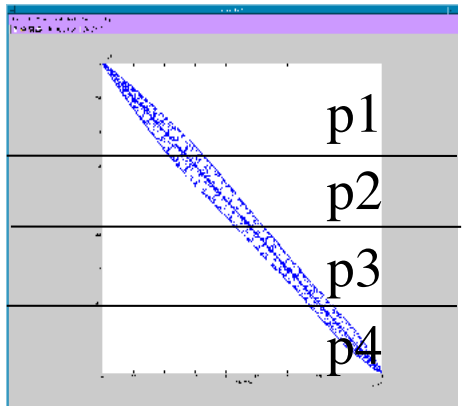
eg. **Cuthill-McKee Ordering**: start from arbitrary node, say '10' and reorder

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

Cuthill-McKee Ordering

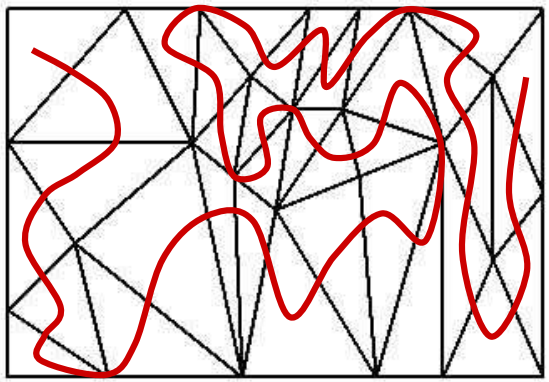
- Reversing the ordering (RCM) results in ordering that is better for sparse LU
- Reduces matrix bandwidth (see example)
- Improves cache performance
- Can be used as partitioner (**parallelization**) but in general does not reduce edge cut





Self-Avoiding Walks (SAW)

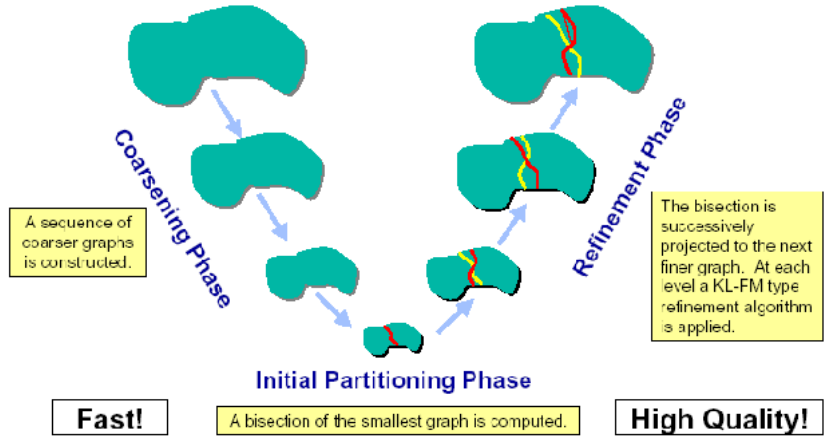
- Enumeration of mesh elements through 'consecutive elements' (sharing face, edge, vertex, etc)



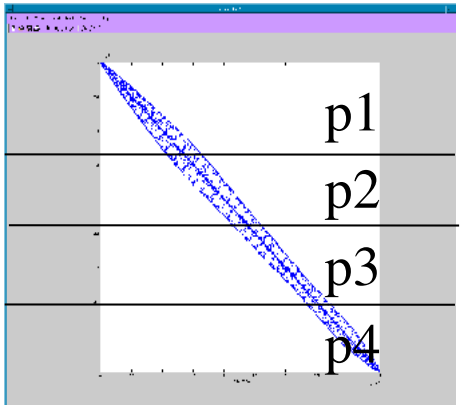
* similar to **space-filling curves** but for unstructured meshes
* improves cache reuse
* can be used as partitioner with good load balance but in general does not reduce edge cut

Graph partitioning

- Refer back to Lecture #8, Part II
  Mesh Generation and Load Balancing
- Can be used for reordering
- Metis/ParMetis:
  - multilevel partitioning
  - Good load balance and minimize edge cut



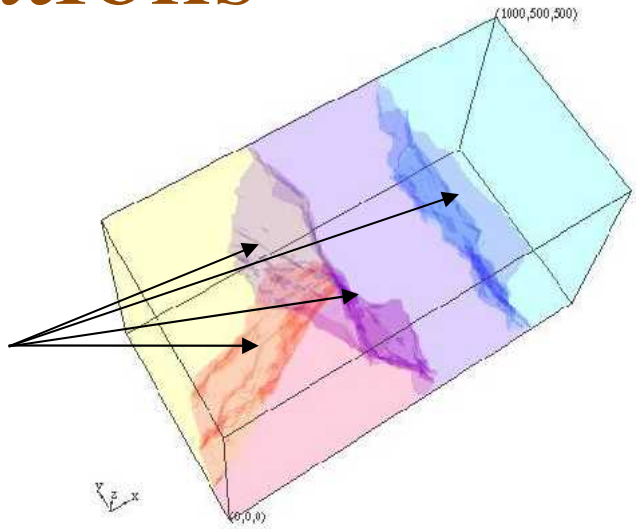
Parallel Mat-Vec Product



- 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
  - And send the min required from x
  - eg. 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) ORIG	MeTiS	RCM	SAW	Avg. Comm (10^6 bytes) 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

Matrix Bandwidth

- Bandwidth: maximum *band size*
  - Max distance between nonzeros in single row of adjacency matrix
  - In terms of graph representation: maximum distance between vertex identifiers appearing in neighborhood of a given vertex
- Is bandwidth a good measure for irregular sparse matrices?
- Does it represent cache utilization?

Other measures

- Quantifying the *gaps* in the adjacency list
  - Difficult to reduce bandwidth due to high degree vertices
  - High degree vertices will have multiple cache misses, low degrees ideally only one - want to account for both
- Minimum (linear/logarithmic) gap arrangement problem:
  - Minimize the sum of distances between vertex identifiers in the adjacency list
  - More representative of cache utilization
  - To be discussed later: impact on graph compressibility

Today: vertex ordering

- *Natural* order
- Random order
- BFS order
- RCM order
- psuedo-RCM order
- **Impacts on execution time of various graphs/algorithms**

Distributed Processing
Blank code and data available on website
(Lecture 15)
`www.cs.rpi.edu/~slotag/classes/FA16/index.html`