CS 267: Applications of Parallel Computers

Graph Partitioning

James Demmel and Kathy Yelick
www.cs.berkeley.edu/~dемmel/cs267_Spr11
Outline of Graph Partitioning Lecture

- Review definition of Graph Partitioning problem
- Overview of heuristics
- Partitioning with Nodal Coordinates
  - Ex: In finite element models, node at point in (x,y) or (x,y,z) space
- Partitioning without Nodal Coordinates
  - Ex: In model of WWW, nodes are web pages
- Multilevel Acceleration
  - BIG IDEA, appears often in scientific computing
- Comparison of Methods and Applications
- Beyond Graph Partitioning: Hypergraphs
Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
  - $N =$ nodes (or vertices),
  - $W_N =$ node weights
  - $E =$ edges
  - $W_E =$ edge weights

- Ex: $N =$ \{tasks\}, $W_N =$ \{task costs\}, edge $(j,k)$ in $E$ means task $j$ sends $W_E(j,k)$ words to task $k$

- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that
  - The sum of the node weights in each $N_j$ is “about the same”
  - The sum of all edge weights of edges connecting all different pairs $N_j$ and $N_k$ is minimized

- Ex: balance the work load, while minimizing communication

- Special case of $N = N_1 \cup N_2$: Graph Bisection
Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
  - $N$ = nodes (or vertices),
  - $W_N$ = node weights
  - $E$ = edges
  - $W_E$ = edge weights

- Ex: $N = \{\text{tasks}\}$, $W_N = \{\text{task costs}\}$, edge $(j,k)$ in $E$ means task $j$ sends $W_E(j,k)$ words to task $k$

- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that
  - The sum of the node weights in each $N_j$ is "about the same"
  - The sum of all edge weights of edges connecting all different pairs $N_j$ and $N_k$ is minimized (shown in black)

- Ex: balance the work load, while minimizing communication

- Special case of $N = N_1 \cup N_2$: Graph Bisection
Some Applications

• Telephone network design
  • Original application, algorithm due to Kernighan

• Load Balancing while Minimizing Communication

• Sparse Matrix times Vector Multiplication
  • Solving PDEs
  • \( N = \{1,\ldots,n\}, \quad (j,k) \in E \text{ if } A(j,k) \text{ nonzero, } \)
  • \( W_N(j) = \# \text{nonzeros in row } j, \quad W_E(j,k) = 1 \)

• VLSI Layout
  • \( N = \{ \text{units on chip} \}, \quad E = \{ \text{wires} \}, \quad W_E(j,k) = \text{wire length} \)

• Sparse Gaussian Elimination
  • Used to reorder rows and columns to increase parallelism, and to decrease “fill-in”

• Data mining and clustering
• Physical Mapping of DNA
• Image Segmentation
Sparse Matrix Vector Multiplication $y = y + A^*x$

Partitioning a Sparse Symmetric Matrix

... declare $A_{\text{local}}, A_{\text{remote}}(1:\text{num\_procs}), x_{\text{local}}, x_{\text{remote}}, y_{\text{local}}$

$y_{\text{local}} = y_{\text{local}} + A_{\text{local}} \times x_{\text{local}}$

for all procs $P$ that need part of $x_{\text{local}}$

send(needed part of $x_{\text{local}}$, $P$)

for all procs $P$ owning needed part of $x_{\text{remote}}$

receive($x_{\text{remote}}$, $P$)

$y_{\text{local}} = y_{\text{local}} + A_{\text{remote}}(P) \times x_{\text{remote}}$
Cost of Graph Partitioning

- Many possible partitionings to search
- Just to divide in 2 parts there are:
  \[ \binom{n}{n/2} = \frac{n!}{((n/2)!)^2} \approx \sqrt{\frac{2}{n\pi}} \cdot 2^n \] possibilities

- Choosing optimal partitioning is NP-complete
  - (NP-complete = we can prove it is a hard as other well-known hard problems in a class Nondeterministic Polynomial time)
  - Only known exact algorithms have cost = exponential(n)
- We need good heuristics
Outline of Graph Partitioning Lectures

• Review definition of Graph Partitioning problem
• Overview of heuristics
  • Partitioning with Nodal Coordinates
    • Ex: In finite element models, node at point in (x,y) or (x,y,z) space
  • Partitioning without Nodal Coordinates
    • Ex: In model of WWW, nodes are web pages
• Multilevel Acceleration
  • BIG IDEA, appears often in scientific computing
• Comparison of Methods and Applications
• Beyond Graph Partitioning: Hypergraphs
First Heuristic: Repeated Graph Bisection

- To partition $N$ into $2^k$ parts
  - bisect graph recursively $k$ times
- Henceforth discuss mostly graph bisection
Edge Separators vs. Vertex Separators

- **Edge Separator**: $E_s$ (subset of $E$) separates $G$ if removing $E_s$ from $E$ leaves two ~equal-sized, disconnected components of $N$: $N_1$ and $N_2$
- **Vertex Separator**: $N_s$ (subset of $N$) separates $G$ if removing $N_s$ and all incident edges leaves two ~equal-sized, disconnected components of $N$: $N_1$ and $N_2$

$G = (N, E)$, Nodes $N$ and Edges $E$
$E_s =$ **green edges** or **blue edges**
$N_s =$ **red vertices**

- Making an $N_s$ from an $E_s$: pick one endpoint of each edge in $E_s$
  - $|N_s| \leq |E_s|$
- Making an $E_s$ from an $N_s$: pick all edges incident on $N_s$
  - $|E_s| \leq d \times |N_s|$ where $d$ is the maximum degree of the graph
- We will find Edge or Vertex Separators, as convenient
Overview of Bisection Heuristics

• Partitioning with Nodal Coordinates
  • Each node has x,y,z coordinates \( \rightarrow \) partition space

• Partitioning without Nodal Coordinates
  • E.g., Sparse matrix of Web documents
    • \( A(j,k) = \# \) times keyword j appears in URL k

• Multilevel acceleration \( \text{(BIG IDEA)} \)
  • Approximate problem by “coarse graph,” do so recursively
Outline of Graph Partitioning Lectures

• Review definition of Graph Partitioning problem
• Overview of heuristics
• **Partitioning with Nodal Coordinates**
  • Ex: In finite element models, node at point in (x,y) or (x,y,z) space
• Partitioning without Nodal Coordinates
  • Ex: In model of WWW, nodes are web pages
• Multilevel Acceleration
  • BIG IDEA, appears often in scientific computing
• Comparison of Methods and Applications
• Beyond Graph Partitioning: Hypergraphs
Nodal Coordinates: How Well Can We Do?

- A planar graph can be drawn in plane without edge crossings
- Ex: m x m grid of \( m^2 \) nodes: \( \exists \) vertex separator \( N_s \) with \( |N_s| = m = \sqrt{|N|} \) (see earlier slide for \( m=5 \) )
- **Theorem** (Tarjan, Lipton, 1979): If \( G \) is planar, \( \exists N_S \) such that
  - \( N = N_1 \cup N_S \cup N_2 \) is a partition,
  - \( |N_1| \leq 2/3 |N| \) and \( |N_2| \leq 2/3 |N| \)
  - \( |N_S| \leq \sqrt{8 \times |N|} \)
- Theorem motivates intuition of following algorithms
Nodal Coordinates: Inertial Partitioning

• For a graph in 2D, choose line with half the nodes on one side and half on the other
  • In 3D, choose a plane, but consider 2D for simplicity

• Choose a line L, and then choose a line $L \perp$ perpendicular to it, with half the nodes on either side

1. Choose a line L through the points
   L given by $a^*(x-xbar)+b^*(y-ybar)=0$, with $a^2+b^2=1$; $(a,b)$ is unit vector $\perp$ to L

2. Project each point to the line
   For each $n_j = (x_j,y_j)$, compute coordinate
   $S_j = -b^*(x_j-xbar) + a^*(y_j-ybar)$ along L

3. Compute the median
   Let $S_{bar} = median(S_1,\ldots,S_n)$

4. Use median to partition the nodes
   Let nodes with $S_j < S_{bar}$ be in $N_1$, rest in $N_2$
Inertial Partitioning: Choosing L

• Clearly prefer L, $L \perp$ on left below

• Mathematically, choose L to be a total least squares fit of the nodes
  • Minimize sum of squares of distances to L (green lines on last slide)
  • Equivalent to choosing L as axis of rotation that minimizes the moment of inertia of nodes (unit weights) - source of name
Inertial Partitioning: choosing L (continued)

\[ \Sigma_j \text{ (length of j-th green line)}^2 = \Sigma_j \left( (x_j - \bar{x})^2 + (y_j - \bar{y})^2 - (-b(x_j - \bar{x}) + a(y_j - \bar{y}))^2 \right) \]

\[ \text{... Pythagorean Theorem} \]

\[ = a^2 \Sigma_j (x_j - \bar{x})^2 + 2ab \Sigma_j (x_j - \bar{x})(y_j - \bar{y}) + b^2 \Sigma_j (y_j - \bar{y})^2 \]

\[ = [a \ b] \begin{bmatrix} X_1 & X_2 \\ X_2 & X_3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \]

Minimized by choosing
\[ (xbar, ybar) = (\Sigma_j x_j, \Sigma_j y_j) / n = \text{center of mass} \]
\[ (a, b) = \text{eigenvector of smallest eigenvalue of} \begin{bmatrix} X_1 & X_2 \\ X_2 & X_3 \end{bmatrix} \]

\( \bar{L} \)

\( (xbar, ybar) \)

\( (a, b) \) is unit vector perpendicular to \( L \)
Nodal Coordinates: Random Spheres

• Generalize nearest neighbor idea of a planar graph to higher dimensions
  • Any graph can fit in 3D without edge crossings
  • Capture intuition of planar graphs of being connected to “nearest neighbors” but in higher than 2 dimensions

• For intuition, consider graph defined by a regular 3D mesh
  • An n by n by n mesh of \( |N| = n^3 \) nodes
  • Edges to 6 nearest neighbors
  • Partition by taking plane parallel to 2 axes
  • Cuts \( n^2 = |N|^{2/3} = O(|E|^{2/3}) \) edges

• For the general graphs
  • Need a notion of “well-shaped” like mesh
Random Spheres: Well Shaped Graphs

• Approach due to Miller, Teng, Thurston, Vavasis

• **Def:** A k-ply neighborhood system in d dimensions is a set \( \{D_1, \ldots, D_n\} \) of closed disks in \( \mathbb{R}^d \) such that no point in \( \mathbb{R}^d \) is strictly interior to more than k disks

• **Def:** An \((\alpha,k)\) overlap graph is a graph defined in terms of \( \alpha \geq 1 \) and a k-ply neighborhood system \( \{D_1, \ldots, D_n\} \):
  There is a node for each \( D_j \), and an edge from j to i if expanding the radius of the smaller of \( D_j \) and \( D_i \) by \( >\alpha \) causes the two disks to overlap

Ex: n-by-n mesh is a (1,1) overlap graph
Ex: Any planar graph is \((\alpha,k)\) overlap for some \( \alpha,k \)

2D Mesh is (1,1) overlap graph
Generalizing Lipton/Tarjan to Higher Dimensions

• **Theorem** (Miller, Teng, Thurston, Vavasis, 1993): Let $G=(N,E)$ be an $(\alpha,k)$ overlap graph in $d$ dimensions with $n=|N|$. Then there is a vertex separator $N_s$ such that
  - $N = N_1 \cup N_s \cup N_2$ and
  - $N_1$ and $N_2$ each has at most $n*(d+1)/(d+2)$ nodes
  - $N_s$ has at most $O(\alpha * k^{1/d} * n^{(d-1)/d})$ nodes

• When $d=2$, same as Lipton/Tarjan

• Algorithm:
  - Choose a sphere $S$ in $\mathbb{R}^d$
  - Edges that $S$ “cuts” form edge separator $E_S$
  - Build $N_s$ from $E_S$
  - Choose $S$ “randomly”, so that it satisfies Theorem with high probability
Stereographic Projection

- Stereographic projection from plane to sphere
  - In $d=2$, draw line from $p$ to North Pole, projection $p'$ of $p$ is where the line and sphere intersect

$$p = (x,y) \quad p' = \frac{(2x, 2y, x^2 + y^2 - 1)}{(x^2 + y^2 + 1)}$$

- Similar in higher dimensions
Choosing a Random Sphere

- Do stereographic projection from $\mathbb{R}^d$ to sphere $S$ in $\mathbb{R}^{d+1}$
- Find *centerpoint* of projected points
  - Any plane through centerpoint divides points \textasciicircum~evenly
  - There is a linear programming algorithm, cheaper heuristics
- *Conformally map* points on sphere
  - *Rotate* points around origin so centerpoint at $(0,\ldots,0,r)$ for some $r$
  - *Dilate* points (unproject, multiply by $\sqrt{(1-r)/(1+r)}$), project
    - this maps centerpoint to origin $(0,\ldots,0)$, spreads points around $S$
- Pick a random plane through origin
  - Intersection of plane and sphere $S$ is “circle”
- Unproject circle
  - yields desired circle $C$ in $\mathbb{R}^d$
- Create $N_s$: $j$ belongs to $N_s$ if $\alpha^*D_j$ intersects $C$
Random Sphere Algorithm (Gilbert)
Random Sphere Algorithm (Gilbert)
Figure 3: Projected mesh points. The large dot is the centerpoint.
Random Sphere Algorithm (Gilbert)
Random Sphere Algorithm (Gilbert)
Figure 5: The separating circle projected back to the plane.
Nodal Coordinates: Summary

- Other variations on these algorithms
- Algorithms are efficient
- Rely on graphs having nodes connected (mostly) to “nearest neighbors” in space
  - algorithm does not depend on where actual edges are!
- Common when graph arises from physical model
- Ignores edges, but can be used as good starting guess for subsequent partitioners that do examine edges
- Can do poorly if graph connection is not spatial:

- Details at
  - www.cs.berkeley.edu/~demmel/cs267/lecture18/lecture18.html
  - www.cs.ucsb.edu/~gilbert
  - www.cs.bu.edu/~steng
Outline of Graph Partitioning Lectures

• Review definition of Graph Partitioning problem
• Overview of heuristics
• Partitioning with Nodal Coordinates
  • Ex: In finite element models, node at point in (x,y) or (x,y,z) space
• Partitioning without Nodal Coordinates
  • Ex: In model of WWW, nodes are web pages
• Multilevel Acceleration
  • BIG IDEA, appears often in scientific computing
• Comparison of Methods andApplications
• Beyond Graph Partitioning: Hypergraphs
Coordinate-Free: Breadth First Search (BFS)

- Given $G(N,E)$ and a root node $r$ in $N$, BFS produces
  - A subgraph $T$ of $G$ (same nodes, subset of edges)
  - $T$ is a tree rooted at $r$
  - Each node assigned a level = distance from $r$
Breadth First Search (details)

- Queue (First In First Out, or FIFO)
  - Enqueue(x,Q) adds x to back of Q
  - x = Dequeue(Q) removes x from front of Q

- Compute Tree $T(N_T,E_T)$

\[
N_T = \{(r,0)\}, \quad E_T = \text{empty set} \\
\text{Enqueue}((r,0),Q) \\
\text{Mark } r \\
\text{While Q not empty} \\
\quad (n,\text{level}) = \text{Dequeue}(Q) \\
\quad \text{For all unmarked children } c \text{ of } n \\
\quad \quad N_T = N_T \cup (c,\text{level}+1) \\
\quad \quad E_T = E_T \cup (n,c) \\
\quad \quad \text{Enqueue}((c,\text{level}+1),Q) \\
\quad \quad \text{Mark } c
\]

... Initially $T = \text{root } r$, which is at level 0
... Put root on initially empty Queue Q
... Mark root as having been processed
... While nodes remain to be processed
... Get a node to process
... Add child c to $N_T$
... Add edge (n,c) to $E_T$
... Add child c to Q for processing
... Mark c as processed
Partitioning via Breadth First Search

- BFS identifies 3 kinds of edges
  - Tree Edges - part of T
  - Horizontal Edges - connect nodes at same level
  - Interlevel Edges - connect nodes at adjacent levels
- No edges connect nodes in levels differing by more than 1 (why?)
- BFS partitioning heuristic
  - \( N = N_1 \cup N_2 \), where
    - \( N_1 = \{\text{nodes at level} \leq L\} \)
    - \( N_2 = \{\text{nodes at level} > L\} \)
  - Choose \( L \) so \( |N_1| \) close to \( |N_2| \)

BFS partition of a 2D Mesh using center as root:
- \( N_1 = \text{levels 0, 1, 2, 3} \)
- \( N_2 = \text{levels 4, 5, 6} \)
Coordinate-Free: Kernighan/Lin

• Take a initial partition and iteratively improve it
  • Kernighan/Lin (1970), cost = \(O(|N|^3)\) but easy to understand
  • Fiduccia/Mattheyses (1982), cost = \(O(|E|)\), much better, but more complicated

• Given \(G = (N,E,W_E)\) and a partitioning \(N = A \cup B\), where \(|A| = |B|\)
  • \(T = \text{cost}(A,B) = \sum \{W(e) \text{ where } e \text{ connects nodes in } A \text{ and } B\}\)
  • Find subsets \(X\) of \(A\) and \(Y\) of \(B\) with \(|X| = |Y|\)
  • Consider swapping \(X\) and \(Y\) if it decreases cost:
    • \(\text{newA} = (A - X) \cup Y\) and \(\text{newB} = (B - Y) \cup X\)
    • \(\text{newT} = \text{cost(\text{newA} , \text{newB}) < T = \text{cost(A,B)}}\)

• Need to compute \(\text{newT}\) efficiently for many possible \(X\) and \(Y\), choose smallest (best)
Kernighan/Lin: Preliminary Definitions

• \( T = \text{cost}(A, B), \quad \text{new}T = \text{cost}(\text{new}A, \text{new}B) \)

• Need an efficient formula for \( \text{new}T \); will use
  • \( E(a) = \text{external cost of } a \text{ in } A = \sum \{W(a,b) \text{ for } b \text{ in } B\} \)
  • \( I(a) = \text{internal cost of } a \text{ in } A = \sum \{W(a,a’) \text{ for other } a’ \text{ in } A\} \)
  • \( D(a) = \text{cost of } a \text{ in } A = E(a) - I(a) \)
  • \( E(b), I(b) \) and \( D(b) \) defined analogously for \( b \text{ in } B \)

• Consider swapping \( X = \{a\} \) and \( Y = \{b\} \)
  • \( \text{new}A = (A - \{a\}) U \{b\}, \quad \text{new}B = (B - \{b\}) U \{a\} \)
  • \( \text{new}T = T - (D(a) + D(b) - 2\times w(a,b)) \equiv T - \text{gain}(a,b) \)
    • \( \text{gain}(a,b) \) measures improvement gotten by swapping \( a \) and \( b \)

• Update formulas
  • \( \text{new}D(a’) = D(a’) + 2\times w(a’,a) - 2\times w(a’,b) \quad \text{for } a’ \text{ in } A, a’ \neq a \)
  • \( \text{new}D(b’) = D(b’) + 2\times w(b’,b) - 2\times w(b’,a) \quad \text{for } b’ \text{ in } B, b’ \neq b \)
Kernighan/Lin Algorithm

Compute $T = \text{cost}(A, B)$ for initial $A, B$ \hspace{1cm} \ldots \text{cost} = O(|N|^2)

Repeat

\begin{itemize}
  \item \text{... One pass greedily computes $|N|/2$ possible $X,Y$ to swap, picks best}
  \item Compute costs $D(n)$ for all $n$ in $N$ \hspace{1cm} \ldots \text{cost} = O(|N|^2)
  \item Unmark all nodes in $N$ \hspace{1cm} \ldots \text{cost} = O(|N|)
  \item While there are unmarked nodes \hspace{1cm} \ldots |N|/2 iterations
    \begin{itemize}
      \item Find an unmarked pair $(a,b)$ maximizing $\text{gain}(a,b)$ \hspace{1cm} \ldots \text{cost} = O(|N|^2)
      \item Mark $a$ and $b$ (but do not swap them) \hspace{1cm} \ldots \text{cost} = O(1)
      \item Update $D(n)$ for all unmarked $n$, \hspace{1cm} \ldots \text{cost} = O(|N|)
        \begin{itemize}
          \item as though $a$ and $b$ had been swapped
        \end{itemize}
    \end{itemize}
  \end{itemize}
Endwhile

\begin{itemize}
  \item \text{... At this point we have computed a sequence of pairs}
  \item \text{... $(a_1,b_1), \ldots, (a_k,b_k)$ and gains $\text{gain}(1), \ldots, \text{gain}(k)$}
  \item \text{... where $k = |N|/2$, numbered in the order in which we marked them}
  \item Pick $m$ maximizing $\text{Gain} = \sum_{k=1}^{m} \text{gain}(k)$ \hspace{1cm} \ldots \text{cost} = O(|N|)
  \item \text{... Gain is reduction in cost from swapping $(a_1,b_1)$ through $(a_m,b_m)$}
  \item If $\text{Gain} > 0$ then \hspace{1cm} \ldots \text{it is worth swapping}
    \begin{itemize}
      \item Update $\text{newA} = A - \{ a_1, \ldots, a_m \} \cup \{ b_1, \ldots, b_m \}$ \hspace{1cm} \ldots \text{cost} = O(|N|)
      \item Update $\text{newB} = B - \{ b_1, \ldots, b_m \} \cup \{ a_1, \ldots, a_m \}$ \hspace{1cm} \ldots \text{cost} = O(|N|)
      \item Update $T = T - \text{Gain}$ \hspace{1cm} \ldots \text{cost} = O(1)
    \end{itemize}
  \item endif
\end{itemize}

Until $\text{Gain} \leq 0$
Comments on Kernighan/Lin Algorithm

• Most expensive line shown in red, $O(n^3)$

• Some gain(k) may be negative, but if later gains are large, then final Gain may be positive
  • can escape “local minima” where switching no pair helps

• How many times do we Repeat?
  • K/L tested on very small graphs ($|N|\leq 360$) and got convergence after 2-4 sweeps
  • For random graphs (of theoretical interest) the probability of convergence in one step appears to drop like $2^{-|N|/30}$
Coordinate-Free: Spectral Bisection

• Based on theory of Fiedler (1970s), popularized by Pothen, Simon, Liou (1990)
• Motivation, by analogy to a vibrating string
• Basic definitions
• Vibrating string, revisited
• Implementation via the Lanczos Algorithm
  • To optimize sparse-matrix-vector multiply, we graph partition
  • To graph partition, we find an eigenvector of a matrix associated with the graph
  • To find an eigenvector, we do sparse-matrix vector multiply
  • No free lunch ...
Motivation for Spectral Bisection

- Vibrating string
- Think of $G = 1$D mesh as masses (nodes) connected by springs (edges), i.e. a string that can vibrate
- Vibrating string has \textit{modes of vibration}, or \textit{harmonics}
- Label nodes by whether mode - or + to partition into N- and N+
- Same idea for other graphs (eg planar graph \textasciitilde trampoline)

\begin{center}
\textbf{Modes of a Vibrating String}
\end{center}

+ +
\begin{center}
\text{Lowest Frequency $\lambda(1)$}
\end{center}

+ +
\begin{center}
\text{Second Frequency $\lambda(2)$}
\end{center}

+ +
\begin{center}
\text{Third Frequency $\lambda(3)$}
\end{center}
Basic Definitions

• **Definition**: The incidence matrix $\mathbf{In}(G)$ of a graph $G(N,E)$ is an $|N|$ by $|E|$ matrix, with one row for each node and one column for each edge. If edge $e=\langle i,j \rangle$ then column $e$ of $\mathbf{In}(G)$ is zero except for the $i$-th and $j$-th entries, which are $+1$ and $-1$, respectively.

• Slightly ambiguous definition because multiplying column $e$ of $\mathbf{In}(G)$ by $-1$ still satisfies the definition, but this won’t matter...

• **Definition**: The Laplacian matrix $\mathbf{L}(G)$ of a graph $G(N,E)$ is an $|N|$ by $|N|$ symmetric matrix, with one row and column for each node. It is defined by

  • $\mathbf{L}(G) (i,i) = \text{degree of node } i$ (number of incident edges)
  • $\mathbf{L}(G) (i,j) = -1$ if $i \neq j$ and there is an edge $\langle i,j \rangle$
  • $\mathbf{L}(G) (i,j) = 0$ otherwise
Example of $\text{In}(G)$ and $L(G)$ for Simple Meshes

Incidence and Laplacian Matrices

**Graph G**

**Incidence Matrix $\text{In}(G)$**

$$
\begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & -1 & & \\
2 & 1 & -1 & \\
3 & & 1 & -1 \\
4 & & & 1 & -1 \\
5 & & & & 1
\end{bmatrix}
$$

**Laplacian Matrix $L(G)$**

$$
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 1 & -1 & & \\
2 & -1 & 2 & -1 & \\
3 & & -1 & 2 & -1 \\
4 & & & -1 & 2 & -1 \\
5 & & & & -1 & 1
\end{bmatrix}
$$

Nodes numbered in **black**

Edges numbered in **blue**
Properties of Incidence and Laplacian matrices

• **Theorem 1**: Given $G$, $\text{In}(G)$ and $L(G)$ have the following properties
  (proof on Demmel’s 1996 CS267 web page)
  
  • $L(G)$ is symmetric. (This means the eigenvalues of $L(G)$ are real and its eigenvectors are real and orthogonal.)
  
  • Let $e = [1, \ldots, 1]^T$, i.e. the column vector of all ones. Then $L(G)^*e=0$.
  
  • $\text{In}(G) \ast (\text{In}(G))^T = L(G)$. This is independent of the signs chosen for each column of $\text{In}(G)$.
  
  • Suppose $L(G)^*v = \lambda^*v$, $v \neq 0$, so that $v$ is an eigenvector and $\lambda$ an eigenvalue of $L(G)$. Then
    
    \[
    \lambda = ||\text{In}(G)^T \ast v||^2 / ||v||^2
    = \sum \{ (v(i)-v(j))^2 \text{ for all edges } e=(i,j) \} / \sum_i v(i)^2
    \]
    
    • The eigenvalues of $L(G)$ are nonnegative:
      
      • $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$
    
    • The number of connected components of $G$ is equal to the number of $\lambda_i$ equal to 0. In particular, $\lambda_2 \neq 0$ if and only if $G$ is connected.
  
  • **Definition**: $\lambda_2(L(G))$ is the **algebraic connectivity** of $G$
Properties of Laplacian Matrix

• **Theorem 1:** Given G, L(G) has the following properties  
  (proof on 1996 CS267 web page)
  
  • L(G) is symmetric.
    • This means the eigenvalues of L(G) are real and its eigenvectors are real and orthogonal.
  • \( \text{In}(G) \times (\text{In}(G))^T = L(G) \)
  • The eigenvalues of L(G) are nonnegative:
    • \( 0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \)
  • The number of connected components of G is equal to the number of \( \lambda_i \) equal to 0.
  • **Definition:** \( \lambda_2(L(G)) \) is the algebraic connectivity of G
    • The magnitude of \( \lambda_2 \) measures connectivity
    • In particular, \( \lambda_2 \neq 0 \) if and only if G is connected.
Spectral Bisection Algorithm

• Spectral Bisection Algorithm:
  • Compute eigenvector \( v_2 \) corresponding to \( \lambda_2(L(G)) \)
  • For each node \( n \) of \( G \)
    • if \( v_2(n) < 0 \) put node \( n \) in partition \( N^- \)
    • else put node \( n \) in partition \( N^+ \)

• Why does this make sense? First reasons...
  • Theorem 2 (Fiedler, 1975): Let \( G \) be connected, and \( N^- \) and \( N^+ \) defined as above. Then \( N^- \) is connected. If no \( v_2(n) = 0 \), then \( N^+ \) is also connected. (proof on 1996 CS267 web page)
  • Recall \( \lambda_2(L(G)) \) is the algebraic connectivity of \( G \)
  • Theorem 3 (Fiedler): Let \( G_1(N,E_1) \) be a subgraph of \( G(N,E) \), so that \( G_1 \) is “less connected” than \( G \). Then \( \lambda_2(L(G_1)) \leq \lambda_2(L(G)) \), i.e. the algebraic connectivity of \( G_1 \) is less than or equal to the algebraic connectivity of \( G \). (proof on 1996 CS267 web page)
Spectral Bisection Algorithm

• Spectral Bisection Algorithm:
  • Compute eigenvector $v_2$ corresponding to $\lambda_2(L(G))$
  • For each node $n$ of $G$
    • if $v_2(n) < 0$ put node $n$ in partition $N-$
    • else put node $n$ in partition $N+$

• Why does this make sense? More reasons...
  • Theorem 4 (Fiedler, 1975): Let $G$ be connected, and $N_1$ and $N_2$ be any partition into part of equal size $|N|/2$. Then the number of edges connecting $N_1$ and $N_2$ is at least $0.25 \times |N| \times \lambda_2(L(G))$. (proof on 1996 CS267 web page)
Motivation for Spectral Bisection (recap)

• Vibrating string has modes of vibration, or harmonics
• Modes computable as follows
  • Model string as masses connected by springs (a 1D mesh)
  • Write down F=ma for coupled system, get matrix A
  • Eigenvalues and eigenvectors of A are frequencies and shapes of modes
• Label nodes by whether mode - or + to get N- and N+
• Same idea for other graphs (eg planar graph ~ trampoline)

Modes of a Vibrating String

- Lowest Frequency $\lambda(1)$
- Second Frequency $\lambda(2)$
- Third Frequency $\lambda(3)$
Details for Vibrating String Analogy

• Force on mass \( j = k^* [x(j-1) - x(j)] + k^* [x(j+1) - x(j)] \)
  \[= -k^* [-x(j-1) + 2x(j) - x(j+1)] \]

• \( F=ma \) yields \( m^* x''(j) = -k^* [-x(j-1) + 2x(j) - x(j+1)] \) \((*)\)

• Writing \((*)\) for \( j=1,2,...,n \) yields

\[
m^* \frac{d^2}{dx^2} \begin{pmatrix} x(1) \\ x(2) \\ \vdots \\ x(j) \\ \vdots \\ x(n) \end{pmatrix} = -k^* \begin{pmatrix} 2 & -1 \\ -1 & 2 & -1 \\ \vdots & \vdots & \vdots \\ -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} x(1) \\ x(2) \\ \vdots \\ x(j) \\ \vdots \\ x(n) \end{pmatrix} = -k^* L \begin{pmatrix} x(1) \\ x(2) \\ \vdots \\ x(n) \end{pmatrix}
\]

\((-m/k) x'' = L^* x\)

Vibrating Mass Spring System
Details for Vibrating String (continued)

• \(-\frac{m}{k} \frac{d^2}{dt^2} x = L^*x\), where \(x = [x_1, x_2, \ldots, x_n]^T\)

• Seek solution of form \(x(t) = \sin(\alpha t) \cdot x_0\)
  
  • \(L^*x_0 = \frac{m}{k} \alpha^2 x_0 = \lambda x_0\)
  
  • For each integer \(i\), get \(\lambda = 2(1 - \cos(i\pi/(n+1))\), \(x_0 = \begin{pmatrix}
\sin(1i\pi/(n+1)) \\
\sin(2i\pi/(n+1)) \\
\vdots \\
\sin(ni\pi/(n+1))
\end{pmatrix}\)

• Thus \(x_0\) is a sine curve with frequency proportional to \(i\)

• Thus \(\alpha^2 = 2\frac{k}{m}(1 - \cos(i\pi/(n+1))\) or \(\alpha \sim \sqrt{\frac{k}{m}}\frac{\pi i}{(n+1)}\)

• \(L = \begin{pmatrix}
2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 \\
\vdots \\
-1 & 2
\end{pmatrix}\) not quite Laplacian of 1D mesh, but we can fix that ...
Motivation for Spectral Bisection

- Vibrating string has modes of vibration, or harmonics

- Modes computable as follows
  - Model string as masses connected by springs (a 1D mesh)
  - Write down $F=ma$ for coupled system, get matrix $A$
  - Eigenvalues and eigenvectors of $A$ are frequencies and shapes of modes

- Label nodes by whether mode - or + to get N- and N+

- Same idea for other graphs (eg planar graph ~ trampoline)

"Vibrating String" for Spectral Bisection
Eigenvectors of $L(1D$ mesh$)$

Eigenvector 1
(all ones)

Eigenvector 2

Eigenvector 3
2nd eigenvector of L(planar mesh)
4th eigenvector of L(planar mesh)
Computing $v_2$ and $\lambda_2$ of $L(G)$ using Lanczos

- Given any $n$-by-$n$ symmetric matrix $A$ (such as $L(G)$), Lanczos computes a $k$-by-$k$ “approximation” $T$ by doing $k$ matrix-vector products, $k \ll n$

Choose an arbitrary starting vector $r$

$b(0) = ||r||$

$j=0$

repeat

$j=j+1$

$q(j) = r/b(j-1)$ … scale a vector (BLAS1)

$r = A^*q(j)$ … matrix vector multiplication, the most expensive step

$r = r - b(j-1)*v(j-1)$ … “axpy”, or scalar*vector + vector (BLAS1)

$a(j) = v(j)^T * r$ … dot product (BLAS1)

$r = r - a(j)*v(j)$ … “axpy” (BLAS1)

$b(j) = ||r||$ … compute vector norm (BLAS1)

until convergence … details omitted

\[
T = \begin{pmatrix}
  a(1) & b(1) & & & \\
  b(1) & a(2) & b(2) & & \\
  b(2) & a(3) & b(3) & & \\
  & \cdots & \cdots & \cdots & \\
  & & & \cdots & b(k-2)
\end{pmatrix}
\]

- Approximate $A$’s eigenvalues/vectors using $T$’s
Spectral Bisection: Summary

- Laplacian matrix represents graph connectivity
- Second eigenvector gives a graph bisection
  - Roughly equal “weights” in two parts
  - Weak connection in the graph will be separator
- Implementation via the Lanczos Algorithm
  - To optimize sparse-matrix-vector multiply, we graph partition
  - To graph partition, we find an eigenvector of a matrix associated with the graph
  - To find an eigenvector, we do sparse-matrix vector multiply
- Have we made progress?
  - The first matrix-vector multiplies are slow, but use them to learn how to make the rest faster
Outline of Graph Partitioning Lectures

- Review definition of Graph Partitioning problem
- Overview of heuristics
- Partitioning with Nodal Coordinates
  - Ex: In finite element models, node at point in (x,y) or (x,y,z) space
- Partitioning without Nodal Coordinates
  - Ex: In model of WWW, nodes are web pages
- Multi-level Acceleration
  - BIG IDEA, appears often in scientific computing
- Comparison of Methods and Applications
- Beyond Graph Partitioning: Hypergraphs
Introduction to Multilevel Partitioning

• If we want to partition G(N,E), but it is too big to do efficiently, what can we do?
  • 1) Replace G(N,E) by a coarse approximation G_c(N_c,E_c), and partition G_c instead
  • 2) Use partition of G_c to get a rough partitioning of G, and then iteratively improve it

• What if G_c still too big?
  • Apply same idea recursively
Multilevel Partitioning - High Level Algorithm

\[(N^+, N^-) = \text{Multilevel\_Partition}(N, E)\]

... recursive partitioning routine returns \(N^+\) and \(N^-\) where \(N = N^+ \cup N^-\)

if \(|N|\) is small

(1) Partition \(G = (N, E)\) directly to get \(N = N^+ \cup N^-\)
    Return \((N^+, N^-)\)

else

(2) **Coarsen** \(G\) to get an approximation \(G_c = (N_c, E_c)\)
(3) \((N_c^+, N_c^-) = \text{Multilevel\_Partition}(N_c, E_c)\)
(4) **Expand** \((N_c^+, N_c^-)\) to a partition \((N^+, N^-)\) of \(N\)
(5) **Improve** the partition \((N^+, N^-)\)
    Return \((N^+, N^-)\)

endif

“V - cycle:”

**How do we**

Coarsen?  Expand?  Improve?
Multilevel Kernighan-Lin

• **Coarsen** graph and **expand** partition using **maximal matchings**

• **Improve** partition using **Kernighan-Lin**
Maximal Matching

- **Definition**: A matching of a graph G(N,E) is a subset E_m of E such that no two edges in E_m share an endpoint.

- **Definition**: A maximal matching of a graph G(N,E) is a matching E_m to which no more edges can be added and remain a matching.

- A simple greedy algorithm computes a maximal matching:

```plaintext
let E_m be empty
mark all nodes in N as unmatched
for i = 1 to |N| ... visit the nodes in any order
    if i has not been matched
        mark i as matched
        if there is an edge e=(i,j) where j is also unmatched,
            add e to E_m
            mark j as matched
    endif
endfor
```
Maximal Matching: Example
Example of Coarsening

How to coarsen a graph using a maximal matching

\[ G = (N, E) \]

\( E_m \) is shown in red

Edge weights shown in blue

Node weights are all one

\[ G_c = (N_c, E_c) \]

\( N_c \) is shown in red

Edge weights shown in blue

Node weights shown in black
Coarsening using a maximal matching (details)

1) Construct a maximal matching $E_m$ of $G(N,E)$ for all edges $e=(j,k)$ in $E_m$
   - Put node $n(e)$ in $N_c$
   - $W(n(e)) = W(j) + W(k)$

2) collapse matched nodes into a single one
   - for all nodes $n$ in $N$ not incident on an edge in $E_m$
     - Put $n$ in $N_c$
     - ... do not change $W(n)$

3) add unmatched nodes
   - ... Now each node $r$ in $N$ is “inside” a unique node $n(r)$ in $N_c$

4) Connect two nodes in $N_c$ if nodes inside them are connected in $E$
   - for all edges $e=(j,k)$ in $E_m$
     - for each other edge $e'=(j,r)$ or $(k,r)$ in $E$
       - Put edge $ee = (n(e),n(r))$ in $E_c$
       - $W(ee) = W(e')$

If there are multiple edges connecting two nodes in $N_c$, collapse them, adding edge weights
Expanding a partition of $G_c$ to a partition of $G$

Converting a coarse partition to a fine partition

Partition shown in green
Multilevel Spectral Bisection

- **Coarsen** graph and **expand** partition using **maximal independent sets**
- **Improve** partition using **Rayleigh Quotient Iteration**
Maximal Independent Sets

• *Definition:* An independent set of a graph $G(N,E)$ is a subset $N_i$ of $N$ such that no two nodes in $N_i$ are connected by an edge.

• *Definition:* A maximal independent set of a graph $G(N,E)$ is an independent set $N_i$ to which no more nodes can be added and remain an independent set.

• A simple greedy algorithm computes a maximal independent set:

  let $N_i$ be empty
  for $k = 1$ to $|N|$  ... visit the nodes in any order
    if node $k$ is not adjacent to any node already in $N_i$
      add $k$ to $N_i$
    endif
  endfor

Maximal Independent Subset $N_i$ of $N$

- $\bullet$ - nodes of $N$
- $\bigstar$ - nodes of $N_i$
Example of Coarsening

Computing $G_c$ from $G$

- $D_1$, $D_2$, $D_3$, $D_4$, $D_5$ - domains
- $\bullet$ and $\blacklozenge$ - nodes of $N$
- $\blacklozenge$ - nodes of $N_i$
- $\blacklozenge$ - nodes of $N_c$
- --- - edges in $E$
- --- - edges in $E_c$
- - encloses domain $D_k = \text{node of } N_c$
Coarsening using Maximal Independent Sets (details)

... Build “domains” $D(k)$ around each node $k$ in $N_i$ to get nodes in $N_c$
... Add an edge to $E_c$ whenever it would connect two such domains
$E_c = \emptyset$
for all nodes $k$ in $N_i$

$D(k) = (\{k\}, \emptyset)$
... first set contains nodes in $D(k)$, second set contains edges in $D(k)$
unmark all edges in $E$
repeat
choose an unmarked edge $e = (k,j)$ from $E$
if exactly one of $k$ and $j$ (say $k$) is in some $D(m)$
mark $e$
add $j$ and $e$ to $D(m)$
else if $k$ and $j$ are in two different $D(m)$’s (say $D(mk)$ and $D(mj)$)
mark $e$
add edge $(mk, mj)$ to $E_c$
else if both $k$ and $j$ are in the same $D(m)$
mark $e$
add $e$ to $D(m)$
else
leave $e$ unmarked
endif
until no unmarked edges
Expanding a partition of $G_c$ to a partition of $G$

- Need to convert an eigenvector $v_c$ of $L(G_c)$ to an approximate eigenvector $v$ of $L(G)$
- Use interpolation:

  For each node $j$ in $N$
  if $j$ is also a node in $N_c$, then
    $v(j) = v_c(j)$  \(\text{... use same eigenvector component}\)
  else
    $v(j) = \text{average of } v_c(k) \text{ for all neighbors } k \text{ of } j \text{ in } N_c$
  endif
endif
Example: 1D mesh of 9 nodes
Improve eigenvector: Rayleigh Quotient Iteration

\( j = 0 \)

pick starting vector \( v(0) \) \( \ldots \) from expanding \( v_C \)

repeat

\( j = j + 1 \)

\( r(j) = v^T(j-1) * L(G) * v(j-1) \)

\( \ldots \) \( r(j) = \text{Rayleigh Quotient of } v(j-1) \)

\( \ldots \) \( = \text{good approximate eigenvalue} \)

\( v(j) = (L(G) - r(j)*I)^{-1} * v(j-1) \)

\( \ldots \) expensive to do exactly, so solve approximately

\( \ldots \) using an iteration called SYMMLQ,

\( \ldots \) which uses matrix-vector multiply (no surprise)

\( v(j) = v(j) / \| v(j) \| \)

\( \ldots \) normalize \( v(j) \)

until \( v(j) \) converges

\( \ldots \) Convergence is very fast: cubic
Example of convergence for 1D mesh
Outline of Graph Partitioning Lectures

• Review definition of Graph Partitioning problem
• Overview of heuristics
• Partitioning with Nodal Coordinates
  • Ex: In finite element models, node at point in (x,y) or (x,y,z) space
• Partitioning without Nodal Coordinates
  • Ex: In model of WWW, nodes are web pages
• Multilevel Acceleration
  • BIG IDEA, appears often in scientific computing
• Comparison of Methods and Applications
• Beyond Graph Partitioning: Hypergraphs
Available Implementations

• Multilevel Kernighan/Lin
  • METIS (www.cs.umn.edu/~metis)
  • ParMETIS - parallel version

• Multilevel Spectral Bisection
  • Chaco (www.cs.sandia.gov/CRF/papers_chaco.html)

• Hybrids possible
  • Ex: Using Kernighan/Lin to improve a partition from spectral bisection

• Recent package, collection of techniques
  • Zoltan (www.cs.sandia.gov/Zoltan)

• See www.cs.sandia.gov/~bahendr/partitioning.html
Comparison of methods

• Compare only methods that use edges, not nodal coordinates
  • CS267 webpage and KK95a (see below) have other comparisons

• Metrics
  • Speed of partitioning
  • Number of edge cuts
  • Other application dependent metrics

• Summary
  • No one method best
  • Multi-level Kernighan/Lin fastest by far, comparable to Spectral in the number of edge cuts
    • www-users.cs.umn.edu/~karypis/metis/publications/main.html
    • see publications KK95a and KK95b
  • Spectral give much better cuts for some applications
    • Ex: image segmentation
    • See “Normalized Cuts and Image Segmentation” by J. Malik, J. Shi
### Number of edges cut for a 64-way partition

For Multilevel Kernighan/Lin, as implemented in METIS (see KK95a)

<table>
<thead>
<tr>
<th>Graph</th>
<th># of Nodes</th>
<th># of Edges</th>
<th># Edges cut for 64-way partition</th>
<th>Expected # cuts for 2D mesh</th>
<th>Expected # cuts for 3D mesh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td>144649</td>
<td>1074393</td>
<td>88806</td>
<td>6427</td>
<td>31805</td>
<td>3D FE Mesh</td>
</tr>
<tr>
<td>4ELT</td>
<td>15606</td>
<td>45878</td>
<td>2965</td>
<td>2111</td>
<td>7208</td>
<td>2D FE Mesh</td>
</tr>
<tr>
<td>ADD32</td>
<td>4960</td>
<td>9462</td>
<td>675</td>
<td>1190</td>
<td>3357</td>
<td>32 bit adder</td>
</tr>
<tr>
<td>AUTO</td>
<td>448695</td>
<td>3314611</td>
<td>194436</td>
<td>11320</td>
<td>67647</td>
<td>3D FE Mesh</td>
</tr>
<tr>
<td>BBMAT</td>
<td>38744</td>
<td>993481</td>
<td>55753</td>
<td>3326</td>
<td>13215</td>
<td>2D Stiffness M.Lin. Prog.</td>
</tr>
<tr>
<td>FINAN512</td>
<td>74752</td>
<td>261120</td>
<td>11388</td>
<td>4620</td>
<td>20481</td>
<td></td>
</tr>
<tr>
<td>LHR10</td>
<td>10672</td>
<td>209093</td>
<td>58784</td>
<td>1746</td>
<td>5595</td>
<td>Chem. Eng.</td>
</tr>
<tr>
<td>MAP1</td>
<td>267241</td>
<td>334931</td>
<td>1388</td>
<td>8736</td>
<td>47887</td>
<td>Highway Net.</td>
</tr>
<tr>
<td>MEMPLUS</td>
<td>17758</td>
<td>54196</td>
<td>17894</td>
<td>2252</td>
<td>7856</td>
<td>Memory circuit</td>
</tr>
<tr>
<td>SHYY161</td>
<td>76480</td>
<td>152002</td>
<td>4365</td>
<td>4674</td>
<td>20796</td>
<td>Navier-Stokes</td>
</tr>
<tr>
<td>TORSO</td>
<td>201142</td>
<td>1479989</td>
<td>117997</td>
<td>7579</td>
<td>39623</td>
<td>3D FE Mesh</td>
</tr>
</tbody>
</table>

**Expected # cuts for 64-way partition of 2D mesh of n nodes**

\[ n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2} \sim 17 \times n^{1/2} \]

**Expected # cuts for 64-way partition of 3D mesh of n nodes**

\[ n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3} \sim 11.5 \times n^{2/3} \]
## Speed of 256-way partitioning (from KK95a)

### Partitioning time in seconds

<table>
<thead>
<tr>
<th>Graph</th>
<th># of Nodes</th>
<th># of Edges</th>
<th>Multilevel Spectral Bisection</th>
<th>Multilevel Kernighan/Lin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td>144649</td>
<td>1074393</td>
<td>607.3</td>
<td>48.1</td>
<td>3D FE Mesh</td>
</tr>
<tr>
<td>4ELT</td>
<td>15606</td>
<td>45878</td>
<td>25.0</td>
<td>3.1</td>
<td>2D FE Mesh</td>
</tr>
<tr>
<td>ADD32</td>
<td>4960</td>
<td>9462</td>
<td>18.7</td>
<td>1.6</td>
<td>32 bit adder</td>
</tr>
<tr>
<td>AUTO</td>
<td>448695</td>
<td>3314611</td>
<td>2214.2</td>
<td>179.2</td>
<td>3D FE Mesh</td>
</tr>
<tr>
<td>BBMAT</td>
<td>38744</td>
<td>993481</td>
<td>474.2</td>
<td>25.5</td>
<td>2D Stiffness M. Lin. Prog.</td>
</tr>
<tr>
<td>FINAN512</td>
<td>74752</td>
<td>261120</td>
<td>311.0</td>
<td>18.0</td>
<td>2D Stiffness M. Lin. Prog.</td>
</tr>
<tr>
<td>LHR10</td>
<td>10672</td>
<td>209093</td>
<td>142.6</td>
<td>8.1</td>
<td>Chem. Eng.</td>
</tr>
<tr>
<td>MAP1</td>
<td>267241</td>
<td>334931</td>
<td>850.2</td>
<td>44.8</td>
<td>Highway Net.</td>
</tr>
<tr>
<td>MEMPLUS</td>
<td>17758</td>
<td>54196</td>
<td>117.9</td>
<td>4.3</td>
<td>Memory circuit</td>
</tr>
<tr>
<td>SHYY161</td>
<td>76480</td>
<td>152002</td>
<td>130.0</td>
<td>10.1</td>
<td>Navier-Stokes</td>
</tr>
<tr>
<td>TORSO</td>
<td>201142</td>
<td>1479989</td>
<td>1053.4</td>
<td>63.9</td>
<td>3D FE Mesh</td>
</tr>
</tbody>
</table>

Kernighan/Lin much faster than Spectral Bisection!
Outline of Graph Partitioning Lectures

• Review definition of Graph Partitioning problem
• Overview of heuristics
• Partitioning with Nodal Coordinates
  • Ex: In finite element models, node at point in (x,y) or (x,y,z) space
• Partitioning without Nodal Coordinates
  • Ex: In model of WWW, nodes are web pages
• Multilevel Acceleration
  • BIG IDEA, appears often in scientific computing
• Comparison of Methods and Applications
• Beyond Graph Partitioning: Hypergraphs
Beyond simple graph partitioning:
Representing a sparse matrix as a hypergraph

\[
\begin{bmatrix}
\times & 0 & \times & 0 \\
0 & \times & \times & 0 \\
0 & \times & \times & 0 \\
0 & \times & 0 & \times
\end{bmatrix}
\]
Using a graph to partition, versus a hypergraph

But graph cut is 4!

⇒ Cut size of graph partition is not an accurate count of communication volume

Source vector entries corresponding to c2 and c3 are needed by both partitions – so total volume of communication is 2:

\[
\begin{bmatrix}
\times & 0 & \times & 0 \\
0 & \times & \times & 0 \\
0 & \times & \times & 0 \\
0 & \times & 0 & \times \\
\end{bmatrix}
\]
Using a graph to partition, versus a hypergraph

Source vector entries corresponding to \( c2 \) and \( c3 \) are needed by both partitions – so total volume of communication is 2

\[
\begin{bmatrix}
\times & 0 & \times & 0 \\
0 & \times & \times & 0 \\
0 & \times & \times & 0 \\
0 & \times & 0 & \times \\
\end{bmatrix}
\]

But graph cut is 3!

⇒ Cut size of graph partition may not accurately count communication volume
Two Different 2D Mesh Partitioning Strategies

Graph:
Cartesian Partitioning

Communication Volume per proc (SpMV) = nodes needed by 1 other proc * 1 + nodes needed by 2 other procs * 2 = 14*1 + 1*2 = 16

Total Communication Volume (SpMV) = nprocs * (comm per proc) = 4 * 16 = 64

Upper left/lower right: (10 * 1) + (1 * 2) = 12

Upper right/lower left: (15 * 1) + (1 * 2) = 17

Total Communication Volume (SpMV) = 2 * 12 + 2 * 17 = 58

Hypergraph:
MeshPart Algorithm [Ucar, Catalyurek, 2010]

Total SpMV communication volume = 58

Total SpMV communication volume = 64
Generalization of the MeshPart Algorithm

In general, for a $P \times Q$ partitioning of an $M \times N$ mesh, \( \text{MeshPart: } \text{vol}(M,N,P,Q) = n(3PQ - (P+Q) - 1) + (P-1)(3Q-5) + (Q-1)(3P-5) \)

where \( n = \frac{M}{P} = \frac{N}{Q} \)

Cartesian: \( \text{vol}(M,N,P,Q) = 2N(P-1) + 2M(Q-1) \)

Suspected to be optimal for certain size matrices and proc. grids

For $N \times N$ mesh on $P \times P$ processor grid:
Usual Cartesian partitioning costs \( \sim 4NP \) words moved
MeshPart costs \( \sim 3NP \) words moved, 25% savings

Source: Ucar and Catalyruk, 2010
Experimental Results: Hypergraph vs. Graph Partitioning

64x64 Mesh (5-pt stencil), 16 processors

Graph Partitioning (Metis)
Total Comm. Vol = 777
Max Vol per Proc = 69

Hypergraph Partitioning (PaToH)
Total Comm. Vol = 719
Max Vol per Proc = 59

\(~8\% \text{ reduction in total communication volume using hypergraph partitioning (PaToH)} \)
\(\text{versus graph partitioning (METIS)} \)
Further Benefits of Hypergraph Model: Nonsymmetric Matrices

• Graph model of matrix has edge \((i,j)\) if either \(A(i,j)\) or \(A(j,i)\) nonzero
• Same graph for \(A\) as \(|A| + |A^T|\)
• Ok for symmetric matrices, what about nonsymmetric?
  • Try \(A\) upper triangular

Graph Partitioning (Metis)
Total Communication Volume = 254
Load imbalance ratio = 6%

Hypergraph Partitioning (PaToH)
Total Communication Volume = 181
Load imbalance ratio = 0.1%
Summary: Graphs versus Hypergraphs

- Pros and cons
  - When matrix is non-symmetric, the graph partitioning model (using $A+A^T$) loses information, resulting in suboptimal partitioning in terms of communication and load balance.
  - Even when matrix is symmetric, graph cut size is not an accurate measurement of communication volume
  - Hypergraph partitioning model solves both these problems
  - However, hypergraph partitioning (PaToH) can be much more expensive than graph partitioning (METIS)

- Hypergraph partitioners: PaToH, HMETIS, ZOLTAN

- For more see Bruce Hendrickson’s web page
  - “Load Balancing Fictions, Falsehoods and Fallacies”
Extra Slides
Beyond Simple Graph Partitioning

• Undirected graphs model symmetric matrices, not unsymmetric ones

• More general graph models include:
  • Hypergraph: nodes are computation, edges are communication, but connected to a set ($\geq 2$) of nodes
    • HMETIS, PATOH, ZOLTAN packages
  • Bipartite model: use bipartite graph for directed graph
  • Multi-object, Multi-Constraint model: use when single structure may involve multiple computations with differing costs

• For more see Bruce Hendrickson’s web page
  • www.cs.sandia.gov/~bahendr/partitioning.html
  • “Load Balancing Myths, Fictions & Legends”
Graph vs. Hypergraph Partitioning

Consider a 2-way partition of a 2D mesh:

The cost of communicating vertex A is 1 – we can send the value in one message to the other processor.

According to the graph model, however, the vertex A contributes 2 to the total communication volume, since 2 edges are cut.

The hypergraph model accurately represents the cost of communicating A (one hyperedge cut, so communication volume of 1).

Result: Unlike graph partitioning model, the hypergraph partitioning model gives exact communication volume (minimizing cut = minimizing communication).

Therefore, we expect that hypergraph partitioning approach can do a better job at minimizing total communication. Let’s look at a simple example…
Further Benefits of Hypergraph Model: Nonsymmetric Matrices

- Graph model of matrix has edge \((i,j)\) if either \(A(i,j)\) or \(A(j,i)\) nonzero
- Same graph for \(A\) as \(|A| + |A^T|\)
- Ok for symmetric matrices, what about nonsymmetric?

Illustrative Bad Example: triangular matrix

Whereas the hypergraph model can capture nonsymmetry, the graph partitioning model deals with nonsymmetry by partitioning the graph of \(A+A^T\) (which in this case is a dense matrix).

This results in a suboptimal partition in terms of both communication and load balancing. In this case,

Total Communication Volume = 60 (optimal is ~12 in this case, subject to load balancing)
Proc1: 76 nonzeros, Proc 2: 60 nonzeros (~26% imbalance ratio)
Experimental Results: Illustration of Triangular Example

Graph Partitioning (Metis)
- Total Communication Volume = 254
- Imbalance ratio = 6%

Hypergraph Partitioning (PaToH)
- Total Communication Volume = 181
- Imbalance ratio = 0.1%

Conclusions from this section:
- When matrix is non-symmetric, the graph partitioning model (using $A+A^T$) loses information, resulting in suboptimal partitioning in terms of communication and load balance.
- Even when matrix is symmetric, graph cut size is not an accurate measurement
Coordinate-Free Partitioning: Summary

• Several techniques for partitioning without coordinates
  • Breadth-First Search – simple, but not great partition
  • Kernighan-Lin – good corrector given reasonable partition
  • Spectral Method – good partitions, but slow

• Multilevel methods
  • Used to speed up problems that are too large/slow
  • Coarsen, partition, expand, improve
  • Can be used with K-L and Spectral methods and others

• Speed/quality
  • For load balancing of grids, multi-level K-L probably best
  • For other partitioning problems (vision, clustering, etc.) spectral may be better
  • Good software available
Is Graph Partitioning a Solved Problem?

- Myths of partitioning due to Bruce Hendrickson
  1. Edge cut = communication cost
  2. Simple graphs are sufficient
  3. Edge cut is the right metric
  4. Existing tools solve the problem
  5. Key is finding the right partition
  6. Graph partitioning is a solved problem

- Slides and myths based on Bruce Hendrickson’s:
  “Load Balancing Myths, Fictions & Legends”
Myth 1: Edge Cut = Communication Cost

• Myth 1: The edge-cut deceit
  edge-cut = communication cost

• Not quite true:
  • #vertices on boundary is actual communication volume
    • Do not communicate same node value twice
  • Cost of communication depends on # of messages too (α term)
  • Congestion may also affect communication cost

• Why is this OK for most applications?
  • Mesh-based problems match the model: cost is ~ edge cuts
  • Other problems (data mining, etc.) do not
Myth 2: Simple Graphs are Sufficient

- Graphs often used to encode data dependencies
  - Do X before doing Y

- Graph partitioning determines data partitioning
  - Assumes graph nodes can be evaluated in parallel
  - Communication on edges can also be done in parallel
  - Only dependence is between sweeps over the graph

- More general graph models include:
  - Hypergraph: nodes are computation, edges are communication, but connected to a set ($\geq 2$) of nodes
  - Bipartite model: use bipartite graph for directed graph
  - Multi-object, Multi-Constraint model: use when single structure may involve multiple computations with differing costs
Myth 3: Partition Quality is Paramount

- When structure are changing dynamically during a simulation, need to partition dynamically
  - Speed may be more important than quality
  - Partitioner must run fast in parallel
  - Partition should be incremental
    - Change minimally relative to prior one
  - Must not use too much memory

- Example from Touheed, Selwood, Jimack and Bersins
  - 1 M elements with adaptive refinement on SGI Origin
  - Timing data for different partitioning algorithms:
    - Repartition time from 3.0 to 15.2 secs
    - Migration time: 17.8 to 37.8 secs
    - Solve time: 2.54 to 3.11 secs
References

• Details of all proofs on Jim Demmel’s 267 web page
• www.cs.berkeley.edu/~ruhe/lantplht/lantplht.html
• www.netlib.org/laso
Summary

• Partitioning with nodal coordinates:
  • Inertial method
  • Projection onto a sphere
  • Algorithms are efficient
  • Rely on graphs having nodes connected (mostly) to “nearest neighbors” in space

• Partitioning without nodal coordinates:
  • Breadth-First Search – simple, but not great partition
  • Kernighan-Lin – good corrector given reasonable partition
  • Spectral Method – good partitions, but slow

• Today:
  • Spectral methods revisited
  • Multilevel methods
Another Example

• Definition: The Laplacian matrix \( L(G) \) of a graph \( G(N,E) \) is an \( |N| \times |N| \) symmetric matrix, with one row and column for each node. It is defined by
  
  - \( L(G) (i,i) = \) degree of node \( i \) (number of incident edges)
  - \( L(G) (i,j) = -1 \) if \( i \neq j \) and there is an edge \((i,j)\)
  - \( L(G) (i,j) = 0 \) otherwise

\[
L(G) = \begin{pmatrix}
2 & -1 & -1 & 0 & 0 \\
-1 & 2 & -1 & 0 & 0 \\
-1 & -1 & 4 & -1 & -1 \\
0 & 0 & -1 & 2 & -1 \\
0 & 0 & -1 & -1 & 2 \\
\end{pmatrix}
\]