A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs
George Karypis and Vipin Kumar

Jared Di Carlo

April 23, 2019
Graph Partitioning

- Divide vertices into $p$ parts of roughly equal size (or sum of vertex weights)
- Minimize edges across parts
- NP-Complete

(a) A poor edge-cut partitioning. Vertices are assigned to partitions at random, thus, there are many inter-partition links.

(b) A good edge-cut of the same graph, where vertices that are highly connected are assigned to the same partition.
Applications of $k$-way Graph Partitioning

- Scheduling work on $k$ processors
  - Edges represent sharing of data between tasks
  - Sparse matrix vector product
- Sparse matrix factorization
  - Reorder matrix to make the factorization sparse too
- Power Law Graphs?
Algorithms for Graph Partitioning

- Spectral Partitioning: Slow
Algorithms for Graph Partitioning

- Spectral Partitioning: Slow
- Geometric Partitioning: Requires vertices to have coordinates
Algorithms for Graph Partitioning

- Spectral Partitioning: Slow
- Geometric Partitioning: Requires vertices to have coordinates
- Multilevel: Fast, but low quality
Multilevel Partitioning

- Three phases
  - Coarsening (collapse vertices)
Multilevel Partitioning

- Three phases
  - Coarsening (collapse vertices)
  - Partition
Multilevel Partitioning

- Three phases
  - Coarsening (collapse vertices)
  - Partition
  - Uncoarsening (possibly refining the partitions)
Coarsening

- Each coarsening iteration collapses a *maximal matching*
  - Matching: set of edges which hit each vertex at most once
  - Coarsen: Collapse edges in matching
  - Maximal Matching: Every non-matched edge touches a vertex which has a matched edge
Coarsening Strategies

Random

- Visit random vertices, add random edges. $O(|E|)$
- Terminate when there are no more vertices that can have edges added
- Works well on “engineering” graphs - meshes
Coarsening Strategies

Heavy Edges
- Greedy
- Visit random vertices, pick heaviest edge to remove: $O(|E|)$
- Can reduce the edge-cut because heavy edges are removed
- Does well for VLSI graphs

Heavy Clique
- Collapse nodes which are unlikely to be split by the bisection
- Randomly visit vertices, pick edges leading to highest edge-density vertex
- Edge Density: $2 \frac{E_U}{U(U-1)} = 2 \frac{CE(u) + CE(v) + EW(u,v)}{(VW(u) + VW(v))(VW(u) + VW(v) - 1)}$
Partitioning Phase

- This step is fast - coarse graph should have around 100 vertices
- Spectral Bisection, KL, GGP
Spectral Bisection

- Consider $Q = D - A$ where $D$ is diagonal degree matrix, $A$ is adjacency matrix.
- Eigenvectors: $Qx = \lambda_i x$
- Let $x$ represent a partition with $x_i \in \{-1, 1\}$
- The product $Qx$ is proportional to the number of cut edges
KL Algorithm

- Iterative - greedily swaps vertices to make things better
- Can get stuck in local minima
- Run the algorithm a few times (5 - 10)
- Can be improved by prioritizing vertices with a large effect
Graph Growing Partition (GGP)

- BFS from a random vertex until half the vertices are added
- Sensitive to initial choice
Uncoarsening

- As vertices are expanded, move ones on the edge to improve edge-cut
- KL, KL(1), KL Boundary
KL, KL(1)

- Same algorithm as KL previously
- Terminates very quickly, as the partition is already good
- Dominated by insertion into data structure
- KL(1) runs a single iteration, allowing simpler data structures
- Boundary KL - only consider vertices that are on the edge
- Use BKL(1) on large graphs, BKL on smaller graphs
Experiments - Graph Partition

- SGI Challenge, 200 MHz MIPS R4400, 1.2 GB RAM
- Vertices: 4960 to 448695
- Edges: 9462 to 3358036
- 2D/3D meshes, stiffness matrix, “Chemical Engineering”, Highway, Stiffness matrix, Circuits (adder, memory, sequential)
- Coarsening: Heavy Edge has the lowest edge cut, and good runtime
- Partitioning: GGP or Spectral, depending on graph
- Uncoarsening: BKL or BKL dynamic - dynamic is faster, but BKL is 2% better
Experiments - Sparse Matrix Factorization

![Table](image)

- **Parallelization** - better than MMD (greedy)
- **56x speedup on 128-CPU Cray T3D**
Comparison

**Table 9**

*Characteristics of various graph partitioning algorithms.*

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of Trials</th>
<th>Needs Coordinates</th>
<th>Quality</th>
<th>Local View</th>
<th>Global View</th>
<th>Run Time</th>
<th>Degree of Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectral Bisection</td>
<td>1</td>
<td>no</td>
<td>●●●●●</td>
<td>●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Multilevel Spectral Bisection</td>
<td>1</td>
<td>no</td>
<td>●●●●●</td>
<td>●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Multilevel Spectral Bisection-KL</td>
<td>1</td>
<td>no</td>
<td>●●●●●●</td>
<td>●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Multilevel Partitioning</td>
<td>1</td>
<td>no</td>
<td>●●●●●●</td>
<td>●●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Levelized Nested Dissection</td>
<td>1</td>
<td>no</td>
<td>●●</td>
<td>●</td>
<td>●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Kernighan-Lin</td>
<td>1</td>
<td>no</td>
<td>●●●●●○</td>
<td>●●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Kernighan-Lin</td>
<td>10</td>
<td>no</td>
<td>●●●●●●</td>
<td>●●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
<tr>
<td>Kernighan-Lin</td>
<td>50</td>
<td>no</td>
<td>●●●●●●</td>
<td>●●</td>
<td>●●●●</td>
<td>▲▲</td>
<td></td>
</tr>
</tbody>
</table>