Partitioning & Clustering Big Graphs

George Karypis
Department of Computer Science & Engineering
Twin Cities
University of Minnesota
Overview

- Overview of graph partitioning
- The multilevel paradigm
- METIS family of partitioning tools
- Multi-threaded algorithms for partitioning & clustering
- Closing remarks
Given a graph \( G=(V, E) \) we want to partition it into \( k \) parts such that:
- each part has roughly the same number of vertices
- and
- the edges that straddle partitions (edge-cut) is minimized

Applications
- Parallel & distributed computing
- Scientific computing
- VLSI physical design
- Data-mining
- Storage and placement
- ...

It is NP-hard. Heuristic algorithms are used!
Overview of the multilevel graph partitioning paradigm
Coarsening Phase

Successive coarse graphs are constructed by computing a matching of the edges, and collapsing together the vertices incident on these edges.

Total Edge-Weight: 37

Heavy-Edge Matching

Total Edge-Weight: 21
Refinement Phase

The refinement is performed by using *move-based* approaches, based on the algorithm by Fiduccia-Mattheyses (FM).

![Diagram showing two partitions with vertices and edgecut](image)

**Partition i**
- ID[v] = 4,
- ED[v] = 8,

**Partition j**
- Edgecut = 8

**Partition i**
- Edgecut = 4

Part 1: 1, 2, 3
Part 2: 1, 2, 3
Why does the multilevel partitioning paradigm work?

- The coarsening phase by hiding a large fraction of the edges, makes the partitioning problem easier.
- Performing refinement at successive finer graphs, enhances the effectiveness of refinement algorithms.
  - Multi-scale refinement

![After Coarsening]
When does the multilevel paradigm have difficulties?

- The value of the objective function in the original graph cannot be (tightly) upper bounded while operating on a coarser graph.
  - We cannot ensure improvements at a coarse graph lead to improvements in the original graph.

- Coarsening fails to make the optimization problem easier in coarser graphs.
  - It is not “in tune” with the objective.

- Coarsening fails to reduce the size of the problem (|V|+|E|).
  - Can increase the runtime/memory requirements.

- The objective function is based on global properties of the graph.
  - Can substantially increase the refinement time.
METIS, ParMETIS, & hMETIS

- Software packages for partitioning unstructured graphs and hypergraphs and computing fill reducing orderings.
  - METIS was released in 1995 (current version 5.x).
  - ParMETIS was released in 1997 (current version 4.x).
  - hMETIS was released in 1998 (current version 2.x)
- They are freely distributed and widely used.
Beyond the traditional partitioning problem (1)

- **Vertex separators**
  - Partition the graph by removing a minimum set of vertices.
  - Broad applications to:
    - matrix reordering for direct solvers
    - concurrency extraction by decoupling computations at each partition
    - overlapping clustering solutions
Beyond the traditional partitioning problem (2)

- **Constraints**
  - **Multiple balancing constraints**
    - balance load & memory requirements,
    - balance the different types of modules that are assigned to each chip in a multi-chip FPGA design,
    - balance incoming & outgoing messages,
    - balance iterative & direct solvers, etc.
  - **Connectivity constraints**
    - ensure that the graph induced by the vertices of each partition is connected.
  - **Placement constraints**
    - ensure that certain vertices are placed in different and/or the same partitions.
  - **No constraints**
    - Objective driven partitioning.
Beyond the traditional partitioning problem (3)

- Objectives
  - Communication volume
  - Subdomain connectivity
  - Redistribution overhead
  - Multiple edge-defined cost functions
  - Path-based objectives
    - timing considerations in VLSI circuits
  - Clustering objectives
    - normalized cut, ratio cut, min-max, modularity, ...
  - Various combinations of the above
Some performance numbers

**Table**: Test graph statistics.

| Graph      | |V|   | |E|   |
|------------|---|---|---|
| ljournal-2007 | 5.3M | 49.5M |
| uk-2002     | 18.5M | 261.8M |
| uk-2007     | 105.9M | 3.3B  |

**Table**: Performance on some graphs.

<table>
<thead>
<tr>
<th>Graph</th>
<th>50/%cdges/time</th>
<th>100/%cdges/time</th>
<th>200/%cdges/time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ljournal-2007</td>
<td>31.2%</td>
<td>35.9%</td>
<td>37.0%</td>
</tr>
<tr>
<td>uk-2002</td>
<td>1.1%</td>
<td>1.3%</td>
<td>1.4%</td>
</tr>
<tr>
<td>uk-2007</td>
<td>0.4%</td>
<td>0.5%</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz, 128GB
IMPROVING SINGLE NODE PERFORMANCE
Multi-threaded graph partitioning/clustering

Opportunities:
- Multi-core processors have become ubiquitous.
- Their cache-coherent shared-memory architecture makes it easier to develop parallel programs.

Challenges:
- Non-uniform access to shared memory.
- Many applications are bound by memory bandwidth.
- Limited memory per core.

<table>
<thead>
<tr>
<th>System</th>
<th>Cores / Node</th>
<th>Memory / Node</th>
<th>Memory / Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan</td>
<td>16</td>
<td>32 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td>Sequoia</td>
<td>16</td>
<td>16 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>K-Computer</td>
<td>8</td>
<td>16 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td>Mira</td>
<td>16</td>
<td>16 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>JUQUEEN</td>
<td>16</td>
<td>16 GB</td>
<td>1 GB</td>
</tr>
</tbody>
</table>

Thursday, May 23, 13
Kmetis – Serial algorithm

- Graph is stored in CSR format.
- Coarsening:
  - Matches connected pairs of vertices for contraction.
  - Vertices are matched in ascending order of degree.
  - Edges prioritized for collapsing by edge weight.
- Initial partitioning:
  - The best of several partitionings generated via recursive bisection is chosen.
- Uncoarsening:
  - Projection of the partitioning.
  - Greedy k-way refinement using only boundary vertices.
Parallel matching approaches

- **Fine-grain matching**
  - Lock the vertex and prospective vertices to match with.

- **Multi-pass matching**
  - First pass to generate match requests.
  - Second pass to grant or deny match requests.

- **Unprotected matching**
  - Allow threads to modify the matching without any locking.
  - Fix broken matches after matches are selected.
Parallel refinement approaches

- Fine-grain approach
  - Locks to and from partitions along with neighbor vertices.

- Coarse-grain approach
  - Each thread moves up to \( c \) vertices.
  - Potential partition weights are communicated.
  - Moves are discarded until a balanced partitioning would result.
  - Vertex book-keeping information and partition weights are updated.
  - Repeat until all of the priority queues are empty.

![Graph showing speedup vs. number of cores for coarse-grain and fine-grain refinement compared to ideal performance on VLSICRCT with 2x 8-core Xeon processors.](chart.png)
Thread lifetimes and data ownership

(a) 2x 8-core Xeon

(b) 8x 4-core Opteron
Overall performance – Mean speedup over kmetis

(c) 2x 8-core Xeon

(d) 8x 4-core Opteron
Memory usage

- mt-metis
- parmetis
- pt-scotch
- kmetis

Memory Usage (MB)

# of cores

Thursday, May 23, 13
Graph clustering

Goal:
- Develop a multi-threaded multi-level modularity maximization graph clustering algorithm.

Key differences/contributions
- No need to worry about balance constraints.
- Focus on coarsening heuristics that are suited for “big data” types of graphs.
  - First-choice & two-hop matching.
- The objective is not locally defined.
  - Focus on “cautiously” optimistic heuristics with frequent global updates.
- Automatically determine the right number of clusters during initial clustering.
Some results

Table: Graphs Used in Scaling Experiments

<table>
<thead>
<tr>
<th>Graph</th>
<th># Vertices</th>
<th># Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>road_usa</td>
<td>23,947,347</td>
<td>28,854,312</td>
</tr>
<tr>
<td>soc-LiveJournal1</td>
<td>4,846,609</td>
<td>42,851,237</td>
</tr>
<tr>
<td>europe.osm</td>
<td>50,912,018</td>
<td>54,054,660</td>
</tr>
<tr>
<td>uk-2002</td>
<td>18,520,486</td>
<td>261,787,258</td>
</tr>
</tbody>
</table>

Table: Clustering Times (s)

<table>
<thead>
<tr>
<th>Graph</th>
<th>1 Thread</th>
<th>8 Threads</th>
<th>16 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>road_usa</td>
<td>20.66</td>
<td>5.69</td>
<td>4.32</td>
</tr>
<tr>
<td>soc-LiveJournal1</td>
<td>39.77</td>
<td>7.47</td>
<td>5.59</td>
</tr>
<tr>
<td>europe.osm</td>
<td>42.89</td>
<td>12.52</td>
<td>10.33</td>
</tr>
<tr>
<td>uk-2002</td>
<td>202.43</td>
<td>44.76</td>
<td>35.87</td>
</tr>
</tbody>
</table>
Scaling results

- road_usa
- soc-LiveJournal1
- europe.osm
- uk-2002
Final words

- Lessons from mt-metis/mt-cluster
  - The shared memory opportunity
    - Explicit shared memory programing leads to reduced overheads.
    - Reduced memory foot-print.
  - Data locality is still important.
  - Synchronization should be avoided.
  - Lessons learned from MP algorithms are still valid!

- Thank you.
  - http://www.cs.umn.edu/~metis