Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering

George Karypis and Vipin Kumar

Brian Shi

CSci 8314

03/09/2017
Outline

Introduction
  Graph Partitioning Problem
  Multilevel Graph Partitioning Algorithm

Parallelizing The Multilevel Graph Partitioning Algorithm
  Initial Partitioning Phase
  Uncoarsening Phase
  Parallel Refinement Algorithm

Parallel Multilevel Sparse Matrix Ordering Algorithm

Performance, Scaling Analysis, and Results
The graph partitioning problem is \( NP \)-complete.

Definition: Let \( G = (V, E) \) be a graph with \( |V| = n \). A \( p \)-way partition of \( G \) consists of \( V_1, \ldots, V_p \subset V \) such that \( V \) is a disjoint union of \( V_1, \ldots, V_p \) with \( |V_i| \approx n/p \) and the edge cuts are minimized.
Multilevel Graph Partitioning Algorithm

- Consists of three main phases.
  1. Coarsening Phase: A matching of $G$ is constructed and edges are contracted.
  2. Initial Partitioning Phase: A partition of a smaller coarsened graph is performed.
  3. Uncoarsening Phase: The partitioned graph is uncontracted and map back to the original graphs.
Coarsening Phase

- Sequence of smaller graphs \( \{G_i\} \) are constructed by finding maximal matchings starting from \( G_0 = G \).

Graph Matching

- Let \( G = (V, E) \) be an undirected graph, then a matching of \( G \) is a subset of edges, \( M \subset E \), such that no vertex in \( V \) is incident to more than one edge in \( M \).
- Vertices in matched edges contracted
- Random Matching
- Heavy Edge Matching
Partitioning and Uncoarsening

1. Initial Partitioning Phase: A partition of a smaller coarsened graph is performed.
2. Serial algorithms Greedy Graph Growing Partition (GGGP).
3. Partitioned graph is projected back to the original graph.
4. Refinement algorithms used to minimize edge-cuts and load balancing.
Parallelizing The Multilevel Graph Partitioning Algorithm

- Assumption: \( p \)-way partitioning with \( p \) processors
- Parallelization exploited in the recursive nature of the bisection algorithm
- \( p \) processors perform one bisection. Then \( p/2 \) processors perform bisections of each half.
- Goal: Parallel algorithm for graph bisection.
Coarsening Phase

Main Idea

▶ Assume $p = 2^{2r}$ processors arranged in a 2-D array, and

$$(V_0, E_0) = (V, E)$$

▶ Distribute the vertices into $\sqrt{p}$ subsets:

$$V_0^0, V_0^1, ..., V_0^{\sqrt{p}-1}$$

▶ Processor $P_{ij}$ will contain subset of edges in $E_0$ with incident vertices in $V_0^i$ and $V_0^j$
Coarsening Phase

- The edge matchings, $M_0^i$, will be done along the diagonal processors.
- After completion each $P_{ii}$ does two broadcasts along its row and columns to distribute $M_0^i$.
- $E_1 = M_0 = \bigcup_i M_0^i$
- Each $P_{ij}$ contains edges with incident vertices in $V_1^i$ and $V_1^j$
- Once $G_k$ is coarse enough we reduce number of working processors by folding.
Initial Partitioning Phase

- Coarsest graph will be contained in a single processor.
- Can be done sequentially using GGGP.
- Several runs of GGGP are run using different random starting vertices.
- We can copy the coarsest graph to multiple processors and run these trials concurrently.
- We keep the partition giving the smallest edge-cut.
Uncoarsening Phase

» Project coarse graphs back to original graphs.
» Refinements are made during each step of the projection
  » Serial algorithms (Kernighan-Lin variants) used when coarse graph resides in one processor.
» The processor folding is reversed.
Parallel Refinement Algorithm

- Each $P_{ij}$ computes local gain $lg_v$ for each $v \in V^i_0$.
- Total gain of $v$, $g_v$, is computed via a sum reduction along the columns.
- Each $P_{ii}$ selects vertices, $U_i \subset V^i_0$, with positive gain.
- Broadcast $U_i$ row and column-wise.
- Recompute the local gains and repeat.
- Balancing partitions
  - Start vertex swaps from heavier parts of partition
  - Use a load balancing iteration if there is more than 2% imbalance.
Parallel Multilevel Sparse Matrix Ordering Algorithm

- Assume a bisection is already constructed. \( A, B \) be the boundary vertices.
- Need to construct vertex separator.
Parallel Vertex Cover Algorithm

- Processors are arranged in a 2-D array.

\[ A_i = A \cap V_0^i \quad \text{and} \quad B_i = B \cap V_0^i \]

- Each \( P_{ij} \) stores \( A_i \) and \( B_j \) and computes minimal cover of edges between \( A_i \), \( B_j \) locally. Denote

\[ A_{ij}^c \subset A \quad \text{and} \quad B_{ij}^c \subset B \quad \text{and} \quad A_{ij}^c \cup B_{ij}^c \] the minimal cover

- Union of \( A_{ij}^c \), \( B_{ij}^c \) across all processors forms a cover.
Parallel Vertex Cover Algorithm

Constructing Minimal Cover

- Broadcast $B^i_c = \bigcup_j B^i_{cj}$ to processors in same column
- Each $P_{ij}$ removes vertices from $A^ij_c$ whose edges are covered in $B^i_c$
  \[ A^ij_c' \subset A^ij_c \]
- Broadcast $A^{i'}_c = \bigcup_j A^{i'}_{cj}$ to processors in same row
- Each $P_{ij}$ removes vertices from $B^j_c$ whose union is $B^{j'}_c$
- Then a minimal cover is got via

$$ S = \left( \bigcup_{i=0}^{\sqrt{p}-1} A_{c'}^i \right) \bigcup \left( \bigcup_{j=0}^{\sqrt{p}-1} B_{c'}^j \right) $$
Scaling Analysis

- Time to broadcast is
  \[ T_{\text{broadcast}} = O\left( \frac{n}{\sqrt{p}} \right) \]
  
  \[ n = |V_0| \text{ and } m = |E_0| \]

- Time for bisection is
  \[ T_{\text{bisection}} = O\left( \frac{m}{p} \right) + O\left( \frac{n}{\sqrt{p}} \right) \]

- Overall runtime of the \( p \)-way partitioning algorithm is
  \[ T_{\text{partition}} = \left( O\left( \frac{m}{p} \right) + O\left( \frac{n}{\sqrt{p}} \right) \right) \log(p) \]
## Test Cases

### Various Matrices Used in Evaluating the Multilevel Graph Partitioning and Sparse Matrix Ordering Algorithm

<table>
<thead>
<tr>
<th>Matrix name</th>
<th>Number of vertices</th>
<th>Number of edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4ELT</td>
<td>15606</td>
<td>45878</td>
<td>2D finite-element mesh</td>
</tr>
<tr>
<td>BCSSTK31</td>
<td>35588</td>
<td>572914</td>
<td>3D stiffness matrix</td>
</tr>
<tr>
<td>BCSSTK32</td>
<td>44609</td>
<td>985046</td>
<td>3D stiffness matrix</td>
</tr>
<tr>
<td>BRACK2</td>
<td>62631</td>
<td>366559</td>
<td>3D finite-element mesh</td>
</tr>
<tr>
<td>CANT</td>
<td>54195</td>
<td>1960797</td>
<td>3D stiffness matrix</td>
</tr>
<tr>
<td>COPTER2</td>
<td>55476</td>
<td>352238</td>
<td>3D finite-element mesh</td>
</tr>
<tr>
<td>CYLINDER93</td>
<td>45594</td>
<td>1786726</td>
<td>3D stiffness matrix</td>
</tr>
<tr>
<td>ROTOR</td>
<td>99617</td>
<td>662431</td>
<td>3D finite-element mesh</td>
</tr>
<tr>
<td>SHELL93</td>
<td>181200</td>
<td>2313765</td>
<td>3D stiffness matrix</td>
</tr>
<tr>
<td>WAVE</td>
<td>156317</td>
<td>1059331</td>
<td>3D finite-element mesh</td>
</tr>
</tbody>
</table>
## Results

The Performance of the Parallel Multilevel Graph Partitioning Algorithm

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$p = 16$</th>
<th></th>
<th>$p = 32$</th>
<th></th>
<th>$p = 64$</th>
<th></th>
<th>$p = 128$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_p$</td>
<td>$EC_{16}$</td>
<td>$S$</td>
<td>$T_p$</td>
<td>$EC_{32}$</td>
<td>$S$</td>
<td>$T_p$</td>
<td>$EC_{64}$</td>
</tr>
<tr>
<td>4ELT</td>
<td>0.48</td>
<td>1443</td>
<td>6.0</td>
<td>0.48</td>
<td>1995</td>
<td>7.0</td>
<td>0.48</td>
<td>3210</td>
</tr>
<tr>
<td>BCSSTK31</td>
<td>1.28</td>
<td>27215</td>
<td>10.7</td>
<td>1.02</td>
<td>43832</td>
<td>17.0</td>
<td>0.87</td>
<td>67134</td>
</tr>
<tr>
<td>BCSSTK32</td>
<td>1.69</td>
<td>43987</td>
<td>12.0</td>
<td>1.33</td>
<td>71378</td>
<td>19.2</td>
<td>1.05</td>
<td>104532</td>
</tr>
<tr>
<td>BRACK2</td>
<td>2.14</td>
<td>14987</td>
<td>8.6</td>
<td>1.83</td>
<td>21545</td>
<td>12.2</td>
<td>1.56</td>
<td>32134</td>
</tr>
<tr>
<td>CANT</td>
<td>3.20</td>
<td>199567</td>
<td>13.4</td>
<td>2.29</td>
<td>322498</td>
<td>23.7</td>
<td>1.71</td>
<td>441459</td>
</tr>
<tr>
<td>COPTER2</td>
<td>2.05</td>
<td>22498</td>
<td>7.4</td>
<td>1.78</td>
<td>32765</td>
<td>11.1</td>
<td>1.59</td>
<td>45230</td>
</tr>
<tr>
<td>CYLINDER93</td>
<td>2.35</td>
<td>131534</td>
<td>14.3</td>
<td>1.71</td>
<td>198675</td>
<td>24.5</td>
<td>1.34</td>
<td>288340</td>
</tr>
<tr>
<td>Rotor</td>
<td>3.16</td>
<td>26532</td>
<td>11.0</td>
<td>2.89</td>
<td>39785</td>
<td>14.4</td>
<td>2.40</td>
<td>57540</td>
</tr>
<tr>
<td>SHELL93</td>
<td>5.80</td>
<td>54765</td>
<td>13.9</td>
<td>4.40</td>
<td>86320</td>
<td>22.5</td>
<td>3.25</td>
<td>130856</td>
</tr>
<tr>
<td>Wave</td>
<td>5.10</td>
<td>57543</td>
<td>10.3</td>
<td>4.70</td>
<td>76785</td>
<td>13.3</td>
<td>3.73</td>
<td>101210</td>
</tr>
</tbody>
</table>

*Note.* For each matrix, the performance is shown for 16, 32, 64, and 128 processors. $T_p$ is the parallel run time for a $p$-way partition on $p$ processors, $EC_p$ is the edge-cut of the $p$-way partition, and $S$ is the speedup over the serial multilevel algorithm.
References

