Efficient and scalable parallel graph partitioning and static mapping

Jun-Ho Her
François Pellegrini
Summary of the talk

• Introduction
• Sequential techniques
• Parallel $k$-way partitioning
• Experimental results
• Conclusion
• On-going Work: parallel $k$-way static mapping
Introduction - Graph partitioning

• Aims to find small separator or cut keeping graph parts evenly balanced

• Applications in various fields
  • Engineering: VLSI layout, image segmentation
  • Scientific computation: domain decomposition for iterative methods, sparse matrix ordering for direct methods

• NP-complete problem: no polynomial time algorithm for optimal solution in general
  → Many algorithms proposed to date:
    • Heuristics (KL, FM, GG), Meta-heuristics (evolutionary algorithms, bubble-growing), spectral methods, ILP
Introduction - Graph bipartitioning

- $K$-way graph partitioning can be approximated by a sequence of recursive bipartitionings
  - Bipartitioning is easier to implement than $k$-way partitioning
    - No need to choose the destination part of vertices
  - It is only an approximation, but a rather good one
    [Simon & Teng, 1993]
Sequential techniques - Multi-level framework

- Principle [Hendrickson & Leland, 1994]
  - Recursive coarsening (matching and contracting)
  - Initial partitioning of the smallest graph
  - Uncoarsening with succession and refinement of the solution
Sequential techniques - Band graph

- Principle [Chevalier & Pellegrini, 2006]
  - Only local improvements along the projected cut is necessary, so work only on a small band around the cut

- “Much” smaller than full graphs
  - Small constant width around the projected cut
  - Small length due to the projected cut
  → Induction from the “good” initial partition and level-by-level relationship
Sequential techniques - Jug of the Danaides (1)

• Principle [Pellegrini, 2007]
  • Analogous to “bubble growing” algorithms but natively integrates the load balancing constraint
  • The graph is modeled as a set of leaking barrels and pipes
  • Two antagonistic liquids flow from two source vertices
  • Liquids vanish when they meet
Sequential techniques - Jug of the Danaides (2)

- Using JotD as the refinement algorithm in the multi-level process:
  - Smooth interfaces
  - Slower than sequential FM (20 times for 500 iterations, but only 3 times for 40 iterations)
  - Band graph anchor vertices used as source vertices
Parallel $k$-way partitioning

• Three levels of concurrency:
  • In the recursive bipartitioning process itself
    – Straightforward, coarse grain parallelism
    – Redistribution of subgraph data across processors
  • In the coarsening phase of the multi-level algorithm
    – Synchronous probabilistic matching algorithm
    – Folding and duplication in the coarser stages
  • In the refinement process during the uncoarsening phase
Recursive bipartition (1)

• Coarse-grain parallelism

• All subgraphs at a same nested dissection level are processed concurrently on separate subsets of processors

• After a separator has been computed, the two separated subgraphs are folded and redistributed each on a half of the available processors
  • Ability to fold a graph on any number of processors (not only a power of 2)
Recursive bipartition (2)

- The two sub-trees are separated logically but also physically, which reduces network congestion.
- The computation of the two induced subgraphs and their folding can be performed in parallel thanks to the creation of a temporary thread per processor (if MPI is thread-safe).
Coarsening phase (1)

• Matchings are performed in parallel
  • Several algorithms (synchronous or asynchronous) have been studied to reduce dependencies between mating decisions

• The coarsened graph can either be:
  • Kept on the same number of processors: decreases memory and processing cost
  • Folded and duplicated on two subsets of processors: increases quality but also cost
Coarsening phase (2)

- It is preferable to use folding and duplication only in the last stages of the coarsening process
Refinement phase (1)

• As in the sequential algorithm, a distributed band graph is built by keeping only vertices which are at some small distance from the projected separator

• Local optimization algorithms are run on the distributed band graph only
  • Parallel diffusion

• Since this graph is very small, it can be multi-centralized such that sequential local optimization algorithms can be applied to its copies
  • Not scalable but rather inexpensive and yields results which are equivalent to or even better than the sequential version
Refinement phase (2)

• Structure of a distributed band graph
  • Anchor vertices may have very high degrees compared to sequential one
    → Two anchor vertices per process
  • The remote anchor vertices for each part form a clique
## Experimental results

- On CCRT-Platine (932 nodes, 4 dual-core procs per node)
- Test graphs

| Graph          | $|V| \times 10^3$ | $|E| \times 10^3$ | Avg.Deg. | Description                      |
|----------------|------------------|------------------|----------|----------------------------------|
| 10MILLIONS     | 10424            | 78649            | 15.09    | 3D electromagnetics              |
| 23MILLIONS     | 23114            | 175686           | 15.20    | 3D electromagnetics              |
| 45MILLIONS     | 45241            | 335749           | 14.84    | 3D electromagnetics              |
| 82MILLIONS     | 82294            | 609508           | 14.81    | 3D electromagnetics              |
| AUDIKW1        | 944              | 38354            | 81.28    | 3D mechanics mesh                |
| BRGM           | 3699             | 151940           | 82.14    | 3D geophysics mesh               |
| CAGE15         | 5154             | 47022            | 18.24    | DNA electrophoresis              |
| COUPOLE8000    | 1768             | 41657            | 47.12    | 3D structural mechanics          |
| THREAD         | 30               | 2220             | 149.32   | Connector problem                |
Comparison – PT-Scotch vs. ParMeTiS (1)

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of processors:Number of parts</th>
<th>( P_{\text{Peak}}:2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>45MILLIONS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{\text{PTS}} )</td>
<td>1.15E+05</td>
<td>1.13E+06</td>
</tr>
<tr>
<td>( C_{\text{PM}} )</td>
<td>1.26E+05</td>
<td>1.38E+06</td>
</tr>
<tr>
<td>( t_{\text{PTS}} )</td>
<td>24.24</td>
<td>102.29</td>
</tr>
<tr>
<td>( t_{\text{PM}} )</td>
<td>84.55</td>
<td>48.24</td>
</tr>
<tr>
<td><strong>82MILLIONS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{\text{PTS}} )</td>
<td>1.46E+05</td>
<td>1.90E+06</td>
</tr>
<tr>
<td>( C_{\text{PM}} )</td>
<td>1.78E+05</td>
<td>2.12E+06</td>
</tr>
<tr>
<td>( t_{\text{PTS}} )</td>
<td>46.48</td>
<td>189.42</td>
</tr>
<tr>
<td>( t_{\text{PM}} )</td>
<td>176.4</td>
<td>85.87</td>
</tr>
</tbody>
</table>
## Comparison – PT-Scotch vs. ParMeTiS (2)

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of processors: Number of parts</th>
<th>32:2</th>
<th>32:32</th>
<th>32:1024</th>
<th>384:2</th>
<th>384:256</th>
<th>384:1024</th>
<th>(P_{\text{Peak}}:2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AUDIKW1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C_{PTS})</td>
<td>1.08E+05</td>
<td>2.08E+06</td>
<td>1.00E+07</td>
<td><strong>1.05E+05</strong></td>
<td>5.81E+06</td>
<td>9.96E+06</td>
<td>1.11E+05</td>
<td></td>
</tr>
<tr>
<td>(C_{PM})</td>
<td>1.14E+05</td>
<td>2.04E+06</td>
<td>9.76E+06</td>
<td>1.15E+05</td>
<td>5.76E+06</td>
<td>9.76E+06</td>
<td>1.12E+05</td>
<td></td>
</tr>
<tr>
<td>(t_{PTS})</td>
<td><strong>3.51</strong></td>
<td>11.84</td>
<td>17.35</td>
<td>5.87</td>
<td>10.72</td>
<td>10.06</td>
<td>3.01(128)</td>
<td></td>
</tr>
<tr>
<td>(t_{PM})</td>
<td>3.9</td>
<td>3.59</td>
<td>5.27</td>
<td>4.45</td>
<td>4.62</td>
<td>4.51</td>
<td>2.37(192)</td>
<td></td>
</tr>
<tr>
<td><strong>THREAD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C_{PTS})</td>
<td><strong>5.60E+04</strong></td>
<td>6.15E+05</td>
<td><strong>1.82E+06</strong></td>
<td>5.60E+04</td>
<td>1.29E+06</td>
<td>1.82E+06</td>
<td>5.62E+04</td>
<td></td>
</tr>
<tr>
<td>(C_{PM})</td>
<td>5.62E+04</td>
<td>6.03E+05</td>
<td>1.84E+06</td>
<td>5.73E+04</td>
<td>1.29E+06</td>
<td>1.84E+06</td>
<td>5.63E+04</td>
<td></td>
</tr>
<tr>
<td>(t_{PTS})</td>
<td>0.53</td>
<td>0.97</td>
<td>1.07</td>
<td><strong>0.85</strong></td>
<td>1.27</td>
<td>1.28</td>
<td>0.47(16)</td>
<td></td>
</tr>
<tr>
<td>(t_{PM})</td>
<td>0.77</td>
<td>0.75</td>
<td>1.99</td>
<td>2</td>
<td>0.89</td>
<td>2.07</td>
<td>0.52(8)</td>
<td></td>
</tr>
</tbody>
</table>
Comparison – PT-Scotch vs. ParMeTiS (3)

• For most of the cases, PTS shows better partition quality
  • About 20% better in the bipartitioning cases for graph 82MILLIONS

• For the highest numbers of partitions, ParMeTiS shows slight better quality for AUDIKW1, THREAD, and BRGM
  • The graphs have high average degree
  • Greedy nature of recursive bipartitioning scheme emphasized for these graphs
Runtime and Partition Quality (1)

PT-Scotch
45MILLIONS

Time (sec.) [log]

# of Proc [log]

Cut size

# of Proc [log]
Runtime and Partition Quality (2)

**PT-Scotch**

82MILLIONS

![Graph showing runtime and partition quality for PT-Scotch, with log-log scale for number of processors and time, and different markers for different partition sizes.](image)
Cut Size Ratio ($C_{PTS}$ over $C_{PM}$)

- Cut size ratio is most often in favor of PT-Scotch vs. ParMeTiS up to 2048 parts
  - Partition quality of ParMeTiS is irregular for small numbers of parts
  - Gets worse when number of parts increases as recursive bipartitioning prevents performing global optimization
Conclusion

- PT-Scotch compared to ParMeTiS
  - Better partition quality for most cases
  - Faster for small numbers of parts
- Boundary optimization
  - PT-Scotch is the unique parallel tool considering the metric
- On-going work
  - Parallel static mapping (see next slides)
On-going Work - Static mapping (1)

- **Definition**: mapping of $V(S)$ and $E(S)$ of source graph to those of architecture graph, respectively.
- **Partial cost function** for the static mapping:

$$f'_C \overset{\text{def}}{=} \sum_{v \in V(S')} w(\{v, v'\}) |\rho_{S,T}(\{v, v'\})|$$

- To date, Scotch only provides sequential static mapping
  - Parallelization is under way
On-going Work - Static mapping (2)

• A sequential technique: Dual Recursive Bipartitioning
• Brings gains up to 20% on solving time on “regular” multi-core architectures, and even more for really heterogeneous clusters
On-going Work - Static mapping (3)

- Sequential DRB:
  - Accounts for the local cut
  - Accounts for the cocycle (external communication load)
On-going Work - Static mapping (4)

• Parallel Dual Recursive Bipartitioning (first trial)
  • Synchronize at every recursion level
  • Simultaneous decision for each pair

→ Only sequential decision works!
On-going Work - Static mapping (5)

• Two phases in the multi-level framework:
  • Direct $k$-way parallel phase and sequential DRB phase
• $K$-way band graph
  • Simple extension from the bipartitioning case
Thank you!