Hypergraph Partitioning

A Parallel Approach

Aleksandar Trifunovic
Email: at701@doc.ic.ac.uk

Department of Computing, Imperial College, London

What is a Hypergraph

- An extension of a graph
- (Hyper)edges can now connect multiple vertices
- A hypergraph is more expressive than a graph

Hypergraph Partitioning Models

- Sparse matrix decomposition for parallel MxV
- VLSI circuit layout
- Data layout in distributed databases
- Clustering in data mining

k-way Hypergraph Partitioning

- Find k disjoint parts (subsets) of the vertex set (a partition)
- Part weight bounded by prescribed threshold
- Aim to optimise interconnect between parts
- Finding optimal solution is NP-hard
**k-way Hypergraph Partitioning**

- Content with finding good sub-optimal solutions
- We use heuristic approaches
  - Iterative improvement
  - Simulated Annealing
  - Genetic Algorithms
  - Tabu Search
  - Geometric Representations
- See survey by Alpert and Kahng (1995)

**The Multilevel Paradigm**

- The Coarsening Phase
  - Identifies strongly connected vertices
    - Proceed vertex–by–vertex
    - Proceed hyperedge–by–hyperedge
  - Merge to form vertices of coarse hypergraph
  - Construct hyperedges of coarse hypergraph
    - Can discard singleton hyperedges
    - Can discard duplicate hyperedges
  - Continue until coarse graph has $O(k)$ vertices
  - Proceed with Initial Partitioning Phase

- The Initial Partitioning Phase
  - Computes a partition of the coarsest hypergraph
    - Hypergraph small
    - Can use (almost) any method
  - Choose best from a number of independent runs
  - Usually fastest of the three phases

- With increasing problem size:
  - It scales well in terms of partition quality
  - It scales well in terms of run time
The Refinement Phase

- Project computed partitions
- Can further refine partition
  - Use iterative improvement techniques
- Bisection vs. \( k\)-way refinement
- V-Cycle refinement
  - Use partition as input to coarsening
  - Initial partitioning and refinement
  - Recursive formulation

Recap

- Need to use heuristic approaches
  - Multilevel Paradigm best so far
- Thus far only sequential algorithms
  - Imposes limit on capacity
  - Imposes a limit on runtime
  - Bottleneck prior to parallel computation?
- Parallel partitioning suggests itself
  - Can address above issues?

The Parallel Algorithm

- Coarsening and uncoarsening done in parallel
- Initial partitioning can be done serially
- Coarsest hypergraph has \( O(k) \) vertices:
  - Assemble the coarsest hypergraph at each processor
  - Compute initial partition serially
  - Choose best partition for uncoarsening
- Parallel Uncoarsening:
  - Project partition
  - Parallel greedy \( k\)-way refinement

Issues in Parallel Partitioning

- Harder to enforce maximum coarse vertex weight
- Even though processors may compute positive gain for their vertex moves, overall there may be no gain at all
**Key Features**

- Hashing the hyperedges:
  - Distributes uniformly across processors
  - Maintains computational load balance
  - Enables cost-optimal communication (on hypercube)

- Two-phase communication:
  - In first phase, matching requests (coarsening) and vertex movement requests (refinement) only permitted from lower to higher index
  - Vice-versa for second phase

- Benefits:
  - Enables matches between remote mutually requesting vertices
  - Helps to enforce maximum coarse vertex weight
  - Prevents vertex thrashing

**The Asymptotic Scalability Model**

- Assumptions:
  - Average vertex degree very small
  - The number of hyperedges reduced by constant factor at every coarsening step
  - Hyperedges uniformly distributed across the processors
  - Take sequential multilevel algorithm as base-case comparison

- Isoefficiency function:
  - \( W = O(k^2 p^2 (\log p + \log k)) \)
  - Same order as that of Karypis’ and Kumar’s parallel graph partitioning algorithm

**The Parkway 2.0 Tool**

- Linux library written in C++ using MPI

- Parallel Coarsening Options:
  - Supports parallel First Choice algorithm

- Sequential Partitioning Options:
  - Interfaces with HMETIS PartKway()
  - Interfaces with PaToH partitioning routines
  - Provides generic recursive bisection partitioning

- Parallel Uncoarsening Options:
  - Parallel greedy \( k \)-way refinement
  - Parallel V-cycle iterations

**Experimental Results**

- Beowulf Linux Cluster architecture:
  - Intel Xeon 2.0 GHz nodes
  - 2GB memory per node
  - Myrinet network - 250 MB/s peak th’put

- Hypergraph representation of transition matrices:
  - High-level semi-Markov voting system model
  - Underlying hypergraph (for 175 voters) has:
    - 1 140 050 vertices
    - 1 140 050 hyperedges
    - 6 657 722 pins
Experimental Results ctd.

<table>
<thead>
<tr>
<th>$p$</th>
<th>Tool</th>
<th>partition size $k = 8$</th>
<th>time (s)</th>
<th>cut-size avg (best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PaToH</td>
<td>41.40</td>
<td>22863</td>
<td>22191</td>
</tr>
<tr>
<td>1</td>
<td>hMeTiS</td>
<td>53.76</td>
<td>25387</td>
<td>24600</td>
</tr>
<tr>
<td>2</td>
<td>Parkway 2.0</td>
<td>66.30</td>
<td>26227</td>
<td>25605</td>
</tr>
<tr>
<td>4</td>
<td>Parkway 2.0</td>
<td>30.41</td>
<td>26230</td>
<td>25785</td>
</tr>
<tr>
<td>8</td>
<td>Parkway 2.0</td>
<td>14.66</td>
<td>26406</td>
<td>26160</td>
</tr>
<tr>
<td>16</td>
<td>Parkway 2.0</td>
<td>6.57</td>
<td>26671</td>
<td>26548</td>
</tr>
<tr>
<td>32</td>
<td>Parkway 2.0</td>
<td>4.10</td>
<td>26570</td>
<td>25786</td>
</tr>
</tbody>
</table>

Comparison with current state-of-the-art:
- Partition quality
- Runtime - scalability

Conclusion

- Benefits of parallelisation:
  - Increase in capacity ($\approx 10^7$ vertices to date)
  - Proven analytical and empirical scalability
- Partition quality:
  - Comparable with best serial hypergraph partitioning tools (hMeTiS and PaToH)
  - Better quality solution than graph-based approximations
- Looking for applications in other domains
- Parkway 2.0 available to download (soon):
  - http://www.doc.ic.ac.uk/~at701/parkway