PT-Scotch in Solstice and beyond: where to go now

François Pellegrini
Summary of the talk

• The Scotch project
• The multi-level framework and its parallelization
• Parallel static mapping
• Conclusion
The **Scotch** project
Graph partitioning (1)

- Graph partitioning is an ubiquitous technique which has proven useful in a wide number of application fields
  - Used to model domain-dependent optimization problems
  - “Good solutions” take the form of partitions which minimize vertex or edge cuts, while balancing the weight of graph parts
- NP-complete problem in the general case
- Many algorithms have been proposed in the literature:
  - Graph algorithms, evolutionary algorithms, spectral methods, linear optimization methods, …
Graph partitioning (2)

• Two main problems for our team:
  • Sparse matrix ordering for direct methods
  • Domain decomposition for iterative methods

• These problems can be modeled as graph partitioning problems on the adjacency graph of symmetric positive-definite matrices
  • Edge separator problem for domain decomposition
  • Vertex separator problem for sparse matrix ordering by nested dissection
The Scotch roadmap

- Devise robust parallel graph partitioning methods
  - Should handle graphs of more than a billion vertices distributed across one thousand processors
- Improve sequential graph partitioning methods if possible
  - Multi-level FM-like algorithms are both fast and efficient on a very large class of graphs but FM algorithms are intrinsically sequential
- Investigate alternate graph models (meshes/hypergraphs)
- Provide a software toolbox for scientific applications
  - Scotch sequential software tools
  - PT-Scotch parallel software tools
Design constraints

• Parallel algorithms have to be carefully designed
  • Algorithms for distributed memory machines
  • Preserve independence between the number of parts \( k \) and the number of processing elements \( P \) on which algorithms are to be executed
  • Algorithms must be “quasi-linear” in \(|V|\) and \(|E|\)
  • Constants should be kept small!
    – Theory is not likely to help much...

• Data structures must be scalable:
  • In \(|V|\) and/or \(|E|\) : graph data must not be duplicated
  • In \( P \) and \( k \) : arrays in \( k|V| \), \( k^2 \), \( kP \), \( P|V| \) or \( P^2 \) are forbidden
The multi-level framework and its parallelization
From k-partitioning to recursive 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]
Recursive bipartitioning in parallel

- After a separator has been computed, the two separated subgraphs are folded and redistributed each on a half of the available processors
  - All subgraphs at a same level are processed concurrently on separate subsets of processors
  - Ability to fold a graph on any number of processors (not only a power of 2)
Multi-level framework

- Principle [Hendrickson & Leland, 1994]
  - Create a family of topologically equivalent coarser graphs by clustering groups of vertices
  - Compute an initial partition of the smallest graph
  - Propagate back the result, with local refinement
Coarsening in parallel

• 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
Parallel matching

• Parallel coarsening bases on parallel matching
  • These matchings do not need to be maximal

• Synchronization between non-local neighbors is critical
  • Dependency chains or loops between mating requests can stall the whole algorithm because of sequential constraints

• Some distributed tie-breaking is required
• Too many requests decrease matching probability
Parallel probabilistic matching

- Principle [Chevalier, 2007]
  - Do not discriminate between local and non-local neighbors when selecting a neighbor for mating
  - Vertices request for matings with their neighbors (whether local or remote) with a prescribed probability

- Reduces topological biases and converges quickly
  - 5 collective passes are enough to match 80% of the vertices on average

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>53.3</td>
<td>12.3</td>
<td>50.4</td>
<td>0.7</td>
</tr>
<tr>
<td>C2</td>
<td>68.7</td>
<td>13.6</td>
<td>51.6</td>
<td>2.2</td>
</tr>
<tr>
<td>C3</td>
<td>76.2</td>
<td>12.2</td>
<td>52.5</td>
<td>3.3</td>
</tr>
<tr>
<td>C4</td>
<td>81.0</td>
<td>10.6</td>
<td>53.2</td>
<td>4.0</td>
</tr>
<tr>
<td>C5</td>
<td>84.5</td>
<td>9.1</td>
<td>53.7</td>
<td>4.5</td>
</tr>
<tr>
<td>LF</td>
<td>100.0</td>
<td>0.0</td>
<td>59.4</td>
<td>6.8</td>
</tr>
</tbody>
</table>
Band graphs

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

- Reduce problem space dramatically
  - Allow one to use expensive algorithms, such as genetic algorithms
Band graphs in parallel

- Anchor vertices may have very high degrees compared to sequential band graphs
  - Two anchor vertices per process
  - Remote anchor vertices for each part form a clique
    - Will soon be a hypercube to accommodate for large numbers of processes
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
Jug of the Danaides (2)

- Using JotD as the refinement algorithm in the multi-level process:
  - Yields smooth interfaces
  - Is slower than sequential FM (20 times for 500 iterations, but only 3 times for 40 iterations)
- Band graph anchor vertices used as source vertices
Runtime and sparse matrix ordering quality

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of processes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>audikw1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>O&lt;sub&gt;PTS&lt;/sub&gt;</strong></td>
<td></td>
<td><strong>5.73E+12</strong></td>
<td><strong>5.65E+12</strong></td>
<td><strong>5.54E+12</strong></td>
<td><strong>5.45E+12</strong></td>
<td><strong>5.45E+12</strong></td>
<td><strong>5.45E+12</strong></td>
</tr>
<tr>
<td><strong>O&lt;sub&gt;PM&lt;/sub&gt;</strong></td>
<td></td>
<td><strong>5.82E+12</strong></td>
<td><strong>6.37E+12</strong></td>
<td><strong>7.78E+12</strong></td>
<td><strong>8.88E+12</strong></td>
<td><strong>8.91E+12</strong></td>
<td><strong>1.07E+13</strong></td>
</tr>
<tr>
<td><strong>t&lt;sub&gt;PTS&lt;/sub&gt;</strong></td>
<td></td>
<td><strong>64.14</strong></td>
<td><strong>43.72</strong></td>
<td><strong>31.25</strong></td>
<td><strong>20.66</strong></td>
<td><strong>13.86</strong></td>
<td><strong>9.83</strong></td>
</tr>
<tr>
<td><strong>t&lt;sub&gt;PM&lt;/sub&gt;</strong></td>
<td></td>
<td><strong>32.69</strong></td>
<td><strong>23.09</strong></td>
<td><strong>17.15</strong></td>
<td><strong>9.80</strong></td>
<td><strong>5.65</strong></td>
<td><strong>3.82</strong></td>
</tr>
</tbody>
</table>

![Graph showing runtime and sparse matrix ordering quality](chart.png)
Runtime and partition quality (1)

PT-Scotch
45MILLIONS

Time (sec.) [log]

# of Proc [log]

Cut size

# of Proc [log]

- 2 parts
- 4 parts
- 8 parts
- 16 parts
- 32 parts
- 64 parts
- 128 parts
- 256 parts
- 512 parts
- 1024 parts
- 2048 parts
• 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
Runtime and partition quality (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
Parallel direct $k$-way graph partitioning

- Extension to $k$ parts of the multilevel framework used for recursive bipartitioning
  - Straightforward for the multi-level framework itself
  - Relies on distributed $k$-way band graphs

- Stability problems with our diffusion-based algorithms
  - Artifacts when there are too few vertices per part
Parallel static mapping
Architectural considerations matter

• Upcoming machines will comprise very large numbers of processing units, and will possess NUMA / heterogeneous architectures
  • More than a million processing elements on the Blue Waters machine to be built at UIUC (joint lab with INRIA)

• Impacts on our research:
  • Topology of target architecture has to be taken into account
    – Static mapping and not only graph partitioning
  • Dynamic repartitioning capabilities are mandatory
Parallel static mapping (1)

- Compute a mapping of $V(S)$ and $E(S)$ of source graph $S$ to $V(T)$ and $E(T)$ of target architecture graph $T$, respectively.

- Communication cost function accounts for distance:

  $$f_C(\tau_{S,T}, \rho_{S,T}) \overset{\text{def}}{=} \sum_{e_S \in E(S)} w(e_S) |\rho_{S,T}(e_S)|$$

- Static mapping features are already present in the sequential Scotch library.
  - We have to go parallel.
Parallel static mapping (2)

• Partial cost function in the context of recursive bipartitioning

\[ f'_C(\tau_{S,T}, \rho_{S,T}) \overset{\text{def}}{=} \sum_{\{v, v'\} \in E(S')} w(\{v, v'\}) \mid \rho_{S,T}(\{v, v'\}) \mid \]

\[ v \in V(S') \]

• Decision making depends on available mapping information
Parallel static mapping (3)

- Recursive bi-mapping cannot be transposed in parallel
  - All subgraphs at some level are supposed to be processed simultaneously for parallel efficiency
  - Yet, ignoring decisions in neighboring subgraphs can lead to “twists”

- Only sequential processing works!
Parallel static mapping (4)

- Parallel multilevel framework for static mapping
  - Parallel coarsening and k-way mapping refinement
  - Initial mapping by sequential recursive bi-mapping
Parallel static mapping (5)

• If the number of parts gets bigger than the size of the biggest graph to be stored on a single node, the sequential initial mapping phase cannot take place
  – Above 1 million parts (that is, cores)

• New roadmap: be able to map graphs of about a trillion vertices spread across a million processing elements
  • Focus on scalability problems related to the number of processors
Conclusion
Solstice goals achieved!

- Some users have experimented with Scotch up to the symbolic frontiers that we had defined
  - Katie Lewis at LLNL: graphs up to 800 Mvertices and 2 Gedges partitioned on 4096 procs
  - Scalability in terms of memory and runtime
  - Load imbalance increases along with the number of processes
- We are stuck by MPI interface limitations
  - All displacement and count values are expressed as ints (32 bits)
  - We must have full 64-bit MPI implementations
The Scotch software package

- All of the algorithms are available to the community
  - Scientific reproducibility
  - Freely available from the INRIA Gforge
  - Modular and documented code ($\approx 100k$ lines of C)
- Upgrades on a regular basis
  - Version 4.0: February 2004: 2500+ direct downloads
  - About one major release per year (5.2 almost ready)
- Usage by third-party software
  - Emilio (CEA/CESTA), Code_Aster (EDF), Dolfin/Fenics (Simula), MUMPS (ENSEEITH, LIP & LaBRI), PaStiX (LaBRI), SuperLU (U. C. Berkeley), Zoltan (Sandia), ...
K-way vertex partitioning with overlap (1)

- Parallel matrix computations
  - Block decomposition with overlap

- Several application domains
  - Quantum chemistry
  - Schur complement techniques for linear system solving
K-way vertex partitioning with overlap (2)

- Compute k vertex-separated parts
- Balance part loads according to inner vertices as well as neighboring separator vertices
  - Separator vertices may contribute to several parts
Dynamic remeshing and repartitioning

• Move upwards from the production of general-purpose tools to more specific application domains
  • Motivation for joining the Bacchus team
• Parallel adaptive remeshing
  • Take into account the numerical stability of the problem being studied
  • Take advantage of the work done in PT-Scotch on distributed graphs
• Dynamically repartition the remeshed graphs
Thanks!

• To all the past and present “Scotch-men”:
  • Cédric Chevalier
  • Charles-Edmond Bichot
  • Jun-Ho Her
  • Sébastien Fourestier
  • Cédric Lachat

• The journey is going on...