Engineering State-of-the-Art Graph Partitioning Libraries @KIT

Vitaly Osipov, Peter Sanders, Christian Schulz
Overview

- Introduction
- Parallel Graph Partitioning (KaPPa)
- n-Level Graph Partitioning (KaSPar)
- Experimental Evaluation
- Future Work
Simulation - FEM

- Simulation space is discretized into a mesh
- Solution of partial differential equations are approximated by linear equations
- Number of vertices can become quite large → time and memory
- Parallel processing required
The Common Parallel Approach

- Mesh partitioned via dual graph
  1. Each volume (data, calculation) is represented by a vertex
  2. Interdependencies are represented by edges
- All PE’s get same amount of work
- Communication is expensive

**Graph Partitioning Problem:**
Partition a graph into (almost) equally sized blocks, such that the number of edges connecting vertices from different blocks is minimal.
\( \epsilon \)-Balanced Graph Partitioning

Partition graph \( G = (V, E, c : V \to \mathbb{R}_{>0}, \omega : E \to \mathbb{R}_{>0}) \) into \( k \) disjoint blocks s.t.

- total node weight of each block \( \leq \frac{1 + \epsilon}{k} \) total node weight
- total weight of cut edges as small as possible

Applications:
finite element simulations, VLSI design, route planning, . . .
Multi-Level Graph Partitioning

Successful in existing systems:
DiBaP, Chaco, Jostle, Metis, Scotch,...
Multi-Level Graph Partitioning

- Edge Contraction
  - \( u, v \rightarrow x \)
  - \( \omega(\{x, z\}) = \omega(\{u, z\}) + \omega(\{v, z\}). \)
  - \( c(x) = c(u) + c(v) \)

![Diagram of graph partitioning process](image)
Why Yet Another MGP Algorithms?

Karlsruhe Parallel Partitioner
Karlsruhe Sequential Partitioner

- scalable parallel algorithm for the case $k = \# \text{ of processors}$
- large inputs
- high quality (better (even parallel) than other seq. algorithms)
- understand / engineer multi level method
- bridge gaps theory ↔ practice
Matching Selection

Goals:

1. large edge weights $\leadsto$ sparsify
2. large #edges $\leadsto$ few levels
3. uniform node weights $\leadsto$ “represent” input
4. small node degrees $\leadsto$ “represent” input

$\leadsto$ unclear objective

$\leadsto$ gap to approx. weighted matching

which only considers 1., 2.

Our Solution:

Apply approx. weighted matching to general edge rating function
Edge Ratings Tried

\[
\omega(\{u, v\})
\]

\[
\text{expansion}(\{u, v\}) := \frac{\omega(\{u, v\})}{c(u) + c(v)}
\]

\[
\text{expansion}^*(\{u, v\}) := \frac{\omega(\{u, v\})}{c(u)c(v)}
\]

\[
\text{expansion}^{*2}(\{u, v\}) := \frac{\omega(\{u, v\})^2}{c(u)c(v)}
\]

\[
\text{innerOuter}(\{u, v\}) := \frac{\omega(\{u, v\})}{\text{Out}(v) + \text{Out}(u) - 2\omega(u, v)}
\]

where \(c = \text{node weight}, \ \omega = \text{edge weight}, \)

\(\text{Out}(u) := \sum_{\{u, v\} \in E} \omega(\{u, v\})\)
Approx. Weighted Matching

- Use parallel prepartitioner (optional)
  currently geometric recursive bipartitioning when coords. available
- Global Path Algorithm [MaueSanders 2007] locally
- parallel [ManneBisseling 2007] for border nodes
Initial Partitioning

- Every PE can perform initial partitioning using different seeds
- Compared PARTY, PMETIS, and SCOTCH
- SCOTCH yielded best results
Use linear time local search [Fiduccia Mattheyses 82] on multiple pairs of blocks in parallel.
Finding Pairs of Blocks

- Color edges of quotient graph $Q$
- Several parallel algorithms tried
- Each color $\rightarrow$ matching in $Q$ $\rightarrow$ independent block pairs
Pairwise Local Search

exchange boundaries

two local searches

adopt best

New queue selection strategy topGain
Karlsruhe Sequential Partitioner

1. Contraction
   - contract a single edge between two levels
     - possibly n levels
     - finegrained contraction $\Rightarrow$ consequitive levels are very similar
     - no matching algorithm required
   - use of different edge ratings $\Rightarrow$ uniform distribution of node weights
   - priority queue defines the order of edges to be contracted

2. Local Search
   - efficient stopping criterion $\Rightarrow$ avoid quadratic runtime

3. General
   - Trial Trees $\Rightarrow$ improve quality by independent trials
Contraction

- edge rating - $\text{expansion}^*(\{u, v\}) = \frac{\omega(\{u, v\})}{c(u)c(v)}$
- addressable priority queue based on pairing heaps
- dynamic graph data structure
Nodes $\rightsquigarrow$ unmarked, **active**, marked

**active** nodes
- compute gains of moving from one block to another
- choose the maximum gain

**when moved**
- **active** $\rightsquigarrow$ **marked**
- can’t become **active** anymore
- unmarked neighbours of **marked** $\rightsquigarrow$ **active**
Local Search - Stopping Criteria

- touching each node on each level would lead to $\Omega(n^2)$ runtime
- $\Rightarrow$ need flexible stopping criteria
- gains in each step $\rightsquigarrow$ identically distributed, independent random variables
  - expectation $\mu$
  - variance $\sigma^2$
- compute $\mu$ and $\sigma^2$ from previous steps
- stop after $p$ steps if $p\mu^2 > \alpha \sigma^2 + \beta$
Results – Example

Street network Europe $|V| = 18M$, $|E| = 44M$, $k = 16$

KaSPar$\leftrightarrow$ParMetis

EC: 1401

EC: 5576
Scalability

Street network Europe ($|V| = 18M, |E| = 44M$)
Quality
Comparison with Other Systems

Geometric mean, imbalance $\epsilon = 0.03$:
11 graphs (78K–18M nodes) $\times k \in \{2, 4, 8, 16, 64\}$

<table>
<thead>
<tr>
<th>algorithm</th>
<th>large graphs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>best</td>
<td>avg.</td>
</tr>
<tr>
<td>KaSPar strong</td>
<td>12 450</td>
<td>12 584</td>
</tr>
<tr>
<td>KaPPa strong</td>
<td>13 323</td>
<td>+8%</td>
</tr>
<tr>
<td>Scotch</td>
<td>14 475</td>
<td>+19%</td>
</tr>
<tr>
<td>kMetis</td>
<td>15 540</td>
<td>+32%</td>
</tr>
</tbody>
</table>

- Repeating scotch as long as KasPar strong run and choosing the best result $\sim 12.1\%$ larger cuts
- Walshaw instances, road networks, Florida Sparse Matrix Collection, random Delaunay triangulations, random geometric graphs, social networks.
Sources of Quality Improvement

1. edge ratings  
   largest effect by far
2. more localized refinement.  
   Surprise! previous parallelizations caused worse quality
3. better queue selection
4. better matching algorithms
5. more time invested
Future Work

1. try other pairwise improvers (Flows or Diffusion) and try V-Cycles (or F-Cycles) graph partitioning ⇒ KaFFPa
2. better initial partitioning for large k
3. integration into Meta-heuristic
4. exploit shared memory parallelism
DIMACS Implementation Challenge
Announcement

- someitem
Thank you!