Scalable Parallel Refinement of Graph Partitions

Christian Schulz$^{1,2}$

$^1$Institute for Theoretical Computer Science, Algorithmics II
University Karlsruhe

$^2$Institute for Applied and Numerical Mathematics
University Karlsruhe

5. Mai 2009
Table of Contents

Motivation
  The Problem
  Motivation

The Algorithm
  Multilevel Approach
  Our Approach - The Algorithm

Current Results
  Quality
  Runtime

Future Work
Outline

Motivation

The Problem

Motivation

The Algorithm

Multilevel Approach

Our Approach - The Algorithm

Current Results

Quality

Runtime

Future Work
The Problem

▶ Consider a graph $G = (V, E)$, $V$ set of vertices, $E \subseteq V \times V$ set of edges

▶ given $G$ and $k > 1$, partition $V$ into $k$ subsets $V_1, V_2, \ldots, V_k$ such that:
   1. all parts are disjoint
   2. all parts have equal size
   3. the number of edges between different parts is minimized
The Problem

- **The Problem**: given $G$ and $k > 1$, partition $V$ into $k$ subsets $V_1, V_2, \ldots V_k$ such that:
  1. all parts are disjoint
  2. all parts have equal size
  3. the number of edges with endpoints in different parts is minimized

- practical applications: small imbalance $\epsilon$ in the part sizes: balance criterion is $\max_i |V_i| \leq (1 + \epsilon)|V|/k$
Outline

Motivation

The Problem

Motivation

The Algorithm

Multilevel Approach

Our Approach - The Algorithm

Current Results

Quality

Runtime

Future Work
Motivation

- important problem with applications in many areas
- i.e. VLSI design, scientific computing (Finite Element Method, LGS)
- the efficient implementation of many parallel algorithms usually requires the solution to a graph partitioning problem
Motivation

a great problem with applications everywhere
Outline

Motivation
The Problem
Motivation

The Algorithm
Multilevel Approach
Our Approach - The Algorithm

Current Results
Quality
Runtime

Future Work
Multilevel Approach

Multilevel K-way Partitioning

Initial Partitioning Phase

Coarsening Phase

Uncoarsening Phase

$G_0$

$G_1$

$G_2$

$G_3$

$G_4$
Outline

Motivation
    The Problem
    Motivation

The Algorithm
    Multilevel Approach
    Our Approach - The Algorithm

Current Results
    Quality
    Runtime

Future Work
My Task
Definitions

- **Edge coloring**
  1. assign a color to each edge of a graph
  2. no vertex has two edges that have the same color
  3. use as few colors as possible

- **Matching**
  1. a set of pairwise non-adjacent edges (no two edges share a common vertex)

- **Quotient graph**
  1. one vertex per partition
  2. \( e = \{P_i, P_j\} \in E_Q \iff \exists\{v, w\} \in E : P[v] = P_i \text{ and } P[w] = P_j \) \((i \neq j)\)

- **Note**: each color in a edge coloring induces a matching
Algorithm Overview

**Algorithm 1 Overview**

**Data**: $G = (V, E)$, initial partition $\mathcal{P}$

distribute $G$ according to $\mathcal{P}$

compute quotient graph $Q = (V_Q, E_Q)$ in parallel

compute edge coloring $C : E_Q \rightarrow S$ of $Q$ in a distributed way

**forall** $c \in S$ **do**

**foreach** $e = \{P_i, P_j\} \in \mathcal{M}_C(c)$ **parallel** do

perform local two-way refinement between $P_i$ and $P_j$
Example
A sample graph with Initial Partition (EC = 17)
Example

Resulting Quotient Graph
Example

Color the edges of the quotient graph
Main Idea
Refine in Parallel
Main Idea

After Refinements
Example
After Refinements (EC = 16)
Parallel Computation of the Quotient Graph

- each vertex has an id
- each processor has $v_{min}$ and $v_{max}$
- distribute this *table* among the processors
- $\Rightarrow \Phi : Id \rightarrow PartitionID$
- local: iterate over local edge list
- each edge has a targetid $\Rightarrow$ targetPartition $= \Phi(targetid)$
Obtaining Edge Colorings
In A Distributed Way - Last Time: Maximal Matching Based

**Algorithm 2** Distributed Edge Coloring Algorithm

**Data**: Neighbors $N$, $G = (V, E)$

**while** $E \neq \emptyset$ **do**

  find maximal matching parallel
  color matched edges with Color c, inc c
  remove matched edges

**end**
Obtaining Edge Colorings
In A Distributed Way - Today: First Fit Based

Algorithm 3 Edge Coloring Algorithm (Sequential)

Data : $G = (V, E)$
Result : Edge Coloring of $G$
forall random $e \in E$ do
  ▶ color $e$ with smallest free color

- worst case: $2\Delta - 1$ Colors, with $\Delta := \max_{v \in V} \deg(v)$
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists
- **active:**
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT(c) or REJECT
  - REJECT all incoming request
- **inactive:**
  - work on all incoming messages
  - color edge with smallest color c in intersect of free color lists
  - reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (→ active / inactive)
- initialize free color lists

- active:
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT(c) or REJECT
  - REJECT all incoming request

- inactive:
  - work on all incoming messages
  - color edge with smallest color c in intersect of free color lists
  - reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists
- **active:**
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT($c$) or REJECT
  - REJECT all incoming request
- **inactive:**
  - work on all incoming messages
  - color edge with smallest color $c$ in intersect of free color lists
  - reply with ACCEPT($c$)
Distributed First Fit

Overview

- throw a coin (→ active / inactive)
- initialize free color lists

**active:**
- choose neighbor
- send free color list, wait for reply
- reply is ACCEPT(c) or REJECT
- REJECT all incoming request

**inactive:**
- work on all incoming messages
- color edge with smallest color c in intersect of free color lists
- reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists
- **active:**
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT(c) or REJECT
  - REJECT all incoming request
- **inactive:**
  - work on all incoming messages
  - color edge with smallest color c in intersect of free color lists
  - reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists

**active:**
- choose neighbor
- send free color list, wait for reply
- reply is ACCEPT(c) or REJECT
- REJECT all incoming request

**inactive:**
- work on all incoming messages
- color edge with smallest color c in intersect of free color lists
- reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists

**active:**
- choose neighbor
- send free color list, wait for reply
- reply is ACCEPT(c) or REJECT
- REJECT all incoming request

**inactive:**
- work on all incoming messages
- color edge with smallest color c in intersect of free color lists
- reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (→ active / inactive)
- initialize free color lists
- **active:**
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT(c) or REJECT
  - REJECT all incoming request
- **inactive:**
  - work on all incoming messages
  - color edge with smallest color $c$ in intersect of free color lists
  - reply with ACCEPT(c)
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists
- **active:**
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT\(c\) or REJECT
  - REJECT all incoming request
- **inactive:**
  - work on all incoming messages
  - color edge with smallest color \(c\) in intersect of free color lists
  - reply with ACCEPT\(c\)
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists
- **active:**
  - choose neighbor
  - send free color list, wait for reply
  - reply is ACCEPT$(c)$ or REJECT
  - REJECT all incoming request
- **inactive:**
  - work on all incoming messages
  - color edge with smallest color $c$ in intersect of free color lists
  - reply with ACCEPT$(c)$
Distributed First Fit

Overview

- throw a coin (-> active / inactive)
- initialize free color lists

**active:**
- choose neighbor
- send free color list, wait for reply
- reply is ACCEPT(c) or REJECT
- REJECT all incoming request

**inactive:**
- work on all incoming messages
- color edge with smallest color c in intersect of free color lists
- reply with ACCEPT(c)
Distributed First Fit

Explained

- initial uncolored quotient graph
Distributed First Fit

Explained

- throw a coin (→ active/inactive)
Distributed First Fit

Explained

- initialize free color lists
Distributed First Fit

Explained

- choose random neighbor
Distributed First Fit
Explained

» active cpus reply with reject
Distributed First Fit
Explained

4: compute smallest free color in intersect
Distributed First Fit
Explained

- again throw a coin
- 4: reply with 0 and color the edge
- 1: recv color 0 and color the edge
Distributed First Fit

Explained

1: 1, 2, 3, 4, 5, 6, 7
2: 0, 1, 2, 3, 4, 5, 6, 7
3: 0, 1, 2, 3, 4, 5, 6, 7
4: 1, 2, 3, 4, 5, 6, 7

▶ and so on ...
Distributed First Fit
Explained

1: 1, 2, 3, 4, 5, 6, 7
2: 0, 1, 2, 3, 4, 5, 6, 7
3: 0, 1, 2, 3, 4, 5, 6, 7
4: 1, 2, 3, 4, 5, 6, 7

▶ and so on ...
Distributed First Fit
Explained

► and so on ...

► if all local edges colored -> stay inactive

0

1: 1, 2, 3, 4, 5, 6, 7

1

2: 0, 1, 2, 3, 4, 5, 6, 7

2

4: 2, 3, 4, 5, 6, 7

4

3: 0, 2, 3, 4, 5, 6, 7

3
Distributed First Fit

Explained

鸠饭

and so on ...
Distributed First Fit
Explained

and so on ...
Distributed First Fit
Explained

and so on ...
Distributed First Fit

Explained

\[ 1: 1, 3, 4, 5, 6, 7 \]
\[ 2: 1, 2, 3, 4, 5, 6, 7 \]
\[ 4: 2, 3, 4, 5, 6, 7 \]
\[ 3: 3, 4, 5, 6, 7 \]

▶ and so on ...
Distributed First Fit

Explained

and so on ...
Distributed First Fit

Explained

1: 1, 3, 4, 5, 6, 7

2: 1, 2, 3, 4, 5, 6, 7

4: 2, 3, 4, 5, 6, 7

3: 3, 4, 5, 6, 7

and so on...
Distributed First Fit
Explained

▶ and so on ...
Distributed First Fit
Explained

and so on ...

Schulz, Christian
Scalable Parallel Refinement of Graph Partitions
Distributed First Fit
Explained

and so on ...
Distributed First Fit

Explained
Edge Coloring As A Communication Protocol
A Processor Pair Refinement

Initial Situation - Locally Stored Partitions

CPU $P_i$

CPU $P_j$
A Processor Pair Refinement

Exchange Data

CPU $P_i$

CPU $P_j$
A Processor Pair Refinement

Both Processors Refine - Exchange Improvement

CPU $P_i$

Winner (Improvement)

CPU $P_j$
A Processor Pair Refinement
Winner Sends Back Changes
Reducing Communication Volume - BFS

- **assumption:** changes happen only in a small area around the boundary

- **idea:** instead of exchanging whole partitions, exchange only a small area around the boundary
Breadth-First Search (BFS)

- Put the boundary nodes on a queue $Q$
- Put the boundary nodes on a list $B$
- **While** queue not empty **do**
  - $v \leftarrow Q\.top()$
  - **If** search depths $> d(v)$ **Return** $B$
  - **Else** Put Neighbors of $v$ on $Q$ and $B$
Local Optimization
Fiducia and Mattheyses - Algorithm

- **Compute gain for all nodes** \( v \in V \)
  
  \[ g(v) = d_{ext}(v) - d_{int}(v) \]

- **store gain of boundary nodes** (e.g. in a heap)
Local Optimization
Fiducia and Mattheyses - Algorithm

- Compute gain for all nodes \( v \in V \)
- \( g(v) = d_{\text{ext}}(v) - d_{\text{int}}(v) \)
- store gain of boundary nodes (e.g. in a heap)
Local Optimization
Fiducia and Mattheyses - Algorithm

- Compute gain for all nodes \( v \in V \)
- \( g(v) = d_{ext}(v) - d_{int}(v) \)
- store gain of boundary nodes (e.g. in a heap)
Local Optimization
Fiducia and Mattheyses - Algorithm

- move nodes using heap selection strategy
- move each node only once
- update gain values of neighbors

Step: 0
Edge Cut: 5
Local Optimization
Fiducia and Mattheyses - Algorithm

- move nodes using heap selection strategy
- move each node only once
- update gain values of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Local Optimization
Fiducia and Mattheyses - Algorithm

- move nodes using heap selection strategy
- move each node only once
- update gain values of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
Local Optimization
Fiducia and Mattheyses - Algorithm

- move nodes using heap selection strategy
- move each node only once
- update gain values of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Local Optimization

Fiducia and Mattheyses - Algorithm

- stop after limit
- take best occurred edge cut w.r.t. edge cut and balance

<table>
<thead>
<tr>
<th>Step</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Local Optimization
Fiducia and Mattheyses - Algorithm

- many opportunities
- search limit, work hard mode, heap selection, balance

<table>
<thead>
<tr>
<th>Step: Edge Cut:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Data Structure
Hashtable as Buffer

- global node id’s / distributed graph data structure
- we use an hash table as buffer for the received nodes
- we implemented hashing with linear probing and wraparound (but are able to use STL as well)
Global Convergence

1. Global Convergence and Theoretical Upper Bound for Speedup is proven (pathological examples/ stars / what is the worst case?)

2. Global Convergence Mode: run until the first time no global reduction of edge cut is possible
Outline

Motivation
The Problem
Motivation

The Algorithm
Multilevel Approach
Our Approach - The Algorithm

Current Results
Quality
Runtime

Future Work
Walshaw’s Partition Archive
Postoptimization of Largest Graphs in Walshaw’s Benchmark - Fully Balanced Partitions

| Graph     | 8   | 16  | 32   | 64   | |V|    | |E|    |
|-----------|-----|-----|------|------|------|------|------|
| auto      | 48094 | 48901 | 79951 | 81500 | 123548 | 125477 | 172826 | 176435 |
| m14b      | 26635 | 27066 | 43783 | 44541 | 66645  | 68027  | 99287  | 101551 |
| wave      | 30870 | 31697 | 44087 | 44711 | 64059  | 65772  | 87162  | 88986  |
| 144       | 26032 | 26762 | 38729 | 39568 | 57060  | 58599  | 79441  | 81973  |
| feocean   | 4348  | 4760  | 8495  | 8622  | 14037  | 14277  | 22076  | 22301  |
| ferotor   | 13378 | 13524 | 21199 | 21241 | 32717  | 32783  | 48804  | 49381  |
| bcsstk31  | 13743 | 13812 | 24416 | 24551 | 38361  | 38484  | 60089  | 60724  |
| bcsstk32  | 22717 | 22757 | 38543 | 38711 | 63667  | 63856  | 97495  | 98859  |
| Avg. Impr.% | 2.37 | 1.22 | 1.41 | 1.75 | | | | |
Improving Metis
Postoptimization of Partitions Generated by Metis

<table>
<thead>
<tr>
<th>Graph</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>Avg. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td>1.68</td>
<td>2.82</td>
<td>2.91</td>
<td>3.48</td>
<td>3.34</td>
<td>3.19</td>
</tr>
<tr>
<td>4elt</td>
<td>3.02</td>
<td>3.49</td>
<td>3.50</td>
<td>3.31</td>
<td><strong>3.63</strong></td>
<td>3.41</td>
</tr>
<tr>
<td>cti</td>
<td>2.87</td>
<td>3.50</td>
<td><strong>3.69</strong></td>
<td>2.95</td>
<td>2.92</td>
<td>3.10</td>
</tr>
<tr>
<td>feocean</td>
<td>4.10</td>
<td><strong>7.52</strong></td>
<td>6.00</td>
<td>7.31</td>
<td>4.98</td>
<td>5.41</td>
</tr>
<tr>
<td>fesphere</td>
<td>5.82</td>
<td>4.09</td>
<td><strong>4.77</strong></td>
<td>4.47</td>
<td>3.56</td>
<td>4.32</td>
</tr>
<tr>
<td>fetooth</td>
<td>3.83</td>
<td>3.46</td>
<td><strong>3.96</strong></td>
<td>4.16</td>
<td><strong>3.96</strong></td>
<td>3.80</td>
</tr>
<tr>
<td>wave</td>
<td>2.35</td>
<td>2.56</td>
<td>3.48</td>
<td>4.13</td>
<td><strong>4.52</strong></td>
<td>3.68</td>
</tr>
<tr>
<td>wingnodal</td>
<td>2.29</td>
<td>3.89</td>
<td>3.53</td>
<td><strong>3.87</strong></td>
<td>3.73</td>
<td>3.46</td>
</tr>
<tr>
<td><strong>Avg. Impr. %</strong></td>
<td><strong>3.25</strong></td>
<td><strong>3.92</strong></td>
<td><strong>3.98</strong></td>
<td><strong>4.21</strong></td>
<td><strong>3.83</strong></td>
<td>Avg.: <strong>3.79</strong></td>
</tr>
</tbody>
</table>

- average improvement out of 10 runs in percent - 10 Runs Per Graph and $k$
Adaptively Refined FEM Meshes

What are adaptively refined FEM meshes?
Adaptively Refined Meshes
Postoptimization of Partitions Generated by Metis

<table>
<thead>
<tr>
<th>#Parts</th>
<th>kMETIS</th>
<th>$g = 3$</th>
<th>$g = 6$</th>
<th>$g = 9$</th>
<th>$g = 12$</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6 632</td>
<td>5 673</td>
<td>5 409</td>
<td>5 296</td>
<td>5 233</td>
<td>21.10</td>
</tr>
<tr>
<td>8</td>
<td>11 794</td>
<td>10 707</td>
<td>10 400</td>
<td>10 144</td>
<td>9 953</td>
<td>15.61</td>
</tr>
<tr>
<td>16</td>
<td>19 580</td>
<td>17 568</td>
<td>16 976</td>
<td>16 733</td>
<td>16 550</td>
<td>15.47</td>
</tr>
<tr>
<td>32</td>
<td>31 734</td>
<td>28 582</td>
<td>27 633</td>
<td>27 142</td>
<td>26 745</td>
<td>15.72</td>
</tr>
<tr>
<td>64</td>
<td>47 543</td>
<td>42 844</td>
<td>41 508</td>
<td>40 905</td>
<td>40 580</td>
<td>14.65</td>
</tr>
<tr>
<td>128</td>
<td>69 322</td>
<td>62 589</td>
<td>60 935</td>
<td>60 268</td>
<td>59 935</td>
<td>13.54</td>
</tr>
<tr>
<td>256</td>
<td>100 463</td>
<td>91 307</td>
<td>89 047</td>
<td>88 359</td>
<td>88 023</td>
<td>12.38</td>
</tr>
<tr>
<td>512</td>
<td>144 314</td>
<td>131 201</td>
<td>129 140</td>
<td>128 479</td>
<td>128 140</td>
<td>10.03</td>
</tr>
<tr>
<td>%</td>
<td>-</td>
<td>10.21</td>
<td>12.89</td>
<td>14.14</td>
<td>14.96</td>
<td></td>
</tr>
</tbody>
</table>

- Edge cuts of graph *adaptive*
Matrix Models

Postoptimization of Partitions Generated by Metis

<table>
<thead>
<tr>
<th>Graph</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>afshell10</td>
<td>166,634</td>
<td>179,819</td>
<td>264,229</td>
</tr>
<tr>
<td>audikw1</td>
<td>1,344,561</td>
<td>1,364,308</td>
<td>1,995,353</td>
</tr>
<tr>
<td>ecology1</td>
<td>6,464</td>
<td>7,356</td>
<td>10,516</td>
</tr>
<tr>
<td>kktpower</td>
<td>131,578</td>
<td>135,337</td>
<td>212,664</td>
</tr>
<tr>
<td>ldoor</td>
<td>152,801</td>
<td>157,180</td>
<td>263,290</td>
</tr>
<tr>
<td>nlpkkt120</td>
<td>1,231,012</td>
<td>1,341,214</td>
<td>1,761,856</td>
</tr>
<tr>
<td>thermal2</td>
<td>12,173</td>
<td>12,346</td>
<td>19,900</td>
</tr>
</tbody>
</table>

- Edge cuts of the partitions
- nlpkkt120 (64 parts) the edge cut has been reduced by over 118,000 edges

<table>
<thead>
<tr>
<th>Graph</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>afshell10</td>
<td>7.32</td>
<td>7.57</td>
<td>7.33</td>
<td>6.90</td>
<td>6.29</td>
</tr>
<tr>
<td>audikw1</td>
<td>1.15</td>
<td>1.53</td>
<td>1.44</td>
<td>2.86</td>
<td>3.44</td>
</tr>
<tr>
<td>ecology1</td>
<td>14.68</td>
<td>12.00</td>
<td>12.12</td>
<td>12.30</td>
<td>9.39</td>
</tr>
<tr>
<td>kktpower</td>
<td>3.13</td>
<td>1.26</td>
<td>2.77</td>
<td>2.56</td>
<td>3.47</td>
</tr>
<tr>
<td>ldoor</td>
<td>3.45</td>
<td>4.62</td>
<td>2.78</td>
<td>3.46</td>
<td>4.32</td>
</tr>
<tr>
<td>nlpkkt120</td>
<td>9.29</td>
<td>6.21</td>
<td>8.21</td>
<td>5.80</td>
<td>4.72</td>
</tr>
<tr>
<td>thermal2</td>
<td>0.94</td>
<td>1.20</td>
<td>1.40</td>
<td>1.79</td>
<td>2.35</td>
</tr>
</tbody>
</table>

- Improvement of partitions
Outline

Motivation
  The Problem
  Motivation

The Algorithm
  Multilevel Approach
  Our Approach - The Algorithm

Current Results
  Quality
  Runtime

Future Work
Scalability
Scalability

<table>
<thead>
<tr>
<th>$p = k$</th>
<th>$t_{ref}$</th>
<th>$\Theta_p$</th>
<th>Work</th>
<th>$S$</th>
<th>$t_{k\text{MetisLastRef}}$</th>
<th>$t_{p\text{METIS}}$</th>
<th>$t_{k\text{METIS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>12.534</td>
<td>3.0</td>
<td>37.45</td>
<td>2.98</td>
<td>0.524</td>
<td>15.48</td>
<td>9.73</td>
</tr>
<tr>
<td>8</td>
<td>8.377</td>
<td>6.5</td>
<td>50.59</td>
<td>6.03</td>
<td>0.528</td>
<td>22.39</td>
<td>9.75</td>
</tr>
<tr>
<td>16</td>
<td>5.356</td>
<td>12.0</td>
<td>51.54</td>
<td>9.62</td>
<td>0.572</td>
<td>28.19</td>
<td>9.87</td>
</tr>
<tr>
<td>32</td>
<td>3.603</td>
<td>18.8</td>
<td>54.88</td>
<td>15.23</td>
<td>0.634</td>
<td>33.49</td>
<td>10.01</td>
</tr>
<tr>
<td>64</td>
<td>2.023</td>
<td>43.4</td>
<td>64.72</td>
<td>31.99</td>
<td>0.747</td>
<td>38.19</td>
<td>10.26</td>
</tr>
<tr>
<td>128</td>
<td>1.166</td>
<td>91.7</td>
<td>76.24</td>
<td>68.32</td>
<td>0.891</td>
<td>42.22</td>
<td>10.85</td>
</tr>
<tr>
<td>256</td>
<td>0.796</td>
<td>195.7</td>
<td>96.17</td>
<td>120.81</td>
<td>1.102</td>
<td>44.93</td>
<td>11.31</td>
</tr>
<tr>
<td>512</td>
<td>0.551</td>
<td>405.1</td>
<td>127.79</td>
<td>231.92</td>
<td>1.383</td>
<td>48.24</td>
<td>12.08</td>
</tr>
</tbody>
</table>

Tabelle: Results for *adaptive* (good quality configuration).
Scalability / Not Enough Work

<table>
<thead>
<tr>
<th>$p = k$</th>
<th>$t_{ref}$</th>
<th>$\Theta_p$</th>
<th>Work</th>
<th>$S$</th>
<th>$t_{kMetisLastRef}$</th>
<th>$t_{pMETIS}$</th>
<th>$t_{kMETIS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1.762</td>
<td>3.0</td>
<td>6.51</td>
<td>3.69</td>
<td>0.524</td>
<td>15.48</td>
<td>9.73</td>
</tr>
<tr>
<td>8</td>
<td>1.344</td>
<td>6.5</td>
<td>8.09</td>
<td>6.01</td>
<td>0.528</td>
<td>22.39</td>
<td>9.75</td>
</tr>
<tr>
<td>16</td>
<td>0.700</td>
<td>12.0</td>
<td>8.44</td>
<td>12.05</td>
<td>0.572</td>
<td>28.19</td>
<td>9.87</td>
</tr>
<tr>
<td>32</td>
<td>0.443</td>
<td>18.8</td>
<td>7.48</td>
<td>16.88</td>
<td>0.634</td>
<td>33.49</td>
<td>10.01</td>
</tr>
<tr>
<td>64</td>
<td>0.238</td>
<td>43.4</td>
<td>7.12</td>
<td>29.92</td>
<td>0.747</td>
<td>38.19</td>
<td>10.26</td>
</tr>
<tr>
<td>128</td>
<td>0.147</td>
<td>91.7</td>
<td>7.24</td>
<td>49.25</td>
<td>0.891</td>
<td>42.22</td>
<td>10.85</td>
</tr>
<tr>
<td>256</td>
<td>0.104</td>
<td>195.7</td>
<td>7.84</td>
<td>75.38</td>
<td>1.102</td>
<td>44.93</td>
<td>11.31</td>
</tr>
<tr>
<td>512</td>
<td>0.076</td>
<td>405.1</td>
<td>11.09</td>
<td>145.92</td>
<td>1.383</td>
<td>48.24</td>
<td>12.08</td>
</tr>
</tbody>
</table>

Tabelle: Results for adaptive (slight quality configuration).
Current Use Case

Recommendation

- we implemented an METIS like Interface
  1. use ParMetis (with lower quality than sequential METIS)
  2. use our post optimization to obtain good edge cut
- or wait on our full implementation
Future Work

- try other pairwise improvers, e.g. flow or diffusion based improvers
- back to sequential graph partitioning - try the best spectral bisection algorithms also in a pairwise fashion on a $k$-partition
- implement the second variant of boundary exchange; initialize the BFS with $\partial^j V_i$
Future Work

- different quotient graph structure (threshold), insert edges iff the number of edges pointing to two different partitions $\{P_i, P_j\}$ is above a threshold $\alpha$ (for pathological cases)
- fourth queue selection strategy for FM algorithm (highest gain selection)
- build an asynchronous communication protocol with the original maximal matching algorithm of Israeli and Itai (sparse subgraph)
Future Work

1. parallelization of the contraction phase (Manuel Holtgrewe)
2. implement projection
3. refine each level parallel
4. migration of nodes?
Thank You!
Algorithm 4 Conjugate Gradient Method

**Data**: Initial choice of $x_0 \in \mathbb{R}^n$

$r_0 := b - Ax_0$

$p_0 := r_0$

$k := 0$

**while** true **do**

$$z^k = A x^k$$

$$\alpha_k := \frac{r_k^\top r_k}{p_k^\top z_k}$$

$$x^{k+1} := x^k + \alpha_k p_k$$

$$r_{k+1} := r_k - \alpha_k z^k$$

**if** $r^{k+1}$ small **then** exit

$$\beta_k := \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$$

$$p_{k+1} := r_{k+1} + \beta_k p_k$$

$k := k + 1$

**end while**
Matrix Model for Computation
Sparse-Matrix Dense-Vector Multiplication

1. A symmetric and positive definite
2. $G = (V, E)$ be a undirected graph ($V = \{1, \ldots, n\}$ and $e = \{i, j\} \in E \iff A_{i,j} \neq 0$
3. each component $x_i, z_i, A_i,\ast$ corresponds to node $i \in V$
4. graph undirected because $A$ is symmetric
5. processor $l$, corresponding to part $V_l$ of $\mathcal{P}$, stores $x_i, z_i, A_i,\ast$, computes $z_i$ for all $i \in V_l$.
6. A good partition of $G$ equals balanced work and minimized communication.
**Algorithm 5** Fiduccia-Mattheyses Refinement Algorithm

**Data**: $G = (V, E)$, initial bisection $\{V_1, V_2\}$

**forall** $v \in V$ do compute gain $g(v)$

repeat

**forall** $v \in \partial V_1$ random do put $g(v)$ into binary heap $\mathcal{H}_1$

**forall** $v \in \partial V_2$ random do put $g(v)$ into binary heap $\mathcal{H}_2$

ordered list $L \leftarrow \emptyset$

unmark all nodes $v \in V$

while $\exists$ unmarked node do

select heap $\mathcal{H}_i$

$v_{\text{maxgain}} \leftarrow \text{maxgain node in } \mathcal{H}_i$

move $v_{\text{maxgain}}$ to other partition and append it to $L$

update $g(v_{\text{maxgain}})$ in $\mathcal{H}_i$

**forall** $v \in N(v_{\text{maxgain}})$ do

update $g$-value

if $v$ new boundary node then insert $g(v)$ into $\mathcal{H}_i$

if $v$ was and is not a boundary node anymore then

remove $g(v)$ from $\mathcal{H}_i$

end if

end if

$j \leftarrow \arg\max_k \tilde{\phi}(k)$

$\gamma \leftarrow \tilde{\phi}(j)$

if $\gamma > 0$ then apply changes

end if

end while

until $\gamma \leq 0$;