High Quality Graph Partitioning

Peter Sanders, Christian Schulz
Algorithm Engineering Lecture WS 12/13

Institute for Theoretical Computer Science
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
Mesh partitioned
1. nodes ↔ data, computation
2. edges ↔ interdependencies

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.
\( \varepsilon \)-Balanced Graph Partitioning

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

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

Applications:
linear equation systems, VLSI design, route planning, \ldots
Overview

- Introduction
- Multilevel Algorithms
- Advanced Techniques
- Evolutionary Techniques
- Experiments
- Summary
Multi-Level Graph Partitioning

Successful in existing algorithms:
Metis, Scotch, DiBaP, . . ., KaP Pa, KaSP ar, KaFF P a, KaFF P aE
Advanced Techniques

Talk Today

- GP Folklore
- Edge Ratings
- Flow Based Refinements
- More Localized Local Search
- Iterated Multilevel
But how are the edges selected?
Graph Partitioning
Matching Selection

Goals:
1. large \textit{edge weights} $\leadsto$ sparsify
2. large \#\textit{edges} $\leadsto$ few levels
3. uniform \textit{node weights} $\leadsto$ “represent” input
4. small node \textit{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 \textit{edge rating} function
Graph Partitioning

Edge Ratings

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

expansion* (\{u, v\}) := \frac{\omega(\{u, v\})}{c(u)c(v)}

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

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

where \(c = \text{node weight}, \ \omega = \text{edge weight},\)
\(Out(u) := \sum_{\{u, v\} \in E} \omega(\{u, v\})\)
Global Paths Algorithm

Approx. Weighted Matching

[Maue Sanders 2007]

- Sort edges according to their weight
- Grow a set of paths and even-length cycles
- Find optimum matching for every path and cycle
- Running time $O(m + \text{sort}(m))$
- Approximation $\frac{1}{2}\text{OPT}$
- outperforms HEM, SHEM
Initial Partitioning
Graph Partitioning

Initial Partitioning - One Possibility

- bipartition algorithm $\Rightarrow$ $k$-way partition algorithm
- \textbf{BIPART}(G):
  1. search for pseudo-peripheral nodes
  2. perform two-sided BFS
  3. post improvement using local search
- repeat (few times) and take best
Graph Partitioning

Initial Partitioning - One Possibility

- bipartition algorithm $\Rightarrow k$-way partition algorithm
- $\text{BIPART}(G)$:
  1. search for pseudo-peripheral nodes
  2. perform two-sided BFS
  3. post improvement using local search
- repeat (few times) and take best
Graph Partitioning
Initial Partitioning - One Possibility

- bipartition algorithm $\Rightarrow$ $k$-way partition algorithm
- BIPART($G$):
  1. search for pseudo-peripheral nodes
  2. perform two-sided BFS
  3. post improvement using local search
- repeat (few times) and take best
Graph Partitioning

Initial Partitioning - One Possibility

- bipartition algorithm $\Rightarrow k$-way partition algorithm
- \textbf{BIPART}(G):
  1. search for pseudo-peripheral nodes
  2. perform two-sided BFS
  3. post improvement using local search
- repeat (few times) and take best
Initial Partitioning

- we currently use Scotch [Pellegrini]
- multiple tries pay off

Open Problem:
Direct $k$-partitioner that achieves better quality or speed.
Local Improvement

input graph

match

contract

initial partitioning

local improvement

output partition

contract uncontract
FM Local Search
compute gain $\forall v \in V$
FM Local Search

- compute $\text{gain} \ \forall \ v \in V$
- $g(v) = d_{ext}(v) - d_{int}(v)$
FM Local Search

- compute gain $\forall \ v \in V$
- $g(v) = d_{ext}(v) - d_{int}(v)$
- store gain of boundary nodes (e.g. in a heap)
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

Step: 0
Edge Cut: 5
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

Step: 0 1
Edge Cut: 5 4
FM Local Search

- move **highest gain** vertices to opposite block
- each node at most **once**
- update gain of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>Edge Cut:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain 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>
FM Local Search

- stop after limit
- take best edge cut
- within balance constraint

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

edge cut

steps
FM Local Search

- stop after limit
- take best edge cut
- within balance constraint

<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>

Edge cut vs steps
FM Local Search – Discussion

+ Generalizable for multiple blocks
+ Linear time
- Unlikely to find improvements requiring $\geq 2$ negative gain moves
More Localized Local Search
More Localized Local Search

inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in \text{rnd} \ T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
More Localized Local Search
inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in \text{rnd} \ T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in \text{rnd} \ T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in \text{rnd } T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in \text{rnd} \ T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with **complete boundary**
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved **at most once**
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in \text{rand } T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in \text{rnd } T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
Flows as Local Improvement
Flows as Local Improvement

Two Blocks

- area $B$, such that each $(s, t)$-min cut is $\epsilon$-balanced cut in $G$
- e.g. 2 times BFS (left, right)
Flows as Local Improvement

Two Blocks

- obtain **optimal cut** in $B$
- since each cut in $B$ yields a feasible partition
  $\rightarrow$ **improved two-partition**
- **advanced techniques** possible and necessary
Flows as Local Improvement

Adaptive Search

- search in larger areas for feasible cuts
- adaptively control the size of corridor $B$
- heuristic for Most Balanced Minimum Cuts [Picard et al. 1980]
- we need: SCC’s, DAGs, Closed Vertex Sets, Topological Sorting

- the maximal upper bound factor is called $\alpha'$
Example

100x100 Grid
Example

Constructed Flow Problem (using BFS)
Example
Apply Max-Flow Min-Cut
Example
Output Improved Partition
Local Improvement for $k$-partitions
Using Flows?

on each pair of blocks

input graph

match

contract

initial partitioning

local improvement

uncontract

output partition

input graph

...
Local Search or Flows?

<table>
<thead>
<tr>
<th>Local Search</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>global</td>
</tr>
<tr>
<td>multiway</td>
<td>two way</td>
</tr>
<tr>
<td>any $\epsilon$</td>
<td>large $\epsilon$</td>
</tr>
<tr>
<td>handicapped for $\epsilon \approx 0$</td>
<td></td>
</tr>
</tbody>
</table>

$\Rightarrow$ Combination works best

**Current Work:** Really powerful technique for $\epsilon \approx 0$
Iterated Multilevel
Iterated Multilevel [Walshaw 2004]

- don’t contract cut edges
- adapt previous solution as initial partitioning
- cuts can only improve
- V-cycles / F-cycles
Global Search

V-Cycles

Coarsening

Uncoarsening

Graph not partitioned

Graph partitioned

○ Graph partitioned

● Graph not partitioned
Global Search
W-Cycles

Coarsening

Uncoarsening

Graph partitioned

Graph not partitioned
Global Search
F-Cycles

Coarsening

Uncoarsening

- Graph partitioned
- Graph not partitioned
Experiments
# Experiments

## Testset

<table>
<thead>
<tr>
<th>graph</th>
<th>$n$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgg17</td>
<td>$2^{17}$</td>
<td>1 457 506</td>
</tr>
<tr>
<td>rgg18</td>
<td>$2^{18}$</td>
<td>3 094 566</td>
</tr>
<tr>
<td>Delaunay17</td>
<td>$2^{17}$</td>
<td>786 352</td>
</tr>
<tr>
<td>Delaunay18</td>
<td>$2^{18}$</td>
<td>1 572 792</td>
</tr>
<tr>
<td>bcsstk29</td>
<td>13 992</td>
<td>605 496</td>
</tr>
<tr>
<td>4elt</td>
<td>15 606</td>
<td>91 756</td>
</tr>
<tr>
<td>fesphere</td>
<td>16 386</td>
<td>98 304</td>
</tr>
<tr>
<td>cti</td>
<td>16 840</td>
<td>96 464</td>
</tr>
<tr>
<td>memplus</td>
<td>17 758</td>
<td>108 384</td>
</tr>
<tr>
<td>cs4</td>
<td>33 499</td>
<td>87 716</td>
</tr>
<tr>
<td>pwt</td>
<td>36 519</td>
<td>289 588</td>
</tr>
<tr>
<td>bcsstk32</td>
<td>44 609</td>
<td>1 970 092</td>
</tr>
<tr>
<td>body</td>
<td>45 087</td>
<td>327 468</td>
</tr>
<tr>
<td>t60k</td>
<td>60 005</td>
<td>178 880</td>
</tr>
<tr>
<td>wing</td>
<td>62 032</td>
<td>243 088</td>
</tr>
<tr>
<td>brack2</td>
<td>62 631</td>
<td>733 118</td>
</tr>
<tr>
<td>finan512</td>
<td>74 752</td>
<td>522 240</td>
</tr>
<tr>
<td>bel</td>
<td>463 514</td>
<td>1 183 764</td>
</tr>
<tr>
<td>nld</td>
<td>893 041</td>
<td>2 279 080</td>
</tr>
<tr>
<td>af_shell9</td>
<td>504 855</td>
<td>17 084 020</td>
</tr>
</tbody>
</table>
# Experimental Evaluation

## Flows

<table>
<thead>
<tr>
<th>Var.</th>
<th>(+F, -MB, -FM)</th>
<th>(+F, +MB, -FM)</th>
<th>(+F, -MB, +FM)</th>
<th>(+F, +MB, +FM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha'$</td>
<td>$\Delta$ cut %</td>
<td>$t[s]$</td>
<td>$\Delta$ cut %</td>
<td>$t[s]$</td>
</tr>
<tr>
<td>16</td>
<td>-1.88</td>
<td>4.17</td>
<td>0.81</td>
<td>3.92</td>
</tr>
<tr>
<td>8</td>
<td>-2.30</td>
<td>2.11</td>
<td>0.41</td>
<td>2.07</td>
</tr>
<tr>
<td>4</td>
<td>-4.86</td>
<td>1.24</td>
<td>-2.20</td>
<td>1.29</td>
</tr>
<tr>
<td>2</td>
<td>-11.86</td>
<td>0.90</td>
<td>-9.16</td>
<td>0.96</td>
</tr>
<tr>
<td>1</td>
<td>-19.58</td>
<td>0.76</td>
<td>-17.09</td>
<td>0.80</td>
</tr>
<tr>
<td>Ref.</td>
<td>(-F, -MB, +FM)</td>
<td>2974</td>
<td>1.13</td>
<td></td>
</tr>
</tbody>
</table>

- final score of different configurations
- $\alpha'$ flow region upper bound factor
- all value are improvements rel. to Ref.

$\leftrightarrow$ +/− F ↔ +/− Flow
$\leftrightarrow$ +/− MB ↔ +/− Most Bal. H.
$\leftrightarrow$ +/− FM ↔ +/− FM Algorithm
## Experimental Evaluation

### Flows - Effectiveness

<table>
<thead>
<tr>
<th>Effectiveness</th>
<th>(+F, +MB, -FM)</th>
<th>(+F, -MB, +FM)</th>
<th>(+F, +MB, +FM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha' )</td>
<td>( \Delta \text{ cut } % )</td>
<td>( \Delta \text{ cut } % )</td>
<td>( \Delta \text{ cut } % )</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>3.70</td>
<td>4.28</td>
</tr>
<tr>
<td>8</td>
<td>1.12</td>
<td>4.16</td>
<td>4.74</td>
</tr>
<tr>
<td>4</td>
<td>3.05</td>
<td>4.04</td>
<td>4.63</td>
</tr>
<tr>
<td>2</td>
<td>8.26</td>
<td>3.02</td>
<td>3.36</td>
</tr>
<tr>
<td>1</td>
<td>16.41</td>
<td>1.62</td>
<td>1.65</td>
</tr>
<tr>
<td>(-F, -MB, +FM)</td>
<td>2833</td>
<td>2831</td>
<td>2827</td>
</tr>
</tbody>
</table>

- each configuration has the **same amount of time**
- all values are improvements relative to **Ref.**

<table>
<thead>
<tr>
<th>+/- F</th>
<th>( \leftrightarrow )</th>
<th>+/- Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/- MB</td>
<td>( \leftrightarrow )</td>
<td>+/- Most Bal. H.</td>
</tr>
<tr>
<td>+/- FM</td>
<td>( \leftrightarrow )</td>
<td>+/- FM Algorithm</td>
</tr>
</tbody>
</table>
## Experimental Evaluation

### Global Search

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\Delta$ cut %</th>
<th>$t$ [s]</th>
<th>Eff. Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 F-cycle</td>
<td>2.69</td>
<td>2.31</td>
<td>2806</td>
</tr>
<tr>
<td>3 V-cycle</td>
<td>2.69</td>
<td>2.49</td>
<td>2810</td>
</tr>
<tr>
<td>2 W-cycle</td>
<td>2.91</td>
<td>2.77</td>
<td>2810</td>
</tr>
<tr>
<td>1 W-cycle</td>
<td>1.33</td>
<td>1.38</td>
<td>2815</td>
</tr>
<tr>
<td>1 F-cycle</td>
<td>1.09</td>
<td>1.18</td>
<td>2816</td>
</tr>
<tr>
<td>2 V-cycle</td>
<td>1.88</td>
<td>1.67</td>
<td>2817</td>
</tr>
<tr>
<td>1 V-cycle</td>
<td>2973</td>
<td>0.85</td>
<td>2834</td>
</tr>
</tbody>
</table>

- relative fast basic configuration
- only global search is varied
- red significantly < blue
- red and blue significantly < reference
## Experiments

### Remove Components

- Remove components *step by step*

<table>
<thead>
<tr>
<th>Repetition</th>
<th>Avg.</th>
<th>t</th>
<th>Eff. Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaFFPa Strong</td>
<td>2 683</td>
<td>8.93</td>
<td>2 636</td>
</tr>
<tr>
<td>-KWay</td>
<td>2 682</td>
<td>9.23</td>
<td>2 636</td>
</tr>
<tr>
<td>-MoreLocalizedSearch</td>
<td>2 729</td>
<td>5.55</td>
<td>2 668</td>
</tr>
<tr>
<td>-FCycle</td>
<td>2 748</td>
<td>3.27</td>
<td>2 669</td>
</tr>
<tr>
<td>-Flow</td>
<td>2 934</td>
<td>1.66</td>
<td>2 799</td>
</tr>
</tbody>
</table>
Experimental Results
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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Avg.</td>
<td>t[s]</td>
</tr>
<tr>
<td>KaFFPa strong</td>
<td>12 053</td>
<td>12 182</td>
<td>121.22</td>
</tr>
<tr>
<td>KaSPar strong</td>
<td>12 450</td>
<td>+3%</td>
<td>87.12</td>
</tr>
<tr>
<td>KaFFPa eco</td>
<td>12 763</td>
<td>+6%</td>
<td>3.82</td>
</tr>
<tr>
<td>Scotch</td>
<td>14 218</td>
<td>+20%</td>
<td>3.55</td>
</tr>
<tr>
<td>KaFFa fast</td>
<td>15 124</td>
<td>+24%</td>
<td>0.98</td>
</tr>
<tr>
<td>kMetis</td>
<td>15 167</td>
<td>+33%</td>
<td>0.83</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
Evolutionary Techniques
Distributed Evolutionary Graph Partitioning

- **Evolutionary Algorithms:**
  - highly inspired by biology
  - population of individuals
  - selection (based on fitness), mutation, recombination, ...

- **Goal:** Integrate KaFFPa in an Evolutionary Strategy

- **Evolutionary Graph Partitioning:**
  - individuals $\leftrightarrow$ partitions
  - fitness $\leftrightarrow$ edge cut
procedure steady-state-EA
create initial population \( \mathcal{P} \)
while stopping criterion not fulfilled
  select parents \( \mathcal{P}_1, \mathcal{P}_2 \) from \( \mathcal{P} \)
  combine \( \mathcal{P}_1 \) with \( \mathcal{P}_2 \) to create offspring \( o \)
  mutate offspring \( o \)
  evict individual in population using \( o \)
return the fittest individual that occurred
two individuals $\mathcal{P}_1$, $\mathcal{P}_2$:
don’t contract cut edges of $\mathcal{P}_1$ or $\mathcal{P}_2$
until no matchable edge is left
coarsest graph $\leftrightarrow$ Q-graph of overlay
exchanging good parts is easy
initial solution: use better of both parents
Example
Two Individuals $\mathcal{P}_1, \mathcal{P}_2$
Example
Multilevel Combine of $P_1, P_2$
Exchanging good parts is easy

Coarsest Level

- >> large weight, < small weight
- start with the better partition (red, $P_2$)
- move $v_4$ to the opposite block
- integrated into multilevel scheme (+local search on each level)
Example
Result of $P_1, P_2$
KaFFPaE
Recombination - Generalization

- recombine a partition with any clustering of the graph e.g. :
- $k' \neq k$ partition with larger imbalances
- natural cuts: sparse cuts close to dense areas [Delling et al. '11]

- plug and play: use the clustering that fits your domain
Parallelization

- each PE has its own island (a local population)
- locally: perform combine and mutation operations
- communicate analog to randomized rumor spreading
  1. rumor ↔ currently best local partition
  2. local best partition changed → send it to $O(\log P)$ random PEs
  3. asynchronous communication (MPI Isend)
→ quality records in a few minutes for small graphs
More Experiments
Example

Street network Europe $|V| = 18M$, $|E| = 44M$, $k = 64$

Buffoon $\leftrightarrow$ kMetis
Quality
Evolutionary Graph Partitioning

<table>
<thead>
<tr>
<th>blocks $k$</th>
<th>KaFFPaE improvement over reps. of KaFFPa</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.2%</td>
</tr>
<tr>
<td>4</td>
<td>1.0%</td>
</tr>
<tr>
<td>8</td>
<td>1.5%</td>
</tr>
<tr>
<td>16</td>
<td>2.7%</td>
</tr>
<tr>
<td>32</td>
<td>3.4%</td>
</tr>
<tr>
<td>64</td>
<td>3.3%</td>
</tr>
<tr>
<td>128</td>
<td>3.9%</td>
</tr>
<tr>
<td>256</td>
<td>3.7%</td>
</tr>
<tr>
<td>overall</td>
<td>2.5%</td>
</tr>
</tbody>
</table>

2h time, 32 cores per graph and $k$, geom. mean
KaFFPaEvolutionary $k=64$

- Mean min cut
- Normalized time $t_n$

- Repetitions
- KaFFPaE

Repetitions
KaFFPaE
Scalability

```
mean min cut (mmc)

p = 1
p = 2
p = 4
p = 8
p = 16
p = 32
p = 64
p = 128
p = 256

normalized time $t_n$

1 10 100 1000 10000
2450 2500 2550 2600
```
Scalability

$p$ up to 256 PEs

- $t_p = 15360 / p$ seconds per instance
- pseudo speedup $S_p(t_n) = c'_1(t_n) / c'_p(t_n)$
- $c'_i(t_n) = \min_{c_i(t') \leq c_1(t_n)} t'$
Scalability

pseudo speedup

normalized time $t_n$

$p = 2$
$p = 4$
$p = 8$
$p = 16$
$p = 32$
$p = 64$
$p = 128$
$p = 256$
Walshaw Benchmark

- 816 instances ($\epsilon \in \{0, 1\%, 3\%, 5\%\}$)
- focus on partition quality

overall quality records $\leq$:

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$\leq$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>78%</td>
</tr>
<tr>
<td>1%</td>
<td>78%</td>
</tr>
<tr>
<td>3%</td>
<td>92%</td>
</tr>
<tr>
<td>5%</td>
<td>94%</td>
</tr>
</tbody>
</table>
Summary

Distributed evol. Alg. [ALENEX12]

Input graph

flows etc. [ESA11]

Partitioning

Uncontract

Output Partition

Parallel [IPDPS10]
n-level [ESA10]

Cycles a la multigrid

Multilevel Graphpartitioning

Initial partitioning

Todo

Local improvement
Current and Future Work

- $\epsilon = / \approx 0$
- open source release
- back to parallelization (+ external?)
- huge $k$
- reconsider $n$-level? (flows?, . . . )
- other objective functions ((max.) communication volume, separators,. . . )
- hypergraph partitioning
- clustering
- mapping onto processors
- other multilevel applications (e.g., graph drawing)
- close gap to theory?
- etc.
Thank you!

Contact: christian.schulz@kit.edu