Shape Optimizing Load Balancing for Parallel Adaptive Numerical Simulations Using MPI

Henning Meyerhenke

Institute of Theoretical Informatics
Karlsruhe Institute of Technology (KIT)

10th DIMACS Challenge Workshop, Feb 13-14, 2012, Atlanta

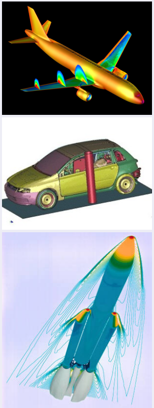
Load Balancing by Repartitioning

- **Application**: Large adaptive numerical simulations on parallel computers
- **Task**: Mapping of mesh (discretization) to processors
- **Objective:** Efficient parallel solution of linear systems (discretized PDEs)

$\Rightarrow$ (Re)Partition mesh (or dual graph) such that:

- the load is balanced and
- application communication is minimized

Meshes



Load Balancing by Repartitioning

- **Application**: Large adaptive numerical simulations on parallel computers
- **Task**: Mapping of mesh (discretization) to processors
- **Objective:** Efficient parallel solution of linear systems (discretized PDEs)

⇒ (Re)Partition mesh (or dual graph) such that:

- the load is balanced and
- application communication is minimized

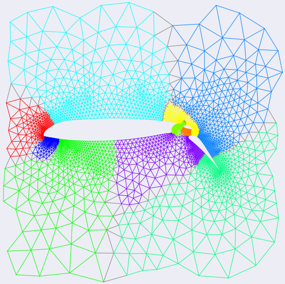
Load Balancing by Repartitioning

- **Application**: Large adaptive numerical simulations on parallel computers
- **Task**: Mapping of mesh (discretization) to processors
- **Objective:** Efficient parallel solution of linear systems (discretized PDEs)

⇒ (Re)Partition mesh (or dual graph) such that:

- the load is balanced and
- application communication is minimized

10-way partition



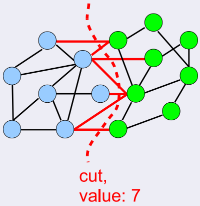
Graph Partitioning and Repartitioning

Static Case: Traditional Graph Partitioning Problem (GPP)

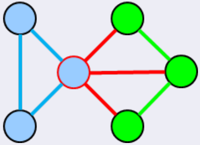
Given a graph $G = (V, E, \omega)$, partition V into $V = \pi_1 \dot{\cup} \ldots \dot{\cup} \pi_k$ by a mapping $\Pi : V \to \{1, \ldots, k\}$ such that

- Π is balanced $(|\pi_1| \approx \cdots \approx |\pi_k|)$,
- the weight of the cut edges $\sum_{\{u,v\} \in E : \Pi(u) \neq \Pi(v)} \omega(u, v)$ is minimized

Edge Cut



Communication



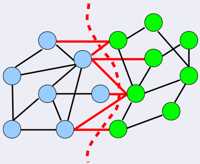
Graph Partitioning and Repartitioning

Static Case: Traditional Graph Partitioning Problem (GPP)

Given a graph $G = (V, E, \omega)$, partition V into $V = \pi_1 \dot\cup \ldots \dot\cup \pi_k$ by a mapping $\Pi : V \to \{1, \ldots, k\}$ such that

- Π is balanced ($|\pi_1| \approx \cdots \approx |\pi_k|$),
- the weight of the cut edges $\sum_{\{u,v\} \in E : \Pi(u) \neq \Pi(v)} \omega(u, v)$ is minimized

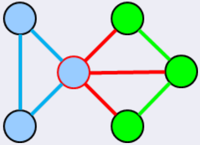
Edge Cut



Dynamic Case: Repartitioning Problem

Solve the GPP with an additional objective: Minimum migration costs

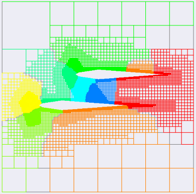
Communication



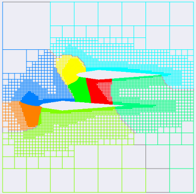
$\mathcal{NP}$-hard $\to$ heuristics in practice

State of the Art and its Limitations

METIS



DibaP

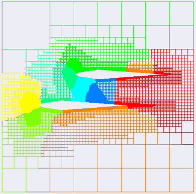


Fast libraries for graph repartitioning:

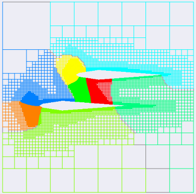
- Use heuristics such as Kernighan-Lin (KL) within multilevel process
- KL focusses too strongly on edge-cut $\Rightarrow$ Partitions are often disconnected and not well shaped
- KL is inherently sequential (but fast parallel variations exist, e. g., ParMETIS, Parallel JOSTLE, KaPPa, Zoltan)

State of the Art and its Limitations

METIS



DibaP



Fast libraries for graph repartitioning:

- Use heuristics such as Kernighan-Lin (KL) within multilevel process
- KL focusses too strongly on edge-cut $\Rightarrow$ Partitions are often disconnected and not well shaped
- KL is inherently sequential (but fast parallel variations exist, e. g., ParMETIS, Parallel JOSTLE, KaPPa, Zoltan)

Additionally, partitions should

- have few boundary vertices,
- have a low diameter and be connected,
- induce low migration costs.

Synchronous computations: Maximum norm

Outline

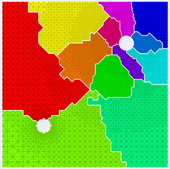
Our Approach
Diffusive Graph Partitioning with good Partition Shapes

Idea 1: Compute good partition shapes with small surfaces!

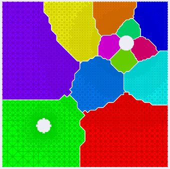
Idea 2: Diffusive process decides *which* elements go where!

- Small partition diameters
- Few cut edges
- Short partition boundaries
- Connected partitions more often
- Small migration costs in case of repartitioning
- Higher, but reasonable running time

Metis (KL)

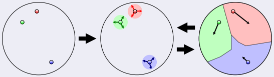


Shape Optimized

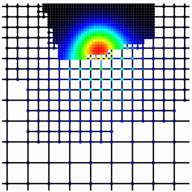


Shape Optimization with Bubble-FOS/C

Bubble framework: `Init`, `AssignPartition` und `ComputeCenters`



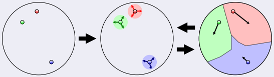
Basic idea: Lloyd's k-means, [Walshaw et al., IJSA'95], [Diekmann et al., J. ParCo'00]



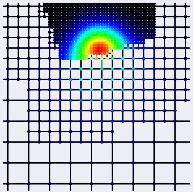
- Similarity measure: Reflect how well connected two nodes are
- Diffusion: Desire of a substance to distribute itself in space
- Substance = loads
- Related to random walks: $w^{(i+1)} = \mathbf{M}w^{(i)}$
- Exploit: Diffusion spreads load faster into densely connected graph regions
- FOS/C: Disturbed variant of first order diffusion FOS to avoid balancing property

Shape Optimization with Bubble-FOS/C

Bubble framework: `Init`, `AssignPartition` und `ComputeCenters`



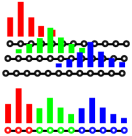
Basic idea: Lloyd's k-means, [Walshaw et al., IJSA'95], [Diekmann et al., J. ParCo'00]



- Similarity measure: Reflect how well connected two nodes are
- Diffusion: Desire of a substance to distribute itself in space
- Substance = loads
- Related to random walks: $w^{(i+1)} = \mathbf{M}w^{(i)}$
- Exploit: Diffusion spreads load faster into densely connected graph regions
- FOS/C: Disturbed variant of first order diffusion FOS to avoid balancing property

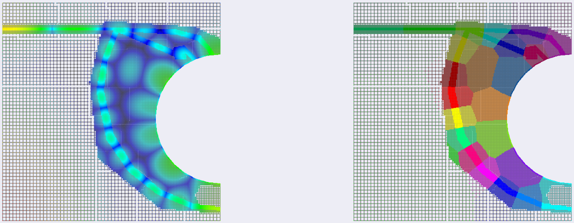
The Operation AssignPartition
[M., Monien, Schamberger, J. ParCo'09]

- Input: Centers. Output: Partition
- For each partition p: FOS/C procedure, disturbance by drain vector d_p, center z_p as so-called source vertex



- Linear system (LS) for π_p: $\mathbf{L}w_p = d_p$
- Assignment of vertex v to part p: $\Pi(v) = \text{argmax}_{1 \leq p \leq k}[w_p]_v$
- Balancing by ScaleBalance

AssignPartition: Maximal load, resulting partition



Optimization Criterion

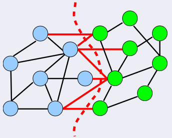
Minimum Balanced Cut (here: $k = 2$, can be extended)

- Quadratic optimization problem for min balanced cut:

$$\min_{y \in \{-1,1\}^n} y^T \mathbf{L} y \quad \text{s. t.} \quad y^T \mathbf{1} = 0$$

- Spectral partitioning: Relax integrality constraint and solve eigenvector problem

Bisection



Theorem [M., ISAAC'10]:

Under mild conditions, `AssignPartition` followed by `ScaleBalance` together compute the global minimum of

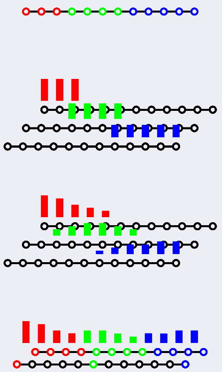
$$\min_{y \in \mathbb{R}^n} \quad y^T \mathbf{L} y$$
$$s.\,t. \quad y^T \cdot d = 2\delta n(\beta_1 + \beta_2)$$

Faster Local Approach TruncCons

Truncated Diffusion Consolidations [M., Monien, Sauerwald, JPDC'09]

- Consolidation: $\Pi \to \Pi$
- Same initial load for nodes of current subdomain, all others 0
- Use a small number ψ (e. g. , $\psi = 14$) of FOS iterations to distribute load
- Exploits: Dense region has many internal short paths
- Inactive nodes: Nodes that have the same load as all their neighbors
- Omit inactive nodes $\Rightarrow$ Computational work only in boundary regions
- Assignment to subdomains as before, consolidation is repeated Λ times (e. g. , $\Lambda = 10$)
- More iterations (Λ / ψ) $\rightsquigarrow$ Better quality, higher running time

Consolidation

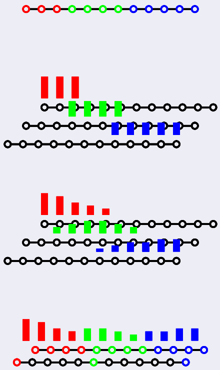


Faster Local Approach TruncCons

Truncated Diffusion Consolidations [M., Monien, Sauerwald, JPDC'09]

- Consolidation: $\Pi \to \Pi$
- Same initial load for nodes of current subdomain, all others 0
- Use a small number ψ (e. g. , $\psi = 14$) of FOS iterations to distribute load
- Exploits: Dense region has many internal short paths
- Inactive nodes: Nodes that have the same load as all their neighbors
- Omit inactive nodes $\Rightarrow$ Computational work only in boundary regions
- Assignment to subdomains as before, consolidation is repeated Λ times (e. g. , $\Lambda = 10$)
- More iterations (Λ / ψ) $\rightsquigarrow$ Better quality, higher running time

Consolidation

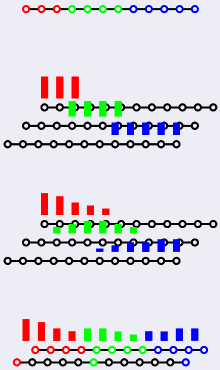


Faster Local Approach TruncCons

Truncated Diffusion Consolidations [M., Monien, Sauerwald, JPDC'09]

- Consolidation: $\Pi \to \Pi$
- Same initial load for nodes of current subdomain, all others 0
- Use a small number ψ (e. g. , $\psi = 14$) of FOS iterations to distribute load
- Exploits: Dense region has many internal short paths
- Inactive nodes: Nodes that have the same load as all their neighbors
- Omit inactive nodes $\Rightarrow$ Computational work only in boundary regions

Consolidation



- Assignment to subdomains as before, consolidation is repeated Λ times (e. g. , $\Lambda = 10$)
- More iterations (Λ / ψ) $\rightsquigarrow$ Better quality, higher running time

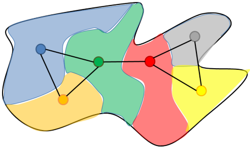
Flow-based Balancing with Diffusion

Decide how many and *which* vertices migrate

- Compute *how many* vertices have to be moved from π_i to π_j $(1 \leq i, j \leq k)$

$\Rightarrow$ Solve load balancing problem on quotient graph by diffusion

- Migrating balancing flow is ℓ_2-minimal



- Decide *which* vertices are selected for migration

$\Rightarrow$ To move n_{ij} vertices v from π_i to π_j:
Find n_{ij} vertices in π_i with the highest load values in diffusion system j.

- Straightforward approach:
  - Gather all candidates for move on processor j
  - Run sequential selection algorithm
  - Scatter threshold load value
  - Migrate vertices with load value above threshold

Multilevel DibaP
Combining Bubble-FOS/C and TruncCons

Multilevel algorithm DibaP (Diffusion-based Partitioning):

1) and 2) **Recursive coarsening:**

- Fine levels: Fast matching algorithm
- Coarse levels: Two Algebraic Multigrid (AMG) coarsening schemes

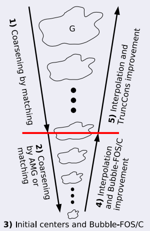
3) **Initial partition:**

- Bubble-FOS/C

4) and 5) **(Local) Improvement:**

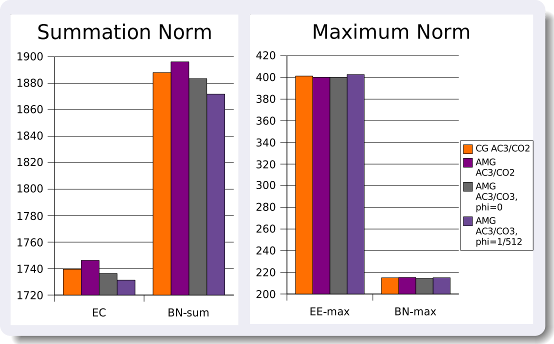
- Small hierarchy levels: Bubble-FOS/C
- Larger hierarchy levels: TruncCons

DibaP Sketch



Hybrid Coarsening
AMG vs Matching

- Comparison of the two coarsening methods for Bubble-FOS/C:

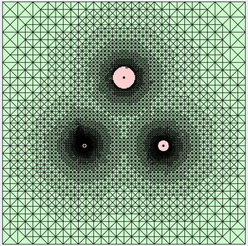


- Small quality penalty for AMG, but can be compensated by more Bubble iterations
- Probable reason for penalty: Star-like coarse vertices

Experiments
Settings, Timing Results

- 3 dynamic graph sequences with 50 frames each
- Generated to mimic 2D adaptive simulations
- Graph sizes between ca. 1M and 5M vertices, degree nearly 3
- Maximum imbalance allowed: 3%
- Platform: Modest cluster with two Intel Xeon X5650 per node (12 cores per node), InfiniBand interconnect

Frame 25 of Small Sequence *slowtric*

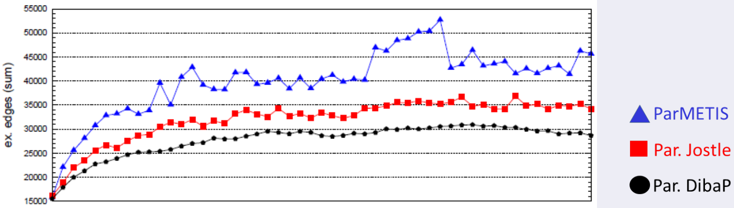


Geom. mean	ParMETIS	Par. JOSTLE	Par. DibaP
36 parts	0.32	0.58	10.50
60 parts	0.26	0.67	12.72

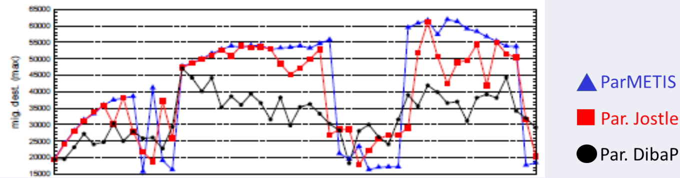
Table: Running time **in seconds** for repartitioning.

Experiments
Quality Results

Edge Cut



Incoming Migration Volume (max norm)



ParMETIS

Small Benchmark Sequence *circles*

Parallel JOSTLE

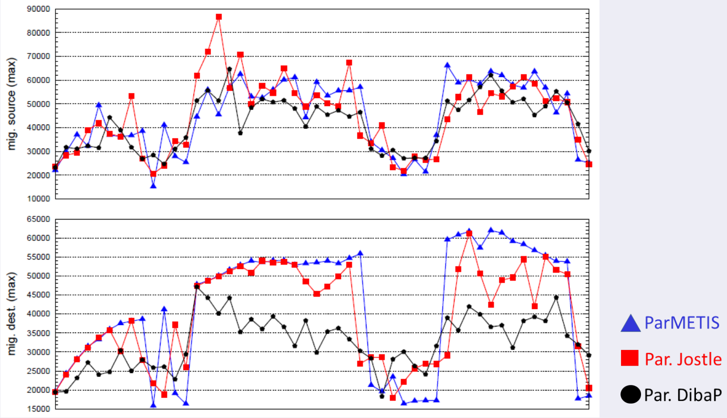
Small Benchmark Sequence *circles*

DibaP

Small Benchmark Sequence *circles*

Experiments
Migration Volume for *biggertrace* Sequence

Outgoing (top) and Incoming (bottom) Migration Volume



Conclusions

- Efficient adaptive numerical computations require good load balancing by repartitioning
- Diffusion identifies dense graph regions
- Bubble-FOS/C can be shown to be a relaxed cut optimizer
- DibaP: Multilevel combination of Bubble-FOS/C and TruncCons, two disturbed diffusive (re)partitioning schemes
- DibaP is especially suitable for **re**partitioning (very good partition quality, migration volume), but repartitioning takes longer than with established tools

⇒ Inherently parallel diffusive repartitioning algorithm

Future work:

- Improve static partitioning of MPI parallel DibaP
- Combination of techniques for faster parallel partitioning

Conclusions

- Efficient adaptive numerical computations require good load balancing by repartitioning
- Diffusion identifies dense graph regions
- Bubble-FOS/C can be shown to be a relaxed cut optimizer
- DibaP: Multilevel combination of Bubble-FOS/C and TruncCons, two disturbed diffusive (re)partitioning schemes
- DibaP is especially suitable for **re**partitioning (very good partition quality, migration volume), but repartitioning takes longer than with established tools

⇒ Inherently parallel diffusive repartitioning algorithm

Future work:

- Improve static partitioning of MPI parallel DibaP
- Combination of techniques for faster parallel partitioning

Thank you!

Questions?

Selected References I

H. Meyerhenke, T. Sauerwald.
Beyond Good Partition Shapes: An Analysis of Diffusive Graph Partitioning.
Submitted to *Algorithmica*, special issue on ISAAC'10.

H. Meyerhenke.
Beyond good shapes: Diffusion-based graph partitioning is relaxed cut optimization.
In *Proc. 21st International Symposium on Algorithms and Computation (ISAAC)*. Springer, 2010.

H. Meyerhenke, B. Monien, and S. Schamberger.
Graph partitioning and disturbed diffusion.
Parallel Computing, 35(10-11):544–569, 2009.

Selected References II

H. Meyerhenke, B. Monien, and T. Sauerwald.
A new diffusion-based multilevel algorithm for computing graph partitions.
Journal of Parallel and Distributed Computing, 69(9):750–761, 2009. Best Paper Awards and Panel Summary: IPDPS 2008.

H. Meyerhenke.
Dynamic Load Balancing for Parallel Numerical Simulations Based on Repartitioning with Disturbed Diffusion.
In *Proc. Internatl. Conference on Parallel and Distributed Systems (ICPADS'09)*. IEEE, 2009.

R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw.
Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM.
J. Parallel Computing, 26:1555–1581, 2000.

Selected References III

B. Hendrickson.
"Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes?".
In: *Proceedings of Irregular'98*, pp. 218–225, Springer-Verlag, 1998.

C. Walshaw, M. Cross, and M. G. Everett.
"A Localised Algorithm for Optimising Unstructured Mesh Partitions".
Intl. J. Supercomputer Appl., Vol. 9, No. 4, pp. 280–295, 1995.