A3: On Balancing of Dynamic Networks

Burkhard Monien[1], Stefan Schamberger[1], Ulf-Peter Schroeder[1], and Henning Meyerhenke[2] *

[1] Heinz Nixdorf Institute and
Faculty of Computer Science, Electrical Engineering and Mathematics,
University of Paderborn
[2] Graduate School of the Paderborn Institute for Scientific Computing (PaSCo)

Abstract. Load balancing is an important prerequisite to efficiently execute dynamic computations on parallel computers. In this context, this project has focussed on two topics: balancing dynamically generated work load cost efficiently in a network and partitioning graphs to equally distribute connected tasks on the processing nodes while reducing the communication overhead. We summarize new insights and results in these areas.

1 Introduction

The efficient usage of parallel computing resources plays a key role for many large scale applications. These applications usually consist of a huge number of smaller calculations which are related via data dependencies. In order to minimize the overall computation time, an efficient parallelization requires these tasks to be distributed equally among all processing nodes. Many applications generate work load dynamically which abolishes any existing equal distribution. Hence, the load must be rebalanced during the runtime of the program. The redistribution should keep the costs induced by the task migration as low as possible. Since placing calculations that depend on each other on different processing nodes results in communication, this should be avoided as far as possible due to the involved costs in terms of latency and bandwidth.

Rebalancing the work load distribution consists of two subproblems: Computing a balancing flow, i. e. determining how much load needs to be migrated over the communication links, and the choice of the tasks to be placed or migrated onto the processing nodes. To study these two problems, the work in this project is based on different models [14, 10].

The first model assumes that all calculations can be performed independently from each other, meaning that either no data dependencies between the tasks exist or that their communication costs are negligable. When determining a rearrangement of the calculations, the attention lies on minimizing the number of migrating tasks, or, from the network's point of view, in stressing the communication links as little as possible. We assume that tasks can be split arbitrarily. This problem is referred to as *dynamic load balancing*. Besides the methods presented here, several alternative approaches to solve it have been investigated and implemented in the Daisy/VDS library within the subproject A2.

While dynamic load balancing focusses on the communication volume occuring during the balancing process, the second model addresses the communication occuring inside the distributed application. The calculations and their dependencies can be modeled by a graph, and the objective is to split this graph into equally sized parts such that the number of edges between

* email: {bm|schaum|ups|henningm}@uni-paderborn.de

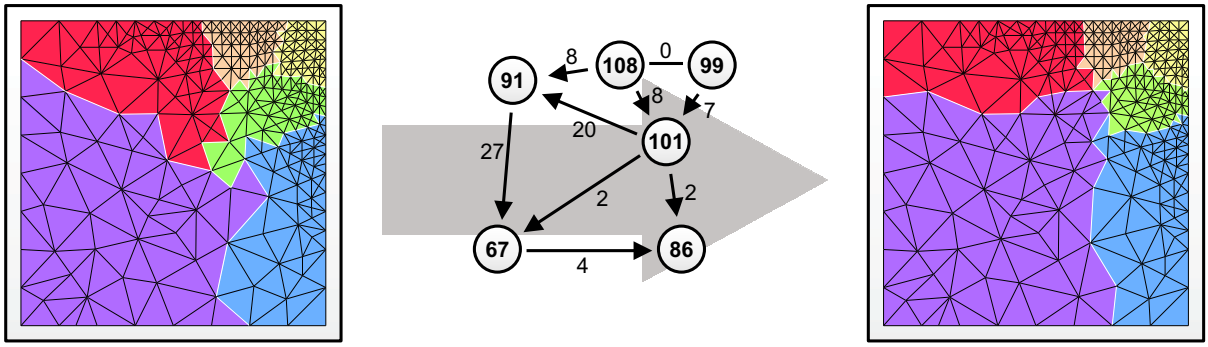

Fig. 1. After refining the mesh, the computational load is unbalanced (left). A balancing flow is computed (middle) and elements are migrated accordingly (right).

different partitions – and hence the induced communication volume – is minimal. This problem is referred to as *graph-partitioning* problem and is known to be NP-complete [28].

An interesting important observation is that the algrebraic representation of a graph reflects many of its properties and allows to adopt algrebraic methods. For a graph $G = (V,E)$ with vertex-edge incidence matrix $\mathbf{A}$, which contains in each column corresponding to edge $e = (u,v)$ the entries -1 and $+1$ in the rows u and v, and 0 elsewhere, the Laplacian matrix $\mathbf{L} \in \mathbb{Z}^{|V|\times|V|}$ of G is defined as $\mathbf{L} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$. Several different methods and their analysis are based on the condition of this matrix.

The goal of this project is to provide theoretical background addressing the dynamic load balancing and the graph partitioning problem. Based on this profound knowledge, improved methods to solve these tasks are developed, implemented and finally applied in practical applications.

One application showing the importance of an efficient load balancing scheme is the *parallel adaptive finite element simulation*. The involved meshes consisting of several million elements representing the discretized geometric space are split into parts and distributed evenly among all processors. Each processor starts computing independently on its part until the next global communication step is required. Depending on the application, the mesh is refined and coarsened in some areas during the computation which causes an imbalance between the processor loads and therefore delays the overall computation. For example, exact simulations of turbulences in fluid dynamics depend on such refinements. In these situations, the computational load must be rebalanced as sketched in figure 1. During this project, the parallel adaptive simulation environment PadFEM has been created. The load balancing and graph partitioning algorithms based on our theoretical research have been integrated and evaluated in this simulation framework.

2 Dynamic Load Balancing

Formally, the load balancing problem is defined as follows. Given a graph $G = (V,E)$ representing the network with $n = |V|$ processing nodes where each node v_i contains work load w_i, the goal is to move load across the links $e_j \in E$ such that finally the weight of each node is

(approximately) equal to

$$\overline{w} = \sum_{i=1}^{n} w_i/n.$$

If the global imbalance vector $w - \overline{w}$ is known, it is possible to find a solution to this problem by solving a linear system of equations [36]. But assuming a more restrictive environment allowing processors of the parallel network only to access information of their direct neighbors, load information has to be exchanged locally in iterations until a balancing flow has been computed.

Two subclasses of local iterative load balancing algorithms are the *diffusion* schemes [9, 4] and the *dimension exchange* schemes [9, 67]. These two classes reflect different communication abilities of the network. Diffusion algorithms assume that a processor can send and receive messages to/from all of its neighbors simultaneously, while the dimension exchange approach is more restrictive and only allows a processor to communicate with one of its neighbors in each iteration. The *alternating direction* iterative scheme [18] represents a mixture of the diffusion and dimension exchange methods. It reduces the number of iteration steps needed for networks constructed by Cartesian products of graphs. The drawback of this scheme is that the resulting flow may have load migration loops tending to infinity.

2.1 The General Diffusion Scheme

The general or first order diffusion scheme (FOS) [9] is derived from Jacobi-Iterations and performs only local operations, i. e. data is only read from neighboring nodes. In each iteration k, it performs the calculations

$$x^{k-1}_{e=(i,j)} = \alpha_e \left(w_i^{k-1} - w_j^{k-1}\right)$$
$$f_e^k = f_e^{k-1} + x_e^{k-1}$$
$$w_i^k = w_i^{k-1} - \sum_{e=(i,*)\in E} x_e^{k-1},$$

with w^0 being the initial work load distribution, x_e^k the amount of load exchanged via edge e in iteration k and f_e^k the computed flow. The parameters α_e have to be properly chosen. In matrix notation, this scheme can be written as $w^k = \mathbf{M}w^k$, where $\mathbf{M} = \mathbf{I} - \alpha\mathbf{L}$ is the diffusion matrix.

An important result is that the First Order Diffusion Scheme converges towards a distinguished balancing flow as stated by the following theorem [11].

Theorem 1. *Given a connected graph and an initial work load distribution, the First Order Diffusion scheme converges towards the $||\cdot||_2$-minimal balancing flow.*

2.2 Optimal Parameters

The convergence of the First Order Diffusion Scheme depends on the parameters α_e. A simple choice is to set all $\alpha_e = (1 + \text{maxdeg}(V))^{-1}$, but for many important graph classes better values are known [22]. In general, the convergence of the iterative diffusion scheme is related to the condition number of the graph's Laplacian matrix $\mathbf{L}$. The knowledge of its spectrum even allows

to determine the optimal α_e values [12, 18]. In this case, only m iteration steps are necessary where m is the number of different eigenvalues of $\mathbf{L}$.

This knowledge can e. g. be applied to reduce the communication in bus like interconnections. This network type allows each processor to communicate with every other, but only one transfer can be executed at a time. To reduce the bus allocations, a virtual topology is introduced which restricts the processors' communication to a few peers only. In order to speed up the load balancing process, topologies with litte different eigenvalues are of interest [19].

2.3 Overrelaxation and Generalization

Similar to the overrelaxation known from the Jacobi-Iterations, this method can also be applied to the First Order Diffusion Scheme. This results in the *Second Order Diffusion Scheme* (SOS) [29] of the form

$$w^1 = \mathbf{M}w^0, w^k = \beta\mathbf{M}w^{k-1} + (1-\beta)w^{k-2},\ k = 2,3,\ldots$$

The fastest convergence is achieved for $\beta = 2/(1+\sqrt{1-\mu_2^2})$, where μ_2 is the second smallest eigenvalue of the diffusion matrix $\mathbf{M}$. The Chebyshev method [12] differs from SOS only by the fact that β depends on k according to

$$\beta_1 = 1,\ \beta_2 = \frac{2}{2-\mu_2^2},\ \beta_k = \frac{4}{4-\mu_2^2\beta_{k-1}},\ k = 3,4,\ldots$$

Summarized, a *polynomial based* load balancing scheme is any scheme for which the work load w^k in step k can be expressed in the form $w^k = p_k(M)w^0$ where $p_k \in \overline{\Pi}_k$. Here, $\overline{\Pi}_k$ denotes the set of all polynomials p of degree $\deg(p) \leq k$ satisfying the constraint $p(1) = 1$. Theorem 1 concerning the convergence can be generalized as follows.

Theorem 2. *Given a connected graph and an initial work load distribution, all diffusion schemes converge towards the $||\cdot||_2$-minimal balancing flow.*

This property has been shown for polynomial based load balancing schemes [12]. The convergence rate of a polynomial based scheme depends on how fast the *error* $e^k = w^k - \overline{w}$ expressing the difference between the load w^k after iteration k and the corresponding average load $\overline{w}$ converges to zero. If α and β are chosen optimally, SOS converges faster than FOS by almost a quadratic factor, while the Chebyshev method performs asymptotically identical to SOS.

2.4 Inhomogeneous Networks

To incorporate heterogeneous computing capacities of the processing nodes and nonuniform communication costs in the network, the diffusion schemes can be generalized. In such an environment, computations perform faster if the load is balanced proportionally to the nodes' computing speed s_i:

$$\overline{w}_i := \frac{\sum_{i=1}^n w_i}{\sum_{i=1}^n s_i} s_i\ .$$

Diffusion in networks with communication links of different capacities are analyzed in [12, 63]. It is shown that the existing balancing schemes can be modified, such that roughly speaking faster communication links get a higher load migration volume than slower ones. These two generalizations can be combined [21].

2.5 Dynamic Networks

The First Order Diffusion Scheme can also be applied to balance load in dynamic networks where communication links fail from time to time or are present depending on the distance of moving nodes [23]. Its convergence depends on the average value of the quotient of the second smallest eigenvalue of the Laplacian matrix and the maximum vertex degree of the networks occurring during the iterations.

2.6 Unsplittable Work Load

In contrast to the above implementations, work load in real-world applications usually cannot be divided arbitrarily often, but only to some extent. The unit-size token model [29] assumes a smallest load entity, the *unit-size token*, and work load is always represented by a collection of this smallest entity. At some state, diffusion schemes are not able to balance the load further due to the computed fractional flows. However, randomized algorithms [20, 24] can be applied to reduce the remaining overload quickly.

3 Graph Partitioning

In its simplest form, the graph partitioning problem can be defined as follows. Given a graph $G = (V,E)$, the vertices of the graph have to be divided into two equally sized sets V_1 and V_2, such that the number of edges connecting vertices from different sets is minimal. This number is referred to as *bisection width*.

The bisection problem can be generalized in several ways. Introducing weights, one looks for solutions where in each part the sum of the respective vertex weights is almost equal and the sum of weights of cut edges is minimal. Furthermore, one could ask to divide the graph into more than two equally sized parts $V_1, \ldots, V_k$, which leads to the k-partitioning problem. Of course, once a bisection algorithm is present, it can be applied to partition a graph into more than two parts by recursive invocation, but in general the direct way can find better solutions [62, 49].

3.1 Bounds on the Bisection Width

Several analytical bounds on the graph bisection width are known. There exists an algorithm which calculates a cut-size that differs from the bisection width by not more than a factor of $O(\sqrt{|V|} \cdot \log(|V|))$ [26]. This first sub-linear factor has been improved to $O(\log^2(|V|))$ [25].

It is known that the graph bisection problem is still NP-complete for graphs of regular degree [6]. Analytical results on these graphs show that almost every large d-regular graph $G = (V,E)$ has a bisection width of at least $c_d \cdot |V|$ where $c_d \to \frac{d}{4}$ as $d \to \infty$ [8, 5].

These bounds can be improved for small values of d. Almost every large 3-regular graph has a bisection width of at least $\frac{1}{9.9}|V| \approx 0.101|V|$ [41, 42]. On the other hand, all sufficiently large 3-regular graphs possess a bisection width of at most $\frac{1}{6}|V|$ [47]. Almost all large 4-regular graphs have a bisection width of at least $\frac{11}{50}|V| = 0.22|V|$ [5], while the bisection width of sufficiently large 4-regular graphs is at most $\frac{2}{5}|V|$.

Some approaches calculate lower bounds on the graph bisection width. These bounds can be used to evaluate the quality of the existing upper bounds as well as to speed up Branch & Bound strategies determining the bisection width of moderately-sized graphs.

One lower bound of the bisection width based on a routing scheme for all pairs of vertices [43]. A small congestion of the routing scheme leads to a high lower bound. Lower bounds on the bisection width can also be derived from algebraic graph theory by relating the bisection problem to an eigenvalue problem. It is well known that the bisection width of a graph $G = (V,E)$ is at least $\lambda_2|V|/4$ with λ_2 being the second smallest eigenvalue of the Laplacian matrix of G. This spectral bound is tight for some graphs [2].

Furthermore, the structure of an optimal bisection can be used to derive improved spectral lower bounds on certain graph classes [2]. For some classes of d-regular graphs one can prove an improved lower bound on the bisection width of roughly $(d/(d-2)) \cdot (\lambda_2|V|/4)$. Furthermore, one can prove a lower bound of $(10+\lambda_2^2-7\lambda_2)/(8+3\lambda_2^3-17\lambda_2^2+10\lambda_2) \cdot (\lambda_2|V|/2)$ for the bisection width of all sufficiently large 3-regular graphs and a lower bound of $(5-\lambda_2)/(7-(\lambda_2-1)^2) \cdot (\lambda_2|V|/2)$ for the bisection width of all sufficiently large 4-regular graphs. These lower bounds are higher than the classical bound of $\lambda_2|V|/4$ for sufficiently large graphs and are applicable to Ramanujan graphs [7, 50]. Any sufficiently large 3-regular Ramanujan graph has a bisection width of at least $0.082|V|$ while sufficiently large 4-regular Ramanujan graphs have a bisection width of at least $0.176|V|$. These values are the best lower bounds for explicitly constructible 3- and 4-regular graphs [47].

These approximation factors are of high theoretical interest, but they are far from acceptable for real applications. Furthermore, the algorithms behind these approximation factors are very complicated and are not suitable to design fast and efficient graph partitioning algorithms.

3.2 Global Heuristics

The first graph partitioning heuristics operate directly on the input-graph. Apart from simple greedy algorithms, a few more elaborated approaches have been developed. A popular one is the spectral bisection which works on the Laplacian matrix $\mathbf{L}$ of the graph. The bisection is based on its second-smallest eigenvalue λ_2. The median m of all components of the corresponding eigenvector e (*Fiedler Vector*) is determined and the vertices of the graph are distributed as $V_1 = \{v \in V : e_v < m\}$ and $V_1 = \{v \in V : e_v > m\}$ [52]. This approach can be extended to more than 2 partitions [32].

In some applications vertices are provided with geometric data. Hence, this additional information can be used to partition the graph. In this field, partitions based on space-filling curves have become popular. Space-filling curves are geometric representations of bijective mappings $M : \{1,\ldots,N^m\} \to \{1,\ldots,N\}^m$. The curve M traverses all N^m cells in the m-dimensional grid

of size N. They have been introduced by Peano and Hilbert in the late 19^{th} century [34, 56]. Partitions based on connected space-filling curves are "quasi optimal" for regular grids and special types of adaptively refined grids [68]. The cut-size is bounded by $C \cdot (|V|/P)^{(d-1)/d}$, where $|V|$ denotes the number of vertices, P the number of partitions, and d the dimension of the graph. The constant C depends on the curve type. The space-filling curve approach is very fast, but it is also known that ignoring the adjacency information of the graph can result in a poor solution quality, especially in case of unstructured graphs [57].

3.3 Local Improvement Strategies

Due to the not always satisfying solution quality of global heuristics, strategies that improve an existing partitioning have been developed. This is achieved by local rearrangements that exchange vertices or sets of vertices between the partitions.

Kernighan-Lin The Kernighan-Lin (KL) heuristic [40] is one of the earliest graph partitioning heuristics and has been developed to optimize placements of electronic circuits. The original algorithm by Kernighan and Lin is based on the exchange of vertex pairs. The method has been modified [27] such that only single vertices are moved. Furthermore, efficient *bucket* data structures enable a linear run-time per pass. Although this heuristic is popular, no bounds on the resulting partition quality are known.

Helpful-Sets The Helpful-Set concept reduces the edge-cut by exchanging sets of vertices between the partitions. The helpfulness of a vertex set is defined by the cut-size reduction that occurs when moving it to another partition. As long as an h-helpful set S with a positive h exists, it is migrated, and an equally sized balancing set $\bar{S}$ from the enlarged part with helpfulness larger than $-h$ is moved back to restore the partition sizes.

An important observation is that as long as the edge-cut is above a certain value, a helpful set and a balancing set do exist. Utilizing this knowlegde provides constructive bounds for the solution quality as stated in the following theorem [35, 46].

Theorem 3. *Let $G = (V,E)$ be a d-regular graph with even d, $d \geq 4$ and $|V| \geq n_0(d)$. Then, the bisection width* $\mathrm{bw}(G)$ *of G is bounded by*

$$\mathrm{bw}(G) \leq \frac{d-2}{4}|V| + 1.$$

Improved results are known for 3 and 4-regular graphs as listed in Section 3.1.

The Helpful-Set heuristic [13, 48, 49] is the algorithmic result of the above theorem and includes several generalizations like the handling of graphs with arbitrary vertex degrees and graphs with vertex and edge weights. It has been succesfully implemented in the graph partitioning library PARTY [53].

3.4 The Multilevel Scheme

The breakthrough in this field is the introduction of the multilevel scheme [33, 61]. Instead of immediately computing a partitioning for the large input graph, vertices are contracted and a

smaller instance with a similar structure is generated. On this instance, the partitioning problem is solved applying a global heuristic. Due to the reduced size it is easier to find sufficiently good solutions. Afterwards, vertices of the original graph are assigned to partitions according to their representatives in the smaller instance. The obtained solution is then further enhanced by a local refinement heuristic.

Instead of applying a global heuristic on the smaller instance, the described process can be repeated recursively, until in the lowest level only a very small graph remains. Hence, a very basic global heuristic can be applied and can even be omitted if the number of remaining vertices equals the requested number of partitions.

The described multilevel algorithm consists of three important tasks: A matching algorithm deciding which vertices are combined in the next level, a global partitioning algorithm applied in the lowest level, and a local refinement algorithm improving the quality of a given partitioning.

3.5 Graph Coarsening

To create a hierachy of smaller graphs with a similar structure, matchings play an important role. A matching algorithm for multilevel partitioning is supposed to be very fast and to have a high matching cardinality and matching weight. Because of the time constraints, the calculation of a maxumim cardinality matching or even a maximum weighted matching would be too time consuming. Therefore, fast algorithms calculating maximal matchings are applied. A very simple algorithm is based on a greedy strategy always adding the next heavier free edge to the matching [1]. A number of alternative approaches have been proposed [33, 38, 39, 44], but do not provide any quality guarantee.

The first linear time approximation algorithm [54, 48] ensures to find a maximum weighted matching with at least half the weight of the optimal solution.

Theorem 4. *Let $G = (V,E)$ be a graph with vertices V and weighted undirected edges E. The Locally-Heaviest algorithm computes a matching of G with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching in linear time $O(|E|)$.*

The decovery of this algorithm initiated in a number of improvements [17, 16]. All of them follow the same strategy: Starting with an initial empty matching, the vertices of the graph are visited in a specific order. For each visited vertex v, it is checked if v is free, i. e. can still be matched, and if v is adjacent to at least one unmatched vertex. If v is free and all neighbors are already matched, v remains free. Otherwise, if v is unmatched and at least one free neighbor exists, the edges to free neighbors are rated and an edge with highest rating is added to the matching.

3.6 Libraries

The efficiency of a graph partitioning methods strongly depends on its specific implementation. There exist several software libraries which provide a large range of different approaches. Examples are CHACO [31], METIS [37], JOSTLE [64], and SCOTCH [51]. The goal of the

libraries is both to provide efficient implementations and to offer a flexible and universal graph partitioning interface to applications.

In this project, the PARTY graph partitioning library has been developed [55, 53, 49]. In contrast to other implementations, it is based on theoretical results yielded during this period. Therefore, it provides quality guarantees for the single steps involved in the calculation.

Although there exists a large number of sequential libraries, only a few parallel implementations are available. This is due to the complexity involved in parallel programming. Furthermore, the applied heuristics like KL are basically of sequential nature, hence modifications are required, which sometimes introduce new limitations. The most popular distributed libraries are the parallel versions of METIS [37] and JOSTLE [64, 66, 65]. These tools essentially apply the same techniques as their sequential counterparts, are quite fast and deliver solutions that are acceptable for most applications.

4 Further Challenges

The existing graph partitioning heuristics provide good solutions and are very fast. However, although great progress has been made in this area, many questions remain [30]. While the global cut-size is the classical metric that most graph partitioners optimize, it is not necessarily the metric that models the real costs of an application. In FEM computations for example, the true communication volume can significantly differ from the number of cut-edges. In this case, the number of vertices situated at partition boundaries reflects the information to be exchanged much more accurately. Furthermore, aspects like latency between the processing nodes due to the structure of the communication network are totally ignored. Another questionable point is the applied norm. In synchronized computations, the slowest processor specifies the overall speed, hence the maximum norm would be appropriate, while the usually applied cut-size is a summation norm.

Dynamically changing applications often require a work load distribution that guarantees both, a low overhead caused by the load migration and little communication during the calculation. This results in a multi-objective optimization, comprising the dynamic load balancing as well as the graph partitiong problem. It can be modeled via a graph. An unbalanced partition π_t has to be transformed into a balanced distribution π_{t+1} while obeying the two constraints.

Currently, most of the mentioned implementations first determine how much load to migrate and then restrict the exchange steps of the local refinement process according to this number. Better solutions can be obtained by integrating the migration costs directly into the improvement procedure itself [3, 60]. An alternative approach focuses on the partition shapes [15] and iteratively decreases their aspect ratios. The involved learning steps can be modified by introducing diffusive operations. This results in an algorithm that determines the migrating vertices depending on $||\cdot||_2$-minimal balancing flows [58, 59, 45]. Although not containing any explicit objectives, this approach finds good solutions not only concerning the edge-cut but also the number of partition boundary vertices.

Although first ideas exist to address the multi-objective optimization problem, much more research is required in this field to better understand the interaction between the different con-

straints. Combining this with the need of more appropriate metrics and norms like the maximum number of boundary vertices per partition, an interesting and important challenge remains to be solved in the future.

5 Acknowledgements

Following persons have contributed to this project: Sergej Bezrukov, Stephan Blazy, Wolfgang Borchers, Ralf Diekmann, Uwe Dralle, Robert Elässer, Jan Hungershöfer, Oliver Marquardt, Henning Meyerhenke, Burkard Monien, Robert Preis, Markus Röttger, Stefan Schamberger, Ulf-Peter Schroeder, Jürgen Schulze, Maria Specovius-Neugebauer, Walter Unger, Kerstin Wielage, and Jens-Michael Wierum.

6 Theses in this project (chronological list)

6.1 Phd Thesis

Ralf Diekmann "Lastverteilungsverfahren für datenparallele Anwendungen", Universität Paderborn, 1998.

Markus Röttger "Effiziente Einbettungen von Gittern in Gitter und Hypercubes", Universität Paderborn, 1998.

Ulf-Peter Schroeder "Einbettung unter besonderer Berücksichtigung von Gitternetzwerken", Universität Paderborn, 2000.

Robert Preis "Analysis and Design of Efficient Graph Partitioning Methods", Universität Paderborn, 2000.

Jürgen Schulze "Effiziente Parallele Verfahren zur Lösung dünn besetzter Gleichungssysteme", Universität Paderborn, 2000.

Stephan Blazy "Numerische Approximation der Stokes-Gleichung mit künstlichen Randbedingungen in 3D Rohrsystemen", Universität Paderborn, 2001.

Robert Elsässer "Spectral Methods for Efficient Loadbalancing Strategies", Universität Paderborn, 2002.

Jens-Michael Wierum "Anwendung diskreter raumfüllender Kurven : Graphpartitionierung und Kontaktsuche in der Finite-Elemente-Simulation", Universität Paderborn, 2003.

Kerstin Wielage "Analysis of Non-Newtonian and Two-Phase Flows", Universität Paderborn, 2003.

6.2 Habilitations

Sergej Bezrukov "Discrete Extremal Problems on Graphs and Posets", Universität Paderborn, 1996.

Maria Specovius-Neugebauer "Approximation of Stokes-Problems in Unbounded Domains", Universität Paderborn, 1997.

References

1. D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.
2. S. Bezrukov, R. Elsässer, B. Monien, R. Preis, and J.-P. Tillich. New spectral lower bounds on the bisection width of graphs. *Theoretical Computer Science*, 320:155–174, 2004.
3. R. Biswas and L. Oliker. PLUM: Parallel load balancing for adaptive unstructured meshes. *Parallel and Distributed Computing*, 51(2):150–177, 1998.
4. J. E. Boillat. Load balancing and poisson equation in a graph. *Concurrency - Practice & Experience*, 2:289–313, 1990.
5. B. Bollobas. The isoperimetric number of random regular graphs. *Europ. J. Combinatorics*, 9:241–244, 1988.
6. T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sisper. Graph bisection algorithms with good average case behaviour. *Combinatorica*, 7(2):171–191, 1987.
7. P. Chiu. Cubic ramanujan graphs. *Combinatorica*, 12(3):275–285, 1992.
8. L.H. Clark and R.C. Entringer. The bisection width of cubic graphs. *Bulletin of th Australian Mathematical Society*, 39:389–396, 1988.
9. G. Cybenko. Load balancing for distributed memory multiprocessors. *Parallel and Distributed Computing*, 7:279–301, 1989.
10. T. Decker, B. Monien, and R. Preis. Towards optimal load balancing topologies. In A. Bode, T. Ludwig, W. Karl, and R. Wismüller, editors, *Proceedings of the 6th EuroPar Conference*, number 1900 in LNCS, pages 277–287, 2000.
11. R. Diekmann, A. Frommer, and B. Monien. Nearest neighbor load balancing on graphs. In *Proceedings of the European Symposium on Algorithms (ESA'98)*, volume 1461 of *LNCS*, pages 429–440, 1998.
12. R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.
13. R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. In D. F. Hsu, A. L. Rosenberg, and D. Sotteau, editors, *Interconnection Networks and Mapping and Scheduling Parallel Computations*, volume 21, pages 57–73. AMS, 1995.
14. R. Diekmann, B. Monien, and R. Preis. Load balancing strategies for distributed memory machines. In F. Karsch, B. Monien, and H. Satz, editors, *Multi-Scale Phenomena and Their Simulation*, pages 255–266. World Scientific, 1997.
15. R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. *Journal of Parallel Computing*, 26:1555–1581, 2000.
16. D. E. Drake and S. Hougardy. Improved linear time approximation algorithms for weighted matchings. In *Approximation, Randomization, and Combinatorial Optimization (APPROX/RANDOM 03)*, LNCS 2764, pages 14–23, 2003.
17. D. E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003.
18. R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction loadbalancing schemes. In P. Amestoy et al., editor, *EuroPar Parallel Processing*, LNCS 1685, pages 280–290, 1999.
19. R. Elsässer, R. Kralovic, and B. Monien. Sparse topologies with small spectrum size. *Theor. Comput. Sci.*, 307(3):549–565, 2003.
20. R. Elsässer and B. Monien. Load balancing of unit size tokens and expansion properties of graphs. In *SPAA*, pages 266–273, 2003.
21. R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35:305–320, 2002.
22. R. Elsässer, B. Monien, and S. Schamberger. Toward optimal diffusion matrices. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, page 67 (CD). IEEE Computer Society, 2002.
23. R. Elsässer, B. Monien, and S. Schamberger. Load balancing in dynamic networks. In *Proceedings of the 7th international Symposium on Parallel Architectures, Algorithms and Networks, (I-SPAN'04)*, pages 193–200. IEEE Conputer Society, 2004.
24. R. Elsässer, B. Monien, and S. Schamberger. Load balancing of indivisible unit size tokens in dynamic and heterogeneous networks. In *Proceedings of the 12th European Symposium on Algorithms*, volume 3221 of *LNCS*, pages 640–651, 2004.
25. U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.
26. U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *Proc. Symp. on Theory of Computing (STOC)*, 2000.
27. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. IEEE Design Automation Conference*, pages 175–181, 1982.
28. M. R. Garey and D. S. Johnson. *Computers and Interactability - A Guide to the Theory of NP-Completeness*. Freemann, 1979.

29. B. Ghosh, S. Muthukrishnan, and M.H. Schultz. First- and second-order diffusive methods for rapid, coarse, distributed load balancing. *Theory of Computing Systems*, 31(4):331–354, 1998.
30. B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Irregular'98*, number 1457 in LNCS, pages 218–225, 1998.
31. B. Hendrickson and R. Leland. *The Chaco user's guide – Version 2.0*, 1994.
32. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.
33. B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing'95*, 1995.
34. D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38:459–460, 1891.
35. J. Hromkovič and B. Monien. The bisection problem for graphs of degree 4 (configuring transputer systems). In *Mathematical Foundations of Computer Science 1991, MFCS'91*, volume 520 of *LNCS*, pages 211–220, 1991.
36. Y. F. Hu and R. F. Blake. In improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25(4):417–444, 1999.
37. G. Karypis and V. Kumar. *MeTis: A Software Package for Partitioning Unstrctured Graphs, Partitioning Meshes, [...], Version 4.0*, 1998.
38. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.
39. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, 20(1), 1999.
40. B. W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *The Bell Systems Technical J.*, pages 291–307, 1970.
41. A. V. Kostochka and L. S. Melnikov. On bounds of the bisection width of cubic graphs. In J. Nesetril and M. Fiedler, editors, *Proc. Fourth Czechoslovakian Symp. on Combinatorics, Graphs and Complexity*, pages 151–154. Elsevier Science Publishers B.V., 1992.
42. A. V. Kostochka and L. S. Melnikov. On a lower bound for the isoperimetric number of cubic graphs. In V. F. Kolchin et al., editor, *Probabilistic Methods in Discrete Mathematics, Proc. third Int. Petrozavodsk Conf.*, volume 1 of *Progress in Pure and Applied Discrete Mathematics*, pages 251–265. TVP/VSP, 1993.
43. F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1992.
44. J. Magun. Greedy matching algorithms, an experimental study. *ACM J. of Experimental Algorithms*, 3, 1998.
45. H. Meyerhenke and S. Schamberger. Balancing parallel adaptive fem computations by solving systems of linear equations. In *Euro-Par 2005*, number 3648 in LNCS, pages 209–219, 2005.
46. B. Monien and R. Diekmann. A local graph partitioning heuristic meeting bisection bounds. In *8th SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.
47. B. Monien and R. Preis. Upper bounds on the bisection width of 3- and 4-regular graphs. In *26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, LNCS 2136, pages 524–536, 2001.
48. B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. *Parallel Computing*, 26(12):1609–1634, 2000.
49. B. Monien and S. Schambeger. Graph partitioning with the party library: Helpful-sets in practice. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pages 198–205. IEEE Computer Society Press, 2004.
50. M. Morgenstern. Existence and explicit constructions of $q+1$ regular ramanujan graphs for every prime power q. *Journal Comb. Theory*, 62(1):44–62, 1994.
51. F. Pellegrini. *SCOTCH 3.1 User's guide*, 1996.
52. A. Pothen, H. D. Simon, and K. P. Liu. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. on Matrix Analysis and Applications*, 11(3):430–452, 1990.
53. R. Preis. The PARTY graphpartitioning-library – user manual, 1998.
54. R. Preis. Linear time $1/2$-approximation algorithm for maximum weighted matching in general graphs. In *Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *LNCS*, pages 259–269, 1999.
55. R. Preis and R. Diekmann. PARTY - a software library for graph partitioning. In B. H. V. Topping, editor, *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71. 1997.
56. H. Sagan. *Space Filling Curves*. Springer, 1994.
57. S. Schambeger and J.M. Wierum. Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In *Proceedings of the 7th International Conference on Parallel Computing Technologies*, volume 2763 of *LNCS*, pages 165–179, 2003.

58. S. Schamberger. On partitioning fem graphs using diffusion. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium, (IPDPS'04)*, page 277 (CD). IEEE Computer Society, 2004.
59. Stefan Schamberger. A shape optimizing load distribution heuristic for parallel adaptive fem computations. In *Proceedings of the 8th International Conference on Parallel Computing Technologies*, volume 3606 of *LNCS*, pages 263–277, 2004.
60. K. Schloegel, G. Karypis, and V. Kumar. A unified algorithm for load-balancing adaptive scientific simulations. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, page 59 (CD). IEEE Computer Society, 2000.
61. Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *J. Par. Dist. Comp.*, 47(2):109–124, 1997.
62. H. Simon and S. Teng. How good is recursive bisection. *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.
63. V. S. Sunderarm and G. A. Geist. Heterogeneous parallel and distributed computing. *Par. Comp.*, 25:1699–1721, 1999.
64. C. Walshaw. *The Jostle user manual: Version 2.2*. University of Greenwich, 2000.
65. C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.
66. C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel Distributed Computing*, 47(2):102–108, 1997.
67. C. Xu and F. C. M. Lau. *Load Balancing in Parallel Computers*. Kluwer, 1997.
68. G. Zumbusch. *Parallel Multilevel Methods: Adaptive Mesh Refinement and Loadbalancing*. Teubner, 2003.