Balancing Parallel Adaptive FEM Computations by Solving Systems of Linear Equations*

Henning Meyerhenke and Stefan Schamberger

Universität Paderborn,
Fakultät für Elektrotechnik, Informatik und Mathematik
Fürstenallee 11, D-33102 Paderborn
{henningm,schaum}@uni-paderborn.de

Abstract. Load balancing plays an important role in parallel numerical simulations. State-of-the-art libraries addressing this problem base on vertex exchange heuristics that are embedded in a multilevel scheme. However, these are hard to parallelize due to their sequential nature. Furthermore, libraries like Metis and Jostle focus on a small edge-cut and cannot obey constraints like connectivity and straight partition boundaries, which are important for some numerical solvers.
In this paper we present an alternative approach to balance the load in parallel adaptive finite element simulations. We compute a distribution that is based on solutions of linear equations. Integrated into a learning framework, we obtain a heuristic that contains a high degree of parallelism and computes well shaped connected partitions. Furthermore, our experiments indicate that we can find solutions that are comparable to those of the two state-of-the-art libraries Metis and Jostle also regarding the classic metrics like edge-cut and boundary length.

Keywords: Parallel adaptive FEM computations, load balancing, graph partitioning.

1 Introduction

Finite Element Methods (FEM) are used extensively by engineers to analyze a variety of physical processes which can be expressed via Partial Differential Equations (PDE). The domain on which the PDEs have to be solved is discretized into a mesh, and the PDEs are transformed into a set of equations defined on the mesh's elements (see e. g. [1]). These can then be solved by iterative methods such as Conjugate Gradient (CG) and Multigrid. Due to the very large amount of elements needed to obtain an accurate approximation of the original problem, this method has become a classical application for parallel computers. The parallelization of numerical simulation algorithms usually follows the Single-Program Multiple-Data (SPMD) paradigm: Each processor executes the same code on a different part of the data. This means that the mesh has to be split into P sub-domains and each sub-domain is then assigned to one of the P processors. To minimize the overall computation time, all processors should thereby roughly

* This work is supported by the German Science Foundation (DFG) project SFB-376 and by DFG Research Training Group GK-693.

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 209–219, 2005.

contain the same amount of elements. Since iterative solution algorithms perform mainly local operations, i. e. data dependencies are defined by the mesh, the parallel algorithm mainly requires communication at the partition boundaries. Hence, these should be as small as possible. Depending on the application, some areas of the simulation space require a higher resolution and therefore more elements. Since the location of these areas is not known beforehand or can even vary over time, the mesh is refined and coarsened during the computation. However, this can cause imbalance between the processors' load and therefore delay the simulation. Hence, the element distribution needs to be rebalanced. The application is interrupted and the at this point static repartitioning problem is solved. Though this interruption should be as short as possible, it is also important to find a new balanced partitioning with small boundaries that does not cause too many elements to change their processor. Migrating elements can be an extremely costly operation since large amounts of data have to be sent over communication links and stored in complex data structures.

The described problem can be expressed as a graph (re-)partitioning problem. The mesh is transformed into a graph where the vertices represent the computational work and the edges their interdependencies. Due to the complexity of the problem, the large input sizes and the given time constraints, existing libraries that address the graph (re-)partitioning problem are based on heuristics. State-of-the-art implementations like Metis [2], Jostle [3] or Party [4] follow the multilevel scheme [5]. Vertices of the graph are contracted according to a matching and a new level consisting of a smaller graph with a similar structure is generated. This is repeated, until in the lowest level only a small graph remains. The (re-)partitioning problem is then solved for this small graph and vertices in higher levels are assigned to partitions according to their representatives in the next lower level. Additionally, a local improvement heuristic is applied in every level. By exchanging vertices between partitions, it reduces the number of cut edges or the boundary size as well as balances the partition sizes. Hence, the final solution quality mainly depends on this heuristic. Implementations are usually based on the Kerninghan-Lin (KL) heuristic [6], while the local refinement in Party is derived from theoretical analysis with Helpful-Sets (HS) [7].

To address the load balancing problem during parallel computations, distributed versions of the libraries Metis and Jostle have been developed. Both of them apply about the same multilevel techniques as their single processor version, but special attention must be paid to the local improvement heuristic due to its sequential nature. As an example, a coloring of the graph's vertices assures in the parallel library ParMetis [8] that during the KL refinement no two neighboring vertices change their partition simultaneously and therefore destroy the consistency of the data structures. In contrast to Metis, where vertices stay on their partition until a new distribution has been computed, the parallel version of Jostle [9] maps each sub-domain to a single processor and vertices which migrate do so already during the computation of the repartitioning. Usually, Metis is very fast while Jostle takes longer but often computes better solutions. The HS heuristic in Party exchanges sets between partitions that sometimes contain

a large number of vertices. Hence, even more overhead would be necessary to ensure data consistency in a parallel implementation.

While the global edge-cut is the classical metric that most graph partitioners optimize, it is not necessarily the best metric to follow because it does not model the real communication and runtime costs of FEM computations as described in [10]. Hence, different metrics have been implemented inside the local refinement process modeling the real objectives more closely. In [11], the costs emerging from vertex transfers is taken into consideration while Metis is also capable of minimizing the number of boundary vertices.

A completely different approach is undertaken in [12]. Since the convergence rate of the CGBI domain decomposition solver in the PadFEM environment depends on the geometric shape of a partition, the integrated load balancer iteratively decreases the aspect ratios by applying a bubble like algorithm. Although different to the multilevel-schemes, this approach also contains a strictly sequential section and suffers from some other difficulties that are described in [13]. However, the latter paper introduces an implementation that eliminates most of these problems by replacing the sequential growing mechanism of the bubble framework by a few iterations of the first order diffusion scheme (FOS) [14]. This leads to a graph partitioning algorithm that contains a high degree of parallelism and produces well shaped partitions. Unfortunately, it is unclear how many FOS iterations must be performed. This question is overcome in [15] introducing FOS/A. This diffusion scheme does not balance the load but converges to a state with a load distribution similar to the situation after a few FOS iterations. Its drawback is the long execution time, and its fine-grain parallelism is hard to exploit on today's processors.

In this paper we present the (re-)partitioning heuristic MF(ϕ), which is based on the same framework as the implementations from [13] and [15]. However, in contrast to the latter that distribute the vertices of a graph according to their load, our approach is based on the flow over the edges. The main advantage is that the computation of a $\|\cdot\|_2$-minimal balancing flow, which is equivalent to solving a system of linear equations, has been studied very well and that a variety of methods addressing it exist. Among them are faster diffusion schemes like the second order scheme (SOS) [14] as well as algorithms that require more global knowledge like CG solvers. Thus, one can choose the most appropriate implementation according to the underlying hardware. The remaining part of the paper is organized as follows. The next section briefly recaptures the bubble framework from [12] and explains the main idea. In Sec. 3 we propose a new growing mechanism which we integrate into this framework in Sec. 4. Afterwards, we present our experiments in Sec. 5 before we give a short conclusion.

2 The Bubble Framework

The idea of the bubble framework is to start with an initial, often randomly chosen vertex (seed) per partition, and all sub-domains are then grown simultaneously in a breadth-first manner. Colliding parts form a common border and

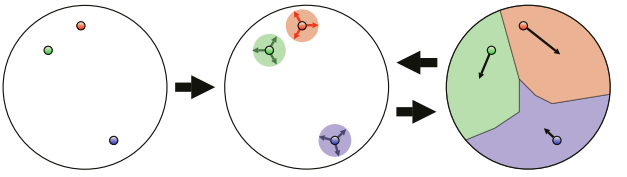

Fig. 1. The three operations of the learning bubble framework: Init: Determination of initial seeds for each partition (left). Grow: Growing around the seeds (middle). Move: Movement of the seeds to the partition centers (right).

keep on growing along this border – "just like soap bubbles". After the whole mesh has been covered and all vertices of the graph have been assigned this way, each component computes its new center that acts as the seed in the next iteration. This is usually repeated until a stable state, where the movement of all seeds is small enough, is reached. This procedure is based on the observation that within "perfect" bubbles, the center and the seed vertex coincide. Figure 1 illustrates the three main operations.

The growing mechanisms from [13] and [15] are based on diffusion. The main idea behind applying it in a graph partitioning heuristic is the fact that load primarily diffuses into densely connected regions of the graph rather than into sparsely connected ones. Following this observation one can expect to identify seeds inside such regions and therefore small partition boundaries in less dense areas. Additionally, since the load spreads around a seed vertex, the partitions should be connected and well shaped.

The remaining part of this paper is based on the following thought: If load diffuses faster into dedicated regions, then the flow over the edges directing there must be higher than the flow over edges pointing elsewhere. Hence, a $\|\cdot\|_2$-minimal flow should provide similar information as a load distribution computed by the FOS/A scheme from [15], with the advantage that a variety of faster methods are known to compute it.

3 A Growing Mechanism Based on Linear Equations

In this section we propose a new growing mechanism that is based on a $\|\cdot\|_2$-minimal flow in a network. This network G_ϕ is composed of the dual graph G corresponding to the mesh, and an extra vertex x that is connected with every other vertex of G. All edges $e \in E$ of G are assigned a weight of $w_e = 1$ while the weight of the edges incident to x are set to some constant $\phi > 0$. Now, independently for each partition p, we place a total of $|V|$ load equally on p's vertices and compute a $\|\cdot\|_2$-minimal flow f_p over the edges that transports all load to the extra vertex x. Since we minimize f_p according to the $\|\cdot\|_2$-norm, the load will not be sent directly to x, but also makes some 'detours' via other vertices in G. According to the idea mentioned in the last section, the flow thereby prefers densely connected regions of the graph. The weight constant ϕ

determines the spreading of the flow. If ϕ is large, it is cheaper to send most load directly to x, while if ϕ is small, the costs of the 'detour' into the graph are compensated by less utilized edges incident to x that can be chosen. In the extreme cases, if $\phi \to \infty$, all load is sent directly to x, while if $\phi \to 0$, the $\|\cdot\|_2$-minimal flow will converge towards the balancing flow that distributes the load equally in the original graph G.

The assignment of the vertices to the partitions is based on the amount of flow over the edges incident to x. We define a height function $h_p : V \cup \{x\} \to \mathbb{R}$ for each partition p, such that $h_p(v) = h_p(u) + f_{p_{(u,v)}} \cdot w_{(u,v)}\, \forall u \in \mathrm{adj}(v)$. Since f_p is the $\|\cdot\|_2$-minimal flow, this function is well defined and unique except for a constant, which we determine by setting the height of x to $h_p(x) = 0$. Now, we assign each vertex to that partition with the maximal height, meaning that the new partitioning π is defined by $\pi(v) = p : h_p(v) \geq h_q(v)\, \forall q \in \{1, \ldots, P\}$. If the maximum is not unique, we choose one of the eligible partitions arbitrarily.

Formally, let $G = (V, E)$ be an undirected, connected graph and $\mathbf{A} \in \{-1, 0, +1\}^{|V| \times |E|}$ its unweighted vertex-edge incidence matrix. $\mathbf{A}$ 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 unweighted Laplacian $\mathbf{L} \in \mathbb{Z}^{|V| \times |V|}$ is defined as $\mathbf{L} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$. If we extend G by an additional vertex x and connect it to every other vertex with an edge of weight ϕ, we obtain the graph $G_\phi = (V \cup \{x\}, E \cup \{\{v, x\} : v \in V\})$ with edge weights $w_e = 1\ \forall e \in E$ and $w_{\{v,x\}} = \phi\ \forall v \in V$. The weighted Laplacian matrix $\mathbf{L}_\phi \in \mathbb{R}^{|V|+1 \times |V|+1}$ of G_ϕ is defined as $\mathbf{L}_\phi = \mathbf{A}_\phi \mathbf{W} {\mathbf{A}_\phi}^{\mathrm{T}}$, where $\mathbf{A}_\phi$ denotes the unweighted vertex-edge incidence matrix of G_ϕ, and the entries of the diagonal matrix $\mathbf{W} \in \mathbb{R}^{|E|+|V| \times |E|+|V|}$ are set to $(w_{ee}) = w_e$. Hence, with $\mathbf{I}$ being the identity, $\mathbf{L}_\phi$ can be written as:

$$\mathbf{L}_\phi = \begin{pmatrix} \begin{pmatrix} & & \\ & \mathbf{L} + \phi\mathbf{I} & \\ & & \end{pmatrix} & \begin{matrix} -\phi \\ \vdots \\ -\phi \end{matrix} \\ \begin{matrix} -\phi & \cdots & -\phi \end{matrix} & |V| \cdot \phi \end{pmatrix} \tag{1}$$

Our goal is to compute a $\|\cdot\|_2$-minimal flow f_p from the vertices of the partition p to the additional vertex x. By setting the vectors $s_p, t \in \mathbb{R}^{|V|+1}$ to

$$(s_{p_v}) = \begin{cases} |V|/|\{v : \pi_p(v) = p\}| & : \ \pi(v) = p \\ 0 & : \ \text{otherwise} \end{cases} \qquad (t_v) = \begin{cases} |V| & : \ v = x \\ 0 & : \ \text{otherwise} \end{cases}$$

we place $|V|$ load equally on p's vertices and the corresponding 'negative' load on x. Then, we have to solve the quadratic minimization problem

$$\text{min!}\ \frac{1}{2} {f_p}^{\mathrm{T}} \mathbf{W}^{-1} f_p \ \text{with respect to}\ \mathbf{A}_\phi f_p = s_p - t\ . \tag{2}$$

Due to [16], we know that we can find the optimal f_p for (2) by first solving the linear equation

$$\mathbf{L}_\phi \lambda_p = s_p - t\ . \tag{3}$$

00 *Algorithm MF(G, π, ϕ, l, i)*
01 **in** *each loop* l
02 **if** π *is undefined*
03 π = *determine-seeds*(G) /* *initial seeds* */
04 **else**
05 **parallel for** *each partition* p /* *contraction* */
06 *solve* $\mathbf{L}_\phi \lambda_p = s_p - t$ *and compute* h_p
08 $\pi(v) = \begin{cases} p : h_p(v) \geq h_p(u)\, \forall u \in V \\ -1 : \text{otherwise} \end{cases}$
09 **parallel for** *each partition* p /* *consolidation* */
10 *solve* $\mathbf{L}_\phi \lambda_p = s_p - t$ *and compute* h_p
12 $\pi(v) = p : h_p(v) \geq h_q(v)\, \forall q \in \{1, \ldots, P\}$
13 **in** *each iteration* i
14 **parallel for** *each partition* p /* *consolidation with ...* */
15 *solve* $\mathbf{L}_\phi \lambda_p = s_p - t$ *and compute* h_p
17 $\pi(v) = p : h_p(v) \geq h_q(v)\, \forall q \in \{1, \ldots, P\}$
18 *scale-balance*(π) /* *... scale balancing* */
19 *greedy-balance*(π) /* *greedy balancing* */
20 **return** *smooth*(π) /* *smoothing* */

Fig. 2. Sketch of the MF(ϕ) heuristic.

$\mathbf{L}_\phi$ is sparse and symmetric positive semidefinite. Since $\langle s_p - t, \mathbb{1} \rangle = 0$ and the rank of $\mathbf{L}_\phi$ is $|V|$, the solution of (3) is unique except for a constant. Nevertheless, we now can determine the unique $\|\cdot\|_2$-minimal flow from the computed potential λ_p as

$$f_{p(u,v)} = w_{\{u,v\}} \cdot (\lambda_{p_u} - \lambda_{p_v}) \ . \tag{4}$$

Since we are interested in the height function $h_p(v)$, we can skip the flow computation (4) and assign $h_p(v) = \lambda_{p_v} - \lambda_{p_x}$. The new partitioning π can then be determined as described above, while the new partition seed is the vertex with the highest load according to h_p.

4 The MF(ϕ) Heuristic

In this section we describe the integration of the proposed growing mechanism into the bubble framework. The resulting algorithm is sketched in Fig. 2. It can either be invoked with or without a valid partitioning π. In the latter case, we determine initial seeds randomly (line 3). Otherwise, we contract the given partitions (lines 5-8) applying the mechanism proposed in Sec. 3. Note that in either case π only contains a single vertex for each partition when entering line 9. Following the bubble framework, we then grow the partitions from the seeds. However, if we determined single seeds right after the last contraction, these would be the same ones as before and no movement would occur. Hence, it is necessary to apply at least one consolidation (lines 9-12) between two contractions. In contrast to a contraction that determines a single vertex per partition

(line 8), a consolidation results in a partitioning (lines 12/17). In the following step, the load is placed equally on the vertices of the whole partition, which causes it to move into denser regions of the graph as mentioned before.

To further enhance the solution quality, additional consolidations can be performed (lines 13-18). Furthermore, these are used for balancing by scaling the height functions h_p. If a partition is too small, h_p is multiplied by a constant $b_p > 1$, while if it is too large, a constant $b_p < 1$ is chosen. Although the choice of b is limited because no partition must become empty, this approach can find almost balanced solutions in most cases. To ensure a certain size, we perform a greedy balancing operation (line 19), where we compute a $\|\cdot\|_2$-minimal flow in the partition graph and move the vertices that cause the least error according to the height functions. The whole learning process is then repeated several times. Before returning the partitioning π, we migrate vertices if the number of their adjacent vertices in another partition is larger than the number in the current partition. This compensates numerical imprecisions that occur during the flow computation and further smoothes the partition boundaries. However, if the number of vertices in a partition is small compared to its boundary length, it might also lead to a higher imbalance.

An interesting point is the lack of an explicit objective function. Except for the balancing, the MF(ϕ) heuristic does not contain any directives what metric to minimize. This is also the case for the algorithms from [13, 15].

The run-time of MF(ϕ) greatly depends on the linear equation solver. Currently, we apply a basic CG implementation. However, due to the special structure of $\mathbf{L}_\phi$, several optimizations are possible. As indicated in lines 5, 9 and 14, all P linear systems can be solved independently. Hence, even if we apply solvers other than diffusive ones which require more global knowledge, a large amount of parallelism remains.

5 Experiments

In this section we describe our experiments with the new heuristic MF(ϕ) and compare its solutions to those of the parallel versions of the state-of-the-art graph (re-)partitioning libraries Metis and Jostle. Furthermore, we include the results of the Party/DB library from [15]. The benchmark instances are created as described in [17] and are available via [18]. Each benchmark consists of 101 frames, each containing a graph of around 15000 vertices. Though the instances are quite small, important observations can already be made. Due to space limitations we only present the data of a single benchmark here. The results of the omitted experiments are similar, however.

The libraries Metis (version 3.1) and Jostle (version 3.0) both offer a large number of options. For the presented evaluation, we chose the recommended values from their manual, respectively, and left the remaining parameters at their default. This means that Metis operates with an *itr* value of 1000.0 and Jostle uses the options *threshold = 20, matching = local, imbalance = 3*. Note that Jostle seems to ignore the imbalance setting and computes totally balanced

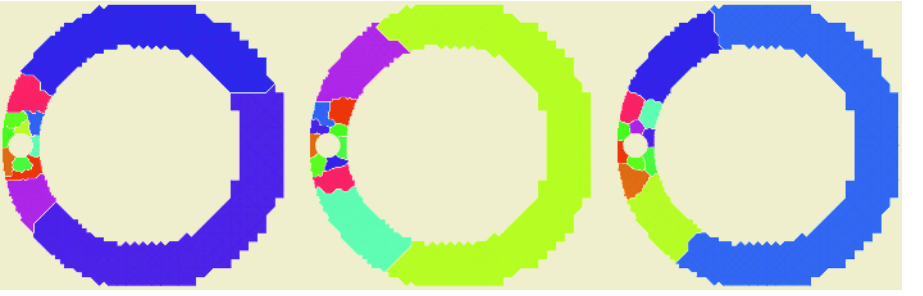

Fig. 3. Partitionings in frame 50 of the 'ring' benchmark computed by Metis (left), Jostle (middle) and the MF(ϕ) heuristic (right).

partitions, except for the initial solution where the sequential versions of the libraries are applied. The MF(ϕ) heuristic is invoked with $\phi = 0.01$ and performs 2 loops with 4 iterations, respectively.

We measure the partitioning quality according to a number of metrics, because it is known that the edge-cut does not necessarily model the real costs [10]. Depending on the application, some of the metrics described in the following might be more important than others. *External edges:* Number of edges that are incident to exactly one vertex of partition p. *Boundary vertices:* Number of vertices of partition p that are adjacent to at least one vertex from a different partition. *Send volume:* The amount of outgoing information is the sum of the adjacent partitions different to p that each vertex residing inside partition p has. *Receive volume:* The amount of incoming information is the number of vertices of partitions different to p adjacent to at least one vertex of partition p. *Diameter:* The longest shortest path between two vertices of the same partition. Infinity, if the partition is not connected. *Outgoing migration:* Number of vertices that have to be migrated to a different partition. *Incoming migration:* Number of vertices that have to be migrated from a different partition. Furthermore, the quality of a partitioning depends on its balance. A less balanced solution allows other metrics to improve further and makes comparisons less meaningful. Please note that we have omitted the run-times since our prototypic implementation is some magnitudes slower than its competitors.

In addition, for the listed metrics we consider three different norms. Given the values $x_1, \ldots, x_P$, the norms are defined as follows: $\|X\|_1 := x_1 + \ldots + x_P$, $\|X\|_2 := (x_1^2 + \cdots + x_P^2)^{1/2}$ and $\|X\|_\infty := \max_{i=1..P} x_i$. The $\|\cdot\|_1$-norm (summation norm) is a global norm. The global edge-cut belongs into this category (it equals half the external edges in this norm). In contrast to the $\|\cdot\|_1$-norm, the $\|\cdot\|_\infty$-norm (maximum norm) is a local norm only considering the worst value. This norm is favorable if synchronized processes are involved. The $\|\cdot\|_2$-norm (Euclidean norm) lays in between the $\|\cdot\|_1$ and the $\|\cdot\|_\infty$-norm and reflects the global situation as well as local peaks, but is omitted here.

Figure 3 displays a single frame from the 'ring' benchmark. In this benchmark, a circle and the refined area around it rotate through a narrow ring. One

Fig. 4. Numerical results of the 'ring' benchmark for Metis (blue triangles), Jostle (red squares), Party/DB (green pentagons) and MF(ϕ) (black circles).

can see that the partitions computed by Metis have quite large fringes, while Jostle and especially MF(ϕ) find smoother partition boundaries. Though the visual display of the mesh provides a first impression of the solution quality, the numerical data of all 101 frames listed in Fig. 4 reveals many more details. Looking at the first row, we can see that Metis usually allows up to 3% imbalance, while Jostle ignores this parameter and totally equalizes the partition sizes. The solutions of the MF(ϕ) heuristic usually have an imbalance of less than 3%, while the Party/DB library has some difficulties to maintain an equal distribution. The next three rows contain the metrics 'external edges', 'boundary length' and 'communication volume'. Their values are similar. The right column, displaying the sum for all partitions, reveals that Metis computes the worst results.

The three other libraries find comparable solutions, while MF(ϕ) and Party/DB show a slight advantage. This advantage is larger in the maximum norm given in the right column. One can see that the boundaries are more equally distributed between all partitions when using the latter heuristics. Row 5 displays the partition 'diameter'. Missing values indicate unconnected partitions, what can be observed several times for Metis and Jostle. MF(ϕ) cannot reach the results from Part/DB in the $\|\cdot\|_1$-norm, but this might result from the different imbalance values. Concerning the maximum norm, there is no difference between all libraries in this benchmark. The last row shows the 'migration'. Metis migrates most, and from other experiments we know that it either transfers very few or very many vertices. The values for the other libraries are smaller, and MF(ϕ) and Party/DB behave more constant than Jostle, what we could also confirm in other benchmarks. Concerning the parameters of MF(ϕ), our experiments show that the number of loops/iterations is a trade-off between the first four metrics 'external edges', 'boundary length', 'communication' and 'diameter', and the 'migration'. A good choice of ϕ depends on the amount of vertices and the number of partitions, but more theoretical analysis is needed to determine the optimal value, which is beyond the scope of this paper.

6 Conclusion

We have presented the new graph (re-)partitioning heuristic MF(ϕ), which is based on solutions of linear equations inside a learning framework. Our experiments with FEM like graphs indicate that it can find comparable or even better partitionings than state-of-the-art libraries concerning a variety of metrics, while important additional constraints like connectivity can be fulfilled.

However, due to its longer run-time, the current implementation of MF(ϕ) cannot compete with Metis or Jostle. Nevertheless, we think that further investigations are justified since a variety of techniques like the multilevel approach, faster diffusion schemes, optimized CG preconditioners or multigrid solvers are known to speed up the computations.

References

1. G. Fox, R. Williams, and P. Messina. *Parallel Computing Works!* Morgan Kaufmann, 1994.
2. G. Karypis and V. Kumar. *MeTis: A Software Package for Partitioning Unstrctured Graphs, Partitioning Meshes, [...], Version 4.0*, 1998.
3. C. Walshaw. *The parallel JOSTLE library user guide: Version 3.0*, 2002.
4. S. Schamberger. Graph partitioning with the Party library: Helpful-sets in practice. In *Comp. Arch. and High Perf. Comp., SBAC-PAD'04*, pages 198–205, 2004.
5. B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing'95*, 1995.
6. B. W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308, 1970.

7. J. Hromkovic and B. Monien. The bisection problem for graphs of degree 4. In *Math. Found. Comp. Sci. (MFCS '91)*, volume 520 of *LNCS*, pages 211–220, 1991.
8. Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *J. Par. Dist. Comp.*, 47(2):109–124, 1997.
9. C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *J. Parallel Computing*, 26(12):1635–1660, 2000.
10. B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Irregular'98*, number 1457 in LNCS, pages 218–225, 1998.
11. L. Oliker and R. Biswas. PLUM: Parallel load balancing for adaptive unstructured meshes. *J. Par. Dist. Comp.*, 52(2):150–177, 1998.
12. R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Shape-opt. mesh part. and load bal. for par. adap. FEM. *J. Parallel Computing*, 26:1555–1581, 2000.
13. S. Schamberger. On partitioning FEM graphs using diffusion. In *HPGC, Intern. Parallel and Distributed Processing Symposium, IPDPS'04*, page 277 (CD), 2004.
14. R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35:305–320, 2002.
15. S. Schamberger. A shape optimizing load distribution heuristic for parallel adaptive FEM computations. Accepted at PACT'05.
16. Y. F. Hu and R. F. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25(4):417–444, 1999.
17. O. Marquardt and S. Schamberger. Open benchmarks for load balancing heuristics in parallel adaptive finite element computations. Accepted at PDPTA'05.
18. S. Schamberger. http://www.upb.de/cs/schaum/benchmark.html.