Accelerating Shape Optimizing Load Balancing for Parallel FEM Simulations by Algebraic Multigrid*

Henning Meyerhenke Burkhard Monien Stefan Schamberger
Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik
Fürstenallee 11, D-33102 Paderborn
E-mail: {henningm|bm|schaum}@uni-paderborn.de

Abstract

We propose a load balancing heuristic for parallel adaptive finite element method (FEM) simulations. In contrast to most existing approaches, the heuristic focuses on good partition shapes rather than on minimizing the classical edge-cut metric. By applying Algebraic Multigrid (AMG), we are able to speed up the two most time consuming calculations of the approach while maintaining its large amount of natural parallelism.

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

1. Introduction

Finite Element Methods (FEM) are used extensively by engineers to analyze a variety of physical processes that can be expressed by 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 (e. g. [6]). Since the derived discretization matrices are sparse, the equations are typically solved by iterative methods such as Conjugate Gradient or Algebraic Multigrid.

Due to the very large amount of elements needed to obtain an accurate approximation of the original problem, FEM simulations have become a classical application for parallel computers. Parallelizations of numerical simulation algorithms usually follow 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 sub-domains, each being assigned to one of the processors. To minimize the overall computation time, all processors should roughly contain the same amount of elements.

Moreover, since iterative solution algorithms perform mainly local operations, i. e. data dependencies exist between neighboring elements of the mesh, the parallel algorithm mainly requires communication at the partition boundaries. Hence, these boundaries should be as small as possible due to the involved communication costs. The described problem can be expressed as a graph partitioning problem and existing state-of-the-art approaches to solve it are presented in the next section.

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 an imbalance between the processor loads and therefore delay the simulation. To avoid this, the distribution of elements needs to be rebalanced. The application is interrupted and the repartitioning problem is solved. To keep the interruption as short as possible, it is necessary to find a new balanced partitioning with small boundaries quickly, with the additional objective not to cause too many elements to change their processor. Migrating elements can be an extremely costly operation since a lot of data has to be sent over communication links and reinserted into complex data structures. Implementations solving a repartitioning problem are referred to in section 2.

In this work we present a heuristic addressing the graph partitioning as well as the repartitioning problem. While existing approaches explicitly minimize the edge-cut, our heuristic does not contain such an objective. Instead, we apply a diffusive process inside a learning framework to determine well shaped par-

*This work is supported by German Science Foundation (DFG) Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo) and DFG Collaborative Research Centre SFB-376.

1-4244-0054-6/06/$20.00

tition boundaries. This yields fewer boundary vertices and therefore reduces the resulting communication volume. We prove the convergence of the applied diffusion scheme FOS/C and show that its solution can also be obtained by solving a system of linear equations. The heuristic requires to repeatedly solve systems that only differ in their right hand side. We therefore apply an Algebraic Multigrid solver with the advantage that its hierarchy depends only on the matrix and needs only to be constructed once. In addition, we demonstrate that we can use this hierarchy to obtain better solutions with the same number of learning steps.

The remaining part of this paper is organized as follows. The next section contains a brief introduction to general purpose graph partitioning and repartitioning algorithms. Section 3 describes our shape optimizing approach and also gives some background on the applied diffusion schemes. In section 4 we introduce the Algebraic Multigrid (AMG) method, explain its integration into the learning framework, and provide some implementation details. We present some of our experiments with the resulting heuristic in section 5 before we give a short conclusion.

2. Related Work

In this section we provide some background knowledge and give a short introduction to state-of-the-art, general purpose load balancing schemes. General purpose means that these libraries work on graphs only, for example on a dual graph of an unstructured mesh describing the data dependencies, and are not provided with additional problem related information. Our focus lies on the implementations included in the evaluation presented in section 5. For a broader overview we refer the reader to e. g. [22]. Since the load balancing algorithms are derived from heuristics solving the graph partitioning problem which must be solved to obtain an initial partitioning, we start with a description of the latter.

2.1. Graph Partitioning Heuristics

Due to the large graph sizes, state-of-the-art graph partitioning libraries like Metis [12], Jostle [26] and Party [17] usually follow the multilevel scheme [8]. Rather than 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. Then, vertices of the original graph are assigned to partitions according to their representatives in the smaller instance. The obtained partitioning is then further enhanced by a local refinement heuristic. Instead of applying a global heuristic on the first smaller instance, the described process can be recursively repeated, until in the lowest level only a very small graph remains. Hence, a very basic global heuristic can be applied or 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.

To create a smaller, similar graph for the next level of the multi-level scheme, a matching algorithm is applied and the matched vertices are combined. Several different variations of matching algorithms serving this purpose have been tested and compared (e. g. [16]).

The most important phase is the local refinement. After the vertices are partitioned according to their representatives in the smaller graph, this phase tries to improve the current partitioning. In most libraries, the local refinement process is based on the Fiduccia-Mattheyses method [5] (FM), a run-time optimized version of the Kernighan-Lin (KL) heuristic [13]. Vertices are exchanged between partitions and the cost reduction is recorded. After every vertex has been moved once, the solution with the best gain is chosen. This is repeated several times until no further improvements can be found.

In contrast to other implementations, the local refinement algorithm in Party is based on theoretical analysis finding upper bounds for the bisection width of regular graphs [10, 15]. Instead of moving single vertices, the Helpful-Sets (HS) heuristic (e. g. [17]) exchanges whole vertex sets between the partitions. However, this approach has only been successfully applied to bi-sectioning yet.

2.2. Parallel Load Balancing Heuristics

To address the load balancing problem, distributed versions of the libraries Metis and Jostle have been developed. Both of them apply about the same multilevel techniques as their respective single processor version, but some phases of the computation need special attention due to their sequential nature. As an example, a coloring of the graph's vertices is used by the parallel library ParMetis [21] to ensure 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 [27] maps each sub-domain to a single processor and vertices which migrate do so already during the computa-

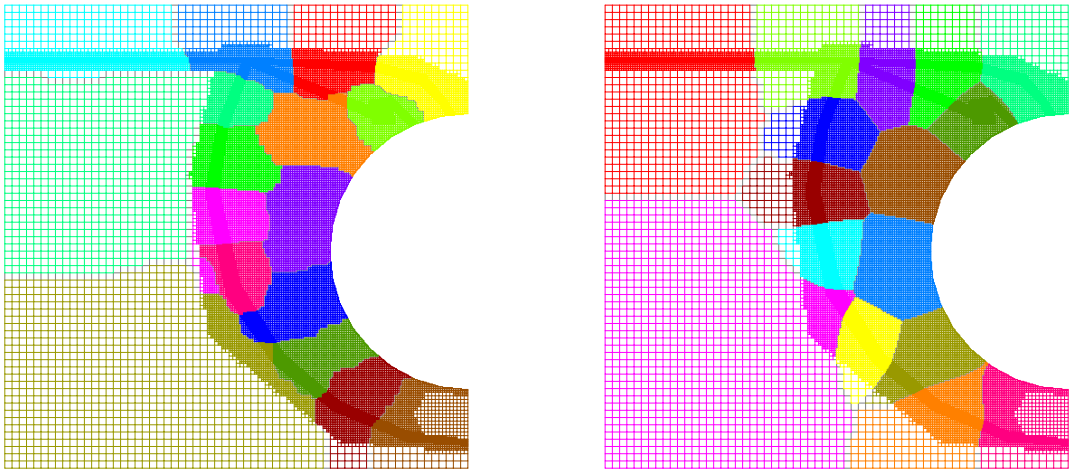

Figure 1. Partitioning the shock graph after 9 refinement steps into 16 partitions. Metis (left) computes a solution with edge-cut 1226 and 2082 boundary vertices, while the shape optimizing approach (right) finds a partitioning with edge-cut 1168 and only 1795 boundary vertices.

tion of the repartitioning. Furthermore, Jostle, apart from the edge-cut minimization, seems to incorporate a shape optimization presented in [28]. However, since we do not have access to the source code of the latest version, we can only make assumptions here. Usually, Metis is very fast while Jostle takes longer but often computes better shaped partitions.

3. Shape Optimized Partitioning

The objective followed by the aforementioned libraries is the edge-cut minimization. However, it is known that this is not necessarily the best metric to follow. A more appropriate metric is the number of boundary vertices. It models the resulting communication volume more accurately, but is unfortunately harder to optimize [7].

This section describes our shape optimizing load balancing approach in more detail. Shape optimized partitions typically exhibit few boundary vertices. We first explain the learning framework and then present our realization of its operations via a diffusion scheme. To get a first impression, figure 1 illustrates a partitioning computed with the edge-cut minimizing library Metis and a solution obtained with the new approach.

3.1. 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 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 to a partition 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 2 illustrates the three main operations.

The three steps can be implemented in several ways. One idea is to choose the initial vertices randomly. To grow the partitions, a breadth-first like algorithm is started from every seed. During this process the partitions alternately acquire one of their free neighbor vertices until all vertices are assigned. Then, the vertex with the minimal maximal distance to all other vertices of the same partition becomes the new seed. However, this approach shows several deficiencies. The initial placement of the partitions may be very bad so that many iterations are required. Also, the partition sizes vary extremely. The time spent on finding new seeds is quite long since a breadth-first-search has to be solved for every vertex. Moreover, the partition quality is often unsatisfactory. Another important disadvantage is that the growth phase cannot be parallelized because vertices are assigned serially and earlier assignments have a great impact on later decisions.

A second approach is described in [3] and has been implemented in a former version of the FEM simulation tool PadFEM. Here, the seeds are distributed more evenly over the graph. To grow the partitions, the smallest partition with at least one adjacent unassigned vertex grabs the vertex with the smallest Eu-

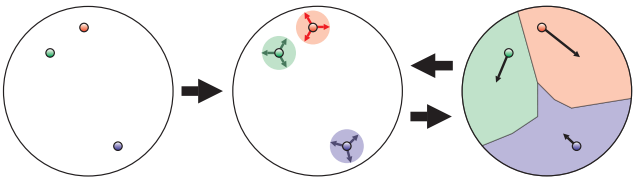

Figure 2. The three operations of the learning bubble framework: Init: Determination of initial seeds for each partition (left). Grow: Growing around the seeds (middle). Move: Movement of the seeds to the partition centers (right).

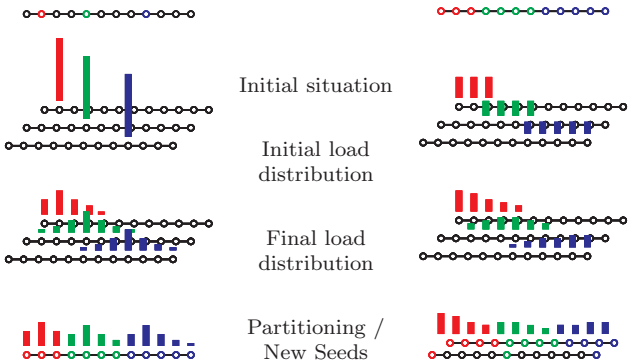

Figure 3. Schematic view: Placing load on single vertices (left) or a partition (right), the diffusion process and the mapping of the vertices to the partitions according to the load.

clidean distance to its seed. The new seed of a partition is determined as the vertex for which the sum of Euclidean distances to all other vertices of the same partition is minimal. To find this vertex quickly, some estimation is used.

This approach solves some of the problems we have seen before. As mentioned, the initial seed distribution is improved. Since the smallest possible partition receives the next vertex, more attention is paid to the balance and the determination of the center has been improved to work faster, too. By including coordinates in the choice of the next vertex, the partitions are usually also geometrically well shaped (and connected), which is the main goal of this approach. Other quality metrics are not considered.

However, by relying on vertex coordinates, this approach is only applicable if these are provided, and sometimes the Euclidean distance does not coincide at all with the path length between vertices, in particular if an FEM mesh contains "holes". It is a general problem when working with coordinates and occurs more heavily for example in space-filling-curve based partitionings [20]. The experiments made in [3] also reveal that the selection mechanism, though improved by preferring under-weighted partitions, does still not lead to sufficiently balanced domains. Hence, to fix this, some additional computations are added after the last bubble iteration. Concerning a possible parallelization, the situation stays the same as described before because the selection process of the vertices is still strictly serial.

3.2. Diffusive Mechanisms

To overcome the problems mentioned in the previous subsection, the growth process in [18] is realized via some iterations of a diffusive process. Note that diffusion in graphs has been studied very well because it can be applied to solve load balancing problems in various scenarios (e. g. [1, 4]). 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 vertex sets that possess a high number of internal and a small number of external edges. Furthermore, diffusion possesses a large amount of parallelism since it performs local operations only.

The growth and seed determination process is illustrated in figure 3. Given a seed vertex for each partition (left), we place load on the seeds and use a diffusive process to have it spread into the graph. This is performed independently for every partition. After the load is distributed, we assign each vertex to that partition it has obtained the highest load amount from.

The next step (right) does not place load on a single vertex but distributes it evenly among all vertices of the given partitions. After performing the diffusion process, the resulting load distribution can either be used for an optional consolidation step or for contracting the partitions to the seed vertices of the next iteration. A consolidation again assigns the vertices to partitions according to the highest load as in the previous step. This further improves the partition shapes. During a contraction, for each partition the vertex containing the highest load becomes its new seed.

The applied diffusion process must possess two important properties. To guarantee connected partitions, it is necessary that vertices that are closer to the partition center receive more load than those further away. The final load distribution therefore must be "hill-like". Furthermore, load must not only be distributed according to the distance in the graph. Instead, the connectivity, meaning the number of paths between vertices, plays a vital role. If more load diffuses into dense regions of the graph, the partition centers are directed into these areas during the learning process. Hence,

the partition boundaries tend to be in sparser regions, which reduces the number of boundary vertices and therefore improves the partition quality.

As mentioned, a first implementation presented in [18] applies FOS. Since FOS converges toward a fully balanced load situation, it must be interrupted at some point to preserve the hill-like structure. Though it is possible to determine a suitable number of iterations, it is rather difficult and since the interruption point depends both on the graph and on the number of partitions, we were not able to find a general rule for this, which results in an unreliable implementation.

To avoid this, [19] presents the disturbed diffusion scheme FOS/A. It is based on FOS, but in each iteration load is shifted back from all non-empty vertices to the seed. This preserves the hill-like structure in the final state. Although all experiments indicate that this diffusion scheme converges, no proof has been found yet. Furthermore, due to the disturbance, FOS/A performs even slower than FOS.

In [14], a hill-like load distribution is computed by solving a system of linear equations. This approach adds an extra vertex to the graph that is connected to every other vertex. Varying the capacity ϕ of the edges incident to the extra vertex thereby controls the spreading of the load.

In the following, we propose to omit the extra vertex. We show that running the algorithm with the according parameter setting $\phi = 0$ is indeed equivalent to applying a disturbed diffusion scheme we call FOS/C. In contrast to FOS/A, the new scheme does allow negative load on the vertices. We provide the theoretical background and prove the convergence of FOS/C. Section 4 then presents an efficient way to solve the systems of linear equations that was prohibited before due to the extra vertex.

3.3. Prerequisites

To show some properties of the new diffusion scheme, we first need to introduce some basic notations. Let $G = (V, E)$ be an unweighted, connected graph with $n = |V|$ vertices. The incidence matrix $\mathbf{A}$ of G is $\mathbf{A} \in \{-1, 0, +1\}^{|V|\times|E|}$. $\mathbf{A}$ 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 Laplacian matrix $\mathbf{L} \in \mathbb{Z}^{|V|\times|V|}$ is defined as $\mathbf{L} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$. $\mathbf{L}$ is symmetric, positive semidefinite, and of rank $|V|-1$. The matrix $\mathbf{M} = \mathbf{I}-\alpha\mathbf{L}$ (with $0 < \alpha < 1$ suitably chosen, e. g. $\alpha = 1/(\mathrm{maxdeg}(G) + 1)$) is a *diffusion matrix*, since it is nonnegative, symmetric and doubly stochastic, and in case G is bipartite, at least one diagonal entry of $\mathbf{M}$ is positive. Then, $1 = \mu_1 > \mu_2 \geq \cdots \geq \mu_n > -1$ are the eigenvalues of $\mathbf{M}$ and $\mathbf{1}$ is the eigenvector corresponding to μ_1.

We now formally introduce the First Order Diffusion Scheme (FOS) from [1].

Definition 1 (FOS). *Given a connected graph $G = (V, E)$ and a suitable constant α. In each iteration, the first order scheme (FOS) performs the operations:*

$$\begin{aligned} f^{(i)}_{e=(u,v)} &= \alpha \cdot \left(w^{(i)}_u - w^{(i)}_v\right) \\ w^{(i+1)}_v &= w^{(i)}_v - \sum_{e=(v,*)} f^{(i)}_e \end{aligned}$$

In matrix notation, FOS can be written as

$$w^{(i+1)} = \mathbf{M}w^{(i)}$$

where $\mathbf{M} = \mathbf{I} - \alpha\mathbf{L}$ is the diffusion matrix.

As already mentioned, FOS converges toward the equally balanced load situation. Furthermore, we know that the computed flow is minimal according to the $||\cdot||_2$-norm [2].

Lemma 1. *Let $\mathbf{M} = \mathbf{I} - \alpha\mathbf{L}$ be the FOS diffusion matrix, $w^{(0)}$ the initial and $\overline{w}$ the balanced load situation. Then, the first order scheme $w^{(i+1)} = \mathbf{M}w^{(i)}$ converges to $\overline{w}$.*

$\mathbf{L}$ does not have full rank. Hence, the existence of a solution depends on the right hand side of the linear equation.

Lemma 2. *The equation $\mathbf{L}w = d$ has a solution (and then infinitely many), iff $d \perp \mathbf{1}$.*

The next lemma states that the $||\cdot||_2$-minimal flow can be computed by solving a linear system.

Lemma 3. *Consider the quadratic minimization problem*

$$\min! \ ||f||_2 \text{ with respect to } \mathbf{A}f = d$$

Provided that $d \perp \mathbf{1}$, the solution to this problem is given by

$$f = \mathbf{A}^T w, \text{ where } \mathbf{L}w = d$$

To prove the convergence of FOS/C, we require the following observation.

Lemma 4. *Let $\mathbf{M}$ be a diffusion matrix and d be a vector perpendicular to $\mathbf{1}$. Then,*

$$\lim_{i\to\infty}(\mathbf{I} + \mathbf{M} + \mathbf{M}^2 + \cdots + \mathbf{M}^i)d = (\mathbf{I} - \mathbf{M})^{-1}d$$

Proof. Recall that $\mathbf{1}$ is an eigenvector to the simple eigenvalue 1 of $\mathbf{M}$. Since $d \perp \mathbf{1}$, i. e. $\sum_{j=1}^{n} d_j = 0$, it follows that $\lim_{i\to\infty} \mathbf{M}^{i+1}d = 0$. Hence,

$$\begin{aligned} & \lim_{i\to\infty}(\mathbf{I} - \mathbf{M})(\mathbf{I} + \mathbf{M} + \mathbf{M}^2 + \cdots + \mathbf{M}^i)d \\ = \ & \lim_{i\to\infty}(\mathbf{I} - \mathbf{M}^{i+1})d = \lim_{i\to\infty} d - \mathbf{M}^{i+1}d \\ = \ & d \end{aligned}$$

Therefore, $(\mathbf{I} + \mathbf{M} + \mathbf{M}^2 + \cdots + \mathbf{M}^i)$ is the inverse to $(\mathbf{I} - \mathbf{M})$ for $i \to \infty$ and any vector d perpendicular to $\mathbf{1}$, so that the claim follows. □

3.4. Diffusion with Constant Draining

We are now introducing a new diffusion scheme. This scheme is based on FOS, but is disturbed in every iteration. In contrast to FOS/A, this disturbance is not restricted to the non-empty vertices, but performed on all vertices.

In contrast to genuine FOS, the FOS/C scheme performs two operations in each iteration. While the first one is the original diffusion step, the second step introduces a disturbance by shifting a small load amount $\delta > 0$ from all vertices of the graph to some selected source vertices $S \subset V$. This disturbance can be described by the drain vector $d \in \mathbb{R}^n$, which is defined as

$$d_v = \begin{cases} -\delta & : v \notin S \\ \delta \cdot |V|/|S| - \delta & : \text{otherwise} \end{cases}$$

The vector d is added to the load vector resulting from the diffusion step. Note that, since $\langle d, \mathbf{1} \rangle = 0$, this does not change the total amount of system load.

Definition 2 (FOS/C). *Given a connected graph $G = (V, E)$ and a suitable constant α. Let $\delta > 0$ be the drain constant and d the corresponding drain vector. Let $S \subset V$ be the set of source vertices. In iteration i, $w_v^{(i)}$ denotes the load on vertex v and $f_e^{(i)}$ the flow over edge e. Let $w^{(0)}$ represent the initial load situation. In each iteration i, the FOS/C scheme performs the computations:*

$$\begin{aligned} f_{e=(u,v)}^{(i)} &= \alpha \cdot \left(w_u^{(i)} - w_v^{(i)} \right) \\ w_v^{(i+1)} &= w_v^{(i)} - \sum_{e=(v,*)} f_e^{(i)} + d_v \end{aligned}$$

In matrix notation, FOS/C can be written as

$$w^{(i+1)} = \mathbf{M} w^{(i)} + d.$$

Theorem 1 (Convergence of FOS/C). *The FOS/C scheme converges for any arbitrary initial load vector $w^{(0)}$.*

Proof. Repeatedly applying the diffusion matrix to the initial load vector $w^{(0)}$, we obtain

$$\begin{aligned} w^{(1)} &= \mathbf{M} w^{(0)} + d \\ w^{(2)} &= \mathbf{M} w^{(1)} + d = \mathbf{M}(\mathbf{M} w^{(0)} + d) + d \\ &= \mathbf{M}^2 w^{(0)} + (\mathbf{M} + \mathbf{I}) d \\ &\vdots \\ w^{(i)} &= \mathbf{M}^i w^{(0)} + (\mathbf{M}^{i-1} + \cdots + \mathbf{M} + \mathbf{I}) d \end{aligned}$$

Due to lemma 4, this yields

$$\begin{aligned} w^{(\infty)} &= \mathbf{M}^\infty w^{(0)} + (\mathbf{I} - \mathbf{M})^{-1} d \\ &= \mathbf{M}^\infty w^{(0)} + (\alpha \mathbf{L})^{-1} d \end{aligned}$$

□

Since $\mathbf{M}^\infty w^{(0)}$ is the evenly balanced load that FOS computes, one can see that the solution of the disturbed scheme FOS/C could also be determined by solving a system of linear equations. In fact, in the converged state, all load that is moved back onto the source vertices has to be sent back in one iteration step. This means that the solution of FOS/C is equivalent to computing a $||\cdot||_2$-minimal flow from the source vertices into the graph.

Corollary 1. *The convergence state $w^{(*)}$ of FOS/C can be characterized as:*

$$\begin{aligned} w^{(*)} &= \mathbf{M} w^{(*)} + d \\ \Leftrightarrow (\mathbf{I} - \mathbf{M}) w^{(*)} &= d \\ \Leftrightarrow \alpha \mathbf{L} w^{(*)} &= d \end{aligned}$$

Hence, the convergence state can be determined by solving the system of linear equations $\mathbf{L}w = d$, where $w = \alpha w^{(*)}$.

Since the solution of $\mathbf{L}w = d$ is only determined up to a constant, we choose the one with $\sum_{v \in V} w_v^{(*)} = 0$. This normalization also ensures that the load distributions computed for each partition have a common reference point and are therefore comparable.

3.5. The Bubble-FOS/C Heuristic

The resulting algorithm is sketched in figure 4. 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 5-7) applying the proposed mechanism. Note that in either case π only contains a single seed vertex for each partition when entering line 8.

Next, we determine a partitioning (lines 8-10). Additional consolidations can be performed (lines 11-14). Furthermore, these are used for balancing by scaling the height functions w_p (line 15). This approach can quickly find almost balanced solutions in most cases. If necessary, we perform an additional greedy balancing operation (line 16) to ensure a certain partition size. For this, 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.

Depending on the quality of the initial solution, it is advisable to repeat the learning process several times. Before returning the partitioning π, we migrate vertices

```
Algorithm Bubble-FOS/C(G, π, 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 L w_p = d_p and normalize w_p
07     π(v) = { p : w_p(v) ≥ w_p(u) ∀u ∈ V
               { −1 : otherwise
08   parallel for each partition p /* compute partitioning */
09     solve L w_p = d_p and normalize w_p
10   π(v) = p : w_p(v) ≥ w_q(v) ∀q ∈ {1, ..., P}
11   in each iteration i        /* optional consolidation with ...*/
12     parallel for each partition p
13       solve L w_p = d_p and normalize w_p
14     π(v) = p : w_p(v) ≥ w_q(v) ∀q ∈ {1, ..., P}
15     scale-balance(π)                          /* ... scale balancing */
16   greedy-balance(π)                           /* greedy balancing */
17 return smooth(π)                              /* smoothing */
```

Figure 4. Sketch of the Bubble-FOS/C heuristic.

```
Algorithm V-cycle(L_f, w, d)
01   if L_f is coarse enough then
02     return w = DirectSolve(L_f, w, d);
     1: Relaxation:
03   for i = 1 to pre-steps do
04     w = Presmooth(L_f, w, d);
     2: Coarse grid correction:
     /* Restriction of residual: */
05   r = R(d − L_f w);
     /* Recursive call with coarse matrix: */
06   e = V-Cycle(L_c, e, r);
     /* Interpolation of coarse error: */
07   w = w + P e;
     3: Relaxation:
08   for i = 1 to post-steps do
09     w = Postsmooth(L_f, w, d);
10   return w;
```

Figure 5. Recursive V-Cycle Scheme

if the number of their neighbors in another partition is larger than the number in their own partition (line 17). This compensates numerical imprecisions that occur during the flow computation and further smoothes the partition boundaries. However, it might lead to a slightly higher imbalance.

4. Algebraic Multigrid

In this section we describe the Algebraic Multigrid (AMG) method and its application within the Bubble framework. It serves as solver for sparse linear systems and provides the hierarchy for multilevel learning. We present implementation details, in particular the algorithms and parameters for the utilized AMG scheme.

4.1. Fundamentals of AMG

Most work performed by the Bubble-FOS/C algorithm consists in solving the linear system $\mathbf{L}w = d$ for each of the P partitions. These could be solved by the very popular Conjugate Gradient algorithm (CG) algorithm (which is also suitable for symmetric positive semi-definite systems as long as the right-hand side is consistent [11]).

However, the convergence of CG tends to slow down considerably when the linear systems to solve become larger. Furthermore, recall that only d differs for each partition, since the matrix $\mathbf{L}$ only depends on the graph and therefore is the same for all P systems. Hence, it should be advantageous to exploit this fact, e. g. by applying Algebraic Multigrid as solver.

Multigrid methods (e. g. [25]) are among the fastest iterative solvers and preconditioners for large linear systems derived from a wide class of PDEs. They are based on the observation that relaxation methods such as Jacobi or Gauss-Seidel can only eliminate high-frequency error components in the solution vector effectively. Therefore, one uses a hierarchy of matrices whose size decreases from one hierarchy level to the next one. The smooth error is passed recursively to the next level, where its low-frequency components become oscillatory and can be smoothed by relaxation methods again.

AMG is an extension of classical multigrid to cases where no geometric information is available. The main difference between the two methods is the construction of the hierarchy. While classical multigrid typically builds it by successive mesh refinements, AMG uses a top-down coarsening approach. For this, only the matrix corresponding to the finest mesh is necessary. Coarsening a matrix $\mathbf{L} = \mathbf{L_f}$ to obtain the coarse matrix $\mathbf{L_c}$ (f = fine, c = coarse) of the next hierarchy level consists of three main steps: First one determines the coarse vertices, which must be able to interpolate those nodes accurately which are not retained within the coarse matrix. Then one determines interpolation weights and sets up the prolongation matrix $\mathbf{P}$ and the restriction matrix $\mathbf{R} = \mathbf{P}^T$. The coarse matrix can now be computed as $\mathbf{R}\mathbf{L_f}\mathbf{P} = \mathbf{L_c}$. This process is repeated recursively until the coarsest matrix is small enough to be solved efficiently by direct methods.

After the hierarchy construction in the setup phase, the actual solution process is performed by an algorithm which consists of the following main operations: presmoothing, restriction, solving the coarse problem recursively, interpolating the coarse solution, and postsmoothing (cf. the V-cycle in figure 5).

Although AMG has been initially developed for M-matrices [24], it can be applied to our problem as well:

Lemma 5. *The equation* $\mathbf{L}w = d$ *describing the convergence state can be solved by an AMG scheme.*

Proof. We have already seen that there exists a solution for the equation $\mathbf{L}w = d$ because $d \perp \mathbf{1}$. It can be easily verified that also $r = d - \mathbf{L_f}w \perp \mathbf{1}$ holds.

On the coarse grid we need to solve $\mathbf{L_c}e = r$, where $\mathbf{L_c} = \mathbf{R}\mathbf{L_f}\mathbf{P}$. Since the column sum of $\mathbf{R}$ is 1, r is perpendicular to $\mathbf{1}$. Consequently, as $\mathbf{L_c}$ is again symmetric positive semi-definite, the coarse problem has a solution, too, and AMG is applicable. □

Since the AMG hierarchy built for one linear system can be reused for all others with the same matrix as well, we can expect superior run-times compared to CG for non-trivial system sizes.

4.2. Learning on the AMG hierarchy

In experiments from e. g. [14] the learning process is performed on the original graph only. If the initial partitioning is undefined or of low quality, many iterations are required to find a good solution.

Therefore, we adopt the multilevel scheme presented in section 2. Rather than computing an additional hierarchy based on matchings, we use the existing AMG hierarchy. This is possible because each matrix in this hierarchy corresponds to a (possibly edge weighted) graph, and two graphs of consecutive levels have a similar structure. We perform Bubble-FOS/C as a refinement heuristic on each level. This reduces the number of required learning iterations and therefore the runtime considerably.

4.3. Implementation

We have implemented our algorithm in C++ and parallelized the most time consuming parts – solving P linear systems concurrently and two matrix-matrix multiplications for each AMG hierarchy level – with POSIX threads.

After converting the graph to its Laplacian matrix, we construct the corresponding AMG hierarchy. Here we use PMIS coarsening [23] to reduce the number of nodes in the next level substantially, so that the number of created levels remains modest. In cases where PMIS coarsens too much, we neglect its result and apply CLJP coarsening [9] instead. While standard CLJP coarsening reduces the weight of vertices whose influence has been taken into account by 1, we vary this value adaptively to control the coarse grid size. Currently, we use a simple M-matrix interpolation from [24]. Alternative schemes exist as discussed in sec. 6.

Table 1. Graphs used in this paper.

Graph	$\|V\|$	$\|E\|$	origin
biplane9	21701	42038	FEM 2D
crack	10240	30380	FEM 2D
crack (dual)	20141	30043	FEM 2D dual
grid100x100	10000	19800	FEM 2D
stufe10	24010	46414	FEM 2D
shock9	36476	71290	FEM 2D
whitacker	9800	28989	FEM 2D
whitacker (dual)	19190	28581	FEM 2D dual

The algorithm then follows the multilevel paradigm by starting the computation on the lowest hierarchy level. On each level, the Bubble-FOS/C algorithm is applied and its partitioning result is interpolated to the next level according to the respective prolongation matrix $\mathbf{P}$. All linear systems are solved by Full Multigrid V-cycles, which join the concept of nested iteration with V-cycles [25], until the desired error tolerance is reached. A standard CG implementation serves as the direct solver on the lowest level inside the V-Cycle.

5. Results

In this section we present some of our experiments executed on a 4-processor Opteron (2.2 GHz, 1 MB cache) machine running Linux (SMP-Kernel 2.4.21). As compiler we use gcc 3.4.1 with level 2 optimization. The included test set comprises eight FEM graphs with a modest number of vertices as listed in table 1. We restrict our presentation to 12-partitionings of two-dimensional FEM meshes.

5.1. Metrics

Concerning the quality of a partitioning, a number of metrics are possible. The traditional one is the edge-cut, i. e. the number of edges between different partitions, but it is known that this usually does not model the real costs. Therefore, we list the total number of boundary vertices as a much more accurate measure of communication costs [7]. Furthermore, the quality of a partitioning depends on its balance. A less balanced solution does not necessarily cause problems during the computation, but the overall run-time of the underlying FEM program typically degrades because processors with a smaller amount of work have to wait for those with higher load. Moreover, imbalance allows other metrics to decrease further and makes comparisons less meaningful.

5.2. Experiments

At first, we compare our new approach with a modified version of the heuristic from [14]. For the latter, we

Table 2. Comparison between the solutions applying the CG solver without a learning hierarchy and the AMG approach with learning hierarchy.

Graph	CG				AMG			
	Time		Cut	Boundary	Time		Cut	Boundary
	1 cpu	4 cpu			1 cpu	4 cpu		
biplane9	61.15 s	26.04 s	774	1136	18.27 s	7.10 s	672	955
crack	15.93 s	5.57 s	1157	1142	3.98 s	1.59 s	1017	1004
crack (dual)	53.12 s	20.96 s	489	949	16.42 s	6.00 s	447	865
grid100x100	8.62 s	2.62 s	684	1000	5.54 s	2.26 s	575	949
stufe10	73.73 s	30.87 s	769	1156	22.83 s	8.73 s	574	725
shock9	138.53 s	54.73 s	1137	1673	40.41 s	15.21 s	961	1480
whitacker	12.20 s	4.62 s	984	970	3.68 s	1.60 s	966	957
whitacker (dual)	47.97 s	18.68 s	488	967	13.66 s	5.43 s	493	973

Table 3. Comparison of the solutions computed by Metis, Jostle and the shape optimizing approach using the AMG solver and the learning hierarchy.

Graph	Metis			Jostle			Bubble-FOS/C		
	Time	Cut	Boundary	Time	Cut	Boundary	Time	Cut	Boundary
biplane9	0.03 s	670	1142	0.15 s	647	1104	7.10 s	672	955
crack	0.02 s	1041	1030	0.06 s	1031	1018	1.59 s	1017	1004
crack (dual)	0.02 s	466	919	0.08 s	450	893	6.00 s	447	865
grid100x100	0.03 s	584	1006	0.09 s	549	992	2.26 s	575	949
stufe10	0.02 s	570	948	0.15 s	546	919	8.73 s	574	725
shock9	0.07 s	1010	1663	0.19 s	909	1665	15.21 s	961	1480
whitacker	0.01 s	1005	992	0.11 s	966	953	1.60 s	966	957
whitacker (dual)	0.01 s	528	1048	0.11 s	515	1027	5.43 s	493	973

omit the extra vertex ($\phi = 0$) and reduce the number of learning steps significantly to speed up the computation. The linear systems are solved by a standard CG implementation and learning is only performed on the original graph.

For the AMG approach, we perform $1 + level$ learning steps and consolidations, respectively, while the CG based algorithm is limited to one operation each on the highest level. As can be seen from table 2, the new approach is about three times faster on average than the old one. Moreover, in almost all cases it attains fewer cut-edges and boundary vertices. Note that the heuristic presented in [14] delivers better solution qualities than presented here for the CG approach when a larger number of learning iterations are performed. The current straightforward implementation without the use of scientific libraries nor processor bound threads achieves a speedup of about 2.5 on four CPUs.

The comparison with the state-of-the-art sequential libraries Metis and Jostle shows that these are about two orders of magnitude faster than our thread-parallelized algorithm (table 3). Yet, this run-time investment is supposed to pay off whenever partitionings of significantly higher quality with respect to the number of boundary vertices are found. Table 3 shows that the Bubble-FOS/C heuristic succeeds in most cases to produce comparable edge-cut values and – more importantly – better numbers of boundary vertices. While Jostle obtains fewer boundary vertices for the graph whitacker, our approach delivers the best results in all other displayed cases concerning this metric. The balance values of the partitionings are not shown explicitly because all partitioners stay within the predefined range of 3% imbalance.

These values confirm the results from previous similar approaches (e. g. [14], where the Bubble framework with diffusive growing mechanisms has also been used for load balancing of dynamically changing meshes).

6. Conclusion and Future Work

In this paper we have proposed an alternative method to compute shape optimized graph partitionings. Compared to existing state-of-the-art libraries, it is often able to find solutions with a lower number of boundary vertices. Unfortunately, its current implementation requires a long run-time.

We have introduced a new diffusion scheme and show that it converges toward a solution that can also be computed by solving a system of linear equa-

tions. Replacing the diffusion scheme by FOS/C and the CG solver by the Algebraic Multigrid Method, we can speed up the involved flow computations substantially. Furthermore, the constructed AMG hierarchy can be applied to improve the learning process in terms of time and quality without the need to compute and store a separate matching hierarchy.

6.1. Future Work

A number of possible enhancements remain. First of all we think that the convergence of AMG can be improved by interpolation schemes [23] more suitable to our coarsening algorithms. Better code optimization and existing mathematical libraries as well as binding the threads to specific CPUs to enhance cache efficiency can further speed up the computations. Extending the implementation to run on distributed memory machines would allow to utilize more processors.

An important concern is the required memory usage which grows quadratically in the number of partitions. However, one can observe that the linear equations do not need to be solved with full precision on the whole graph. Reducing the uninteresting parts by e. g. adaptive coarsening, one could significantly reduce the memory usage as well as the run-time.

References

[1] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Parallel and Distributed Computing*, 7(2):279–301, 1989.

[2] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.

[3] 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.

[4] R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35:305–320, 2002.

[5] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Design Automation Conf.*, 1984.

[6] G. Fox, R. Williams, and P. Messina. *Parallel Computing Works!* Morgan Kaufmann, 1994.

[7] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Irregular'98*, number 1457 in LNCS, pages 218–225, 1998.

[8] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing'95*, 1995.

[9] V. E. Henson and U. Meier-Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Appl. Numer. Math.*, 41(1):155–177, 2002.

[10] 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.

[11] E. F. Kaasschieter. Preconditioned conjugate gradients for solving singular systems. *J. of Computational and Applied Mathematics*, 24(1-2):265–275, 1988.

[12] G. Karypis and V. Kumar. *MeTis: A Software Package for Partitioning Unstrctured Graphs, Partitioning Meshes, [...], Version 4.0*, 1998.

[13] B. W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308, 1970.

[14] H. Meyerhenke and S. Schamberger. Balancing parallel adaptive fem computations by solving systems of linear equations. In *Proc. Euro-Par 2005*, number 3648 in LNCS, pages 209–219, 2005.

[15] B. Monien and R. Preis. Upper bounds on the bisection width of 3- and 4-regular graphs. In *Mathematical Foundations of Computer Science (MFCS '02)*, volume 2136 of *LNCS*, pages 524–536, 2002.

[16] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. *Par. Comp.*, 26(12):1609–1634, 2000.

[17] S. Schamberger. Graph partitioning with the Party library: Helpful-sets in practice. In *Comp. Arch. and High Perf. Comp.*, pages 198–205, 2004.

[18] S. Schamberger. On partitioning FEM graphs using diffusion. In *HPGC, Intern. Par. and Dist. Processing Symposium, IPDPS'04*, page 277 (CD), 2004.

[19] S. Schamberger. A shape optimizing load distribution heuristic for parallel adaptive FEM computations. In *Parallel Computing Technologies, PACT'05*, number 2763 in LNCS, pages 263–277, 2005.

[20] S. Schamberger and J. M. Wierum. Graph partitioning in scientific simulations: Multilevel schemes vs. space-filling curves. In *Par. Comp. Technologies, PACT'03*, number 2763 in LNCS, pages 165–179, 2003.

[21] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *J. Par. Dist. Comp.*, 47(2):109–124, 1997.

[22] K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high performance scientific simulations. In J. D. et al., editor, *The Sourcebook of Parallel Computing*. Morgan Kaufmann, 2002.

[23] H. D. Sterck, U. Meier-Yang, and J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. Technical Report UCRL-JRNL-206780, Lawrence Livermore National Laboratory, 2004.

[24] K. Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C. W. Oosterlee, and A. Schüller, editors, *Multigrid*, pages 413–532. Academic Press, London, 2000. Appendix A.

[25] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London, 2000.

[26] C. Walshaw. *The parallel JOSTLE library user guide: Version 3.0*, 2002.

[27] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *J. Parallel Computing*, 26(12):1635–1660, 2000.

[28] C. Walshaw, M. Cross, R. Diekmann, and F. Schlimbach. Multilevel mesh partitioning for opt. domain shape. *J. High Perf. Comp. App.*, 13(4):334–353, 1999.