Preprocessing for 2D FE–BE Domain Decomposition Methods*

G. Haase M. Kuhn
Institute of Analysis and Computational Mathematics
Johannes Kepler University Linz, Altenberger Str. 69
A-4040 Linz, Austria.
(ghaase@numa.uni-linz.ac.at, kuhn@numa.uni-linz.ac.at)

Abstract

The non-overlapping Domain Decomposition (DD) method is a powerful tool for the coupling of Finite Element (FE) and Boundary Element (BE) methods. Moreover it provides a natural basis for constructing efficient parallel solvers. However, both, the efficiency and the robustness of DD–solvers depends heavily on the underlying decomposition of the domain of interest into subdomains. In this paper, we introduce the **A**daptive **D**omain **D**ecomposition **Pre**processor **ADDPre** which realizes an automatic decomposition of the computational domain into p subdomains, where p is the number of processors to be used. We discuss the codes being involved, the algorithms which they are based on and the data–formats being used for describing the decomposition of the problem.

Numerical examples, demonstrating the performance of the preprocessor are presented.

Key words: Domain Decomposition Methods, Preprocessing, Parallel Solvers for Partial Differential Equations, Coupling of FEM and BEM.

AMS subject classifications: 65N55, 65N22, 65F10, 65N30, 65N38, 65Y10, 65Y05, 05C90.

1 Introduction

Nowadays, Domain Decomposition (DD) and related algorithms are of great interest, since they are the basic tool for constructing algorithms which are well suited to run on Multiple Instruction Multiple Data (MIMD) parallel computers, including clusters of workstations, with message-passing. These parallel machines provide sufficient CPU power and sufficiently large storage capacity as it is necessary for the numerical simulation of complex processes. Another important aspect is that non-overlapping

*Acknowledgments: This work has been supported by the Austrian Science Fund – 'Fonds zur Förderung der wissenschaftlichen Forschung' – within the SFB F013 "Numerical and Symbolic Scientific Computing".

DD methods allow us to "marry" the advantages of the Finite Element Method (FEM) to those of the Boundary Element Method (BEM) via a unified coupled variational formulation, see [6, 14] and the references therein. Therefore, efficient parallel solvers for large systems of algebraic equations resulting from, e.g., the finite element (FE), the boundary element (BE), or a coupled FE/BE discretization of the partial differential equation, have been developed [6, 14, 15].

To achieve the inherent high parallel efficiency of the solvers it is necessary to distribute the work to the p processors of the parallel computer with a good load balance. The latter means that the total waiting time, i.e., the sum over all processors of the time that a processor has to wait idle for the others, should be small. As an example, in the FEM-DD for linear elliptic problems, almost equal numbers of nodes, elements, and coupling boundaries should be assigned to each processor. However, for DD methods which we have in mind a simple (even load balanced) distribution of the mesh is not sufficient. Instead a "well shaped" decomposition of the computational domain into subdomains is required. For this purpose we have developed the preprocessing tool *ADDPre* (Adaptive Domain Decomposition Preprocessor) which is going to be discussed in this paper.

In a first stage, the tool requires some algorithm for distributing meshes. Several such algorithms have been described in the literature. Even tools as *Chaco* [8], *TOP/DOMDEC* [3], *Jostle* [19], *Metis* [10] are available. Furthermore, the tool *Exadasy* [11] provides a general user interface allowing the application and comparison of several such tools for mesh partitioning. Here, we restrict our considerations to the the recursive spectral bisection method (rsb) [2, 17] which allows to partition meshes represented by their graphs. We will apply a generalized version that allows us to decompose a naturally given (by the material coefficients) initial decomposition consisting of P_{mat} subregions of different size into an arbitrary number p of final well-balanced subdomains. First, we determine the number of subdomains in which each subregion is to be decomposed (it needs not to be a power of 2), then we apply the modified rsb to each subregion.

Further, complicated coupling boundaries between the subdomains can reduce the efficiency of the DD based solvers mentioned above. Thus, it is desirable to smooth the coupling boundaries that separate subdomains inside a subregion. Roughly speaking, we look for "well shaped" subdomains. This goal is treated in a second stage of *ADDPre*. Hereby, the coupling boundaries that separate subregions with different coefficients remain unchanged as in the standard DD methods.

As a third aspect, adaptive refinement strategies can lead to load-imbalance on the fine grids. Therefore, we provide the possibility to generate an a-priori refined mesh in the preprocessing. If this mesh is used as an input for the spectral bisection method, there will be a quasi-static load balance on the fine grids.

In this paper, we consider a potential problem arising, e.g., from magnetostatics, as a test problem. It reads formally as follows

$$\begin{aligned} -\mathrm{div}\ (\nu(x)\nabla u(x)) &= \mathrm{S}(x) + \frac{\partial \mathrm{H_{0y}}(x)}{\partial \mathrm{x}} - \frac{\partial \mathrm{H_{0x}}(x)}{\partial \mathrm{y}}, \quad x \in \Omega &(1)\\ u(x) &= 0, \quad x \in \Gamma_D \equiv \Gamma := \partial\Omega. &(2)\end{aligned}$$

Here, Ω is a bounded domain, and ν denotes a piecewise constant coefficient function which defines an a-priori decomposition of the domain Ω by

$$\bar{\Omega} = \bigcup_{j=1}^{P_{mat}} \bar{\hat{\Omega}}_j, \quad \text{with} \quad \hat{\Omega}_i \cap \hat{\Omega}_j = \emptyset \quad \forall i \neq j, \quad \text{and} \quad \nu(x) = \nu_j \quad \text{for} \quad x \in \hat{\Omega}_j. \qquad (3)$$

For the physical model, and for $\mathrm{H_{0x}}$ and $\mathrm{H_{0y}}$ which stand for sources associated with permanent magnets, we refer to [14].

The rest of the paper is organized as follows. In Section 2, we describe the continuous problem together with the objectives of our algorithms. As we will see, the problem splits basically into two subtasks, the partitioning of meshes, which is discussed in Section 3, and the generation of subdomains described in Section 4. The data formats being involved and the a–priori refinement strategy are described in Section 5. After having described the algorithms being involved, we present the tool *ADDPre* itself in Section 6. Finally, we give some numerical results in Section 7 and conclude with Section 8.

2 Mesh-Independent Formulation of the Objectives

As in the case of magnetostatics, we assume that the problem which is to be solved can be described by PDEs which have to be fulfilled in some given computational domain. For the moment we assume that this domain is bounded. An extension to unbounded domains where BEM is used to model the exterior domain is straight forward. There are given various kinds of data associated with the PDE, involving at least the coefficients and the right hand side. In the simplest case, different sets of data are assigned to different materials defining regions of the computational domain. More generally we call those regions defined by the change of data *data regions*. Thus, each of the data regions can be characterized by some set of data. Typically, the number of data regions denoted by N_M is rather small (in our examples less than 20) such that a further decomposition into $p \geq N_M$ subdomains is required.

Definition 2.1 *A domain decomposition is called* conform *with respect to the data regions if there exists a unique data region for each subdomain of the decomposition such that the intersection is not empty.*

The conformity of a decomposition seems to be a quite natural property. In fact, the efficiency of certain components of the algorithm for solving the discrete problem relies on this property. Hereby, we assume that any further discretization , e.g., by a finite element mesh, which is based on a conform decomposition is conform itself.

Remark 2.1 *In the simplest case, the number of subdomains coincides with the number of processors of a parallel computer which are to be used to solve the discretized problem. It is possible as well to map more than one subdomain to one processor or, vice versa, to use more than one processor for a certain subdomain.*

We formulate the following requirements to a decomposition.

(C1) The decomposition is conform.

(C2) The maximum over the local computational efforts expected for each subdomain is minimal.

(C3) The union of interfaces between the subdomains is minimal.

(C4) The boundaries of the subdomains consist of a rather small number of smooth segments.

We are going to discuss these requirements now.

The first condition (C1) means that the union of data regions is considered as a predefined decomposition. Figure 1 shows this general approach for a given domain Ω with N_M data regions $\hat{\Omega}_i$ which are decomposed into p subdomains Ω_i. The sets of indices $\mathcal{I}_i$ $(i = 1, .., N_M)$ are defined such that the union of $\overline{\Omega}_j$ gives $\overline{\hat{\Omega}}_i$ for $j \in \mathcal{I}_i$. Each decomposition which follows this concept is conform.

Regions defined by the change of data	Decomposition of existing regions $\longrightarrow$	Load balanced decomposition into subdomains
$\overline{\Omega} = \bigcup_{i=1}^{N_M} \overline{\hat{\Omega}}_i$	—— $\overline{\hat{\Omega}}_i = \bigcup_{j \in \mathcal{I}_i} \overline{\Omega}_j$ $\longrightarrow$	$\overline{\Omega} = \bigcup_{i=1}^{p} \overline{\Omega}_i$

Figure 1: Basic concept for a decomposition into subdomains.

The second condition (C2) demands a well balanced distribution of the overall work to the subdomains. In order to describe this condition formally we introduce a continuous density function $d(.) \in C(\overline{\Omega}) \bigcap_{i=1}^{N_M} C^2(\hat{\Omega}_i)$ such that

$$d(x) > 0 \;\; \forall x \in \overline{\Omega} \quad \text{and} \quad \frac{1}{|\Omega|} \int_{\Omega} d(x)\, dx = 1 \tag{4}$$

and

$$\Delta\, d_i(x) = 0 \quad i = 1, .., N_M \tag{5}$$

for the Laplacian Δ, where $d_i(x) := d(x)$ for $x \in \overline{\hat{\Omega}}_i$, $(i = 1, .., N_M)$. Assuming the special behaviour of $d(\cdot)$ and in particular (5) we can compute $d(\cdot)$ at interior points $x \in \hat{\Omega}_i$ from the values on the boundary $\partial\hat{\Omega}_i$. Hence, it suffices to define (or to be able to compute) $d(\cdot)$ on the interfaces $\overline{\hat{\Omega}}_i \cap \overline{\hat{\Omega}}_j$ $(i, j = 1, .., N_M)$. This property is essential for our concept presented below. Furthermore, let $\beta(.) \in L^\infty(\Omega)$ be a piecewise constant function such that $\beta(x) = \beta_i > 0$ for $x \in \hat{\Omega}_i$, $(i = 1, .., N_M)$, and $\sum_{i=1}^{N_M} \beta_i = 1$. Then, we define the expected computational effort $\mathcal{E}^C_{glo}$ by

$$\mathcal{E}^C_{glo} := \int_{\Omega} \beta(x)\, d(x)\, dx = \sum_{i=1}^{N_M} \beta_i \int_{\hat{\Omega}_i} d(x)\, dx. \tag{6}$$

Remark 2.2 *Additional data regions may have to be defined in order to allow more general functions $d(\cdot)$ or β.*

Let

$$\hat{\mathcal{E}}_i^C := \beta_i \int_{\hat{\Omega}_i} d(x)\, dx \quad i = 1, .., N_M \tag{7}$$

denote the expected computational effort for $\hat{\Omega}_i$. Of course, $\mathcal{E}_{glo}^C = \sum_{i=1}^{N_M} \hat{\mathcal{E}}_i^C$ holds. Let p_i, $(i = 1, .., N_M)$ denote the number of processors which are to be assigned to $\hat{\Omega}_i$. Then, according to the second condition (C2) the subdomains Ω_j, $j \in \mathcal{I}_i$, have to be chosen such that

$$\max_{j \in \mathcal{I}_i} \mathcal{E}_j^C \longrightarrow \min, \quad \text{where } \mathcal{E}_j^C := \int_{\Omega_j} d(x)\, dx. \tag{8}$$

The solution is to define Ω_j such that

$$\mathcal{E}_j^C = \frac{\hat{\mathcal{E}}_i^C}{p_i},$$

i.e., the computational effort $\hat{\mathcal{E}}_i^C$ for $\hat{\Omega}_i$ is equally distributed among the subdomains Ω_j, $j \in \mathcal{I}_i$ of $\hat{\Omega}_i$. Hence, in order to satisfy condition (C2) it remains to find N_M positive integers p_i with $\sum_{i=1}^{N_M} p_i = p$ such that

$$\mathcal{E}^C := \max_{j=1,..,p} \mathcal{E}_j^C \longrightarrow \min\,. \tag{9}$$

Remark 2.3 *In the optimal load-balanced case the maximum local effort is $\mathcal{E}_{glo}^C/p$. However, because the decomposition has to be conform it may not be feasible. Assuming that the number of subdomains p is sufficiently large, the distortion of the optimal balance and its influence on the performance of the algorithm remain acceptably low.*

Proceeding to the third condition (C3), we define a measure for the size of the interfaces which describes both, the computational effort and communication with respect to interface data. Let Γ_{ij} be the interface between two subdomains Ω_i and Ω_j, i.e., $\Gamma_{ij} := \overline{\Omega}_i \cap \overline{\Omega}_j$. Then we have for the computational effort on Γ_{ij}

$$\gamma_{ij} := \beta_{ij} \int_{\Gamma_{ij}} d(x)\, dx. \tag{10}$$

with positive weights β_{ij}.

Remark 2.4 *Choosing the parameters β_{ij} we have to take care of both, the computational effort for handling interface problems and the cost for the communication between the two domains (processors) i and j.*

Additionally we define

$$\mathcal{E}_{loc}^\Gamma := \max_{i,j=1,..,p} \gamma_{ij}, \quad \mathcal{E}_{Loc}^\Gamma := \max_{i=1,..,p} \sum_{j=1}^{p} \gamma_{ij}, \quad \mathcal{E}_{glo}^\Gamma := \sum_{i=1}^{p} \sum_{j=i}^{p} \gamma_{ij} \tag{11}$$

for the maximum interface efforts for a single interface and for a single subdomain, respectively, and for the global effort across all interfaces. Introducing parameters $0 \le \delta_1, \delta_2 \le 1$ we can express condition (C3) as follows

$$\mathcal{E}^\Gamma := \delta_1\, \mathcal{E}^\Gamma_{loc} + \delta_2\, \mathcal{E}^\Gamma_{Loc} + (1-\delta_1-\delta_2)\mathcal{E}^\Gamma_{glo} \longrightarrow \min\,. \tag{12}$$

In order to discuss the fourth condition (C4) we assume that the boundaries Γ_i of the subdomains Ω_i are a union of m_i smooth segments g_i^j, i.e.

$$\Gamma_i = \bigcup_{j=1}^{m_i} g_i^j\,.$$

Such smooth segments are, e.g., straight lines or arcs of a circle in the 2–dimensional case or planes and parts of a surface of a sphere in the 3–dimensional case. Then, condition (3) says

$$\mathcal{S}^\Gamma_{loc} := \max_{i=1,..,p} m_i \to \min\,. \tag{13}$$

Introducing

$$\mathcal{S}^\Gamma_{glo} := \sum_{i=1}^{p} m_i \tag{14}$$

and the parameters $0 \le \delta_3, \delta_4$ we can write more generally

$$\mathcal{S}^\Gamma := \delta_3\, \mathcal{S}^\Gamma_{loc} + \delta_4\, \mathcal{S}^\Gamma_{glo} \longrightarrow \min\,. \tag{15}$$

Remark 2.5 *The smoothness of the subdomains has again impact on the efficiency of certain components of the algorithm solving the discrete system. In particular,* $\mathcal{S}^\Gamma_{loc}$ *has to be independent of the spatial discretization parameter* h *!*

Let the parameters $\delta_1, \delta_2, \delta_3, \delta_4$ be fixed. We can formulate our task for given parameters $0 \le \alpha_{EC}, \alpha_{S\Gamma}, \alpha_{E\Gamma} < 1$ with $\alpha_{EC} + \alpha_{S\Gamma} + \alpha_{E\Gamma} = 1$ as follows.

Problem 2.1 *Find a conform decomposition such that*

$$\alpha_{EC}\, \mathcal{E}^C + \alpha_{E\Gamma}\, \mathcal{E}^\Gamma + \alpha_{S\Gamma}\, \mathcal{S}^\Gamma \longrightarrow \min\,. \tag{16}$$

Of course, Problem 2.1 cannot be solved exactly for practical applications. The following two-step algorithm is a first approximation to the solution of Problem 2.1. It is the basis for further considerations.

Algorithm 2.1

1. *Solve Problem 2.1 for* $\tilde\alpha_{EC} := \alpha_{EC}/(1-\alpha_{E\Gamma})$, $\tilde\alpha_{S\Gamma} := \alpha_{S\Gamma}/(1-\alpha_{E\Gamma})$, $\tilde\alpha_{E\Gamma} := 0$.
2. *Modify the solution of step 1. by re–defining (smoothing) the interfaces in order to obtain an approximate solution for the original set of parameters* $\alpha_{EC}, \alpha_{S\Gamma}, \alpha_{E\Gamma}$.

Algorithm 2.1 consists of two subsequent tasks which are discussed together with solutions in the following two sections.

3 Discretization and Mesh Partitioning

In order to solve Problem 2.1 for $\alpha_{E\Gamma} := 0$ we introduce a conform discretization of the domain. Let τ be such a discretization consisting of N elements τ_i such that $\overline{\Omega} = \bigcup_{i=1}^{N} \overline{\tau}_i$. Such a discretization should represent the density function $d(\cdot)$.

Definition 3.1 *Let $\mathcal{I}$ be any set of indices such that $\mathcal{I} \subset \{1,..,N\}$ and $\overline{\Omega}_{\mathcal{J}} := \bigcup_{i\in\mathcal{I}} \overline{\tau}_i$. Moreover let $\Gamma_\tau := \bigcup_{i=1}^{N} \partial\tau_i$ and $\Gamma_{\mathcal{I}} := \bigcup_{i\in\mathcal{I}} \partial\tau_i$. Then the discretization τ is said to* approximate the density function $d(.)$ *iff*

$$\frac{1}{|\Omega_{\mathcal{I}}|} \int_{\Omega_{\mathcal{I}}} d(x)\, dx \approx \frac{n_{\mathcal{I}}}{N} \tag{17}$$

and

$$\frac{\int_{\Gamma_{\mathcal{I}}} d(x)\, dx}{\int_{\Gamma_\tau} d(x)\, dx} \approx \frac{n_{\Gamma_\tau}}{n_{\Gamma_{\mathcal{I}}}}, \tag{18}$$

where $n_{\mathcal{I}}$ is the number of indices in $\mathcal{I}$ and n_{Γ_τ}, $n_{\Gamma_{\mathcal{I}}}$ are the number of edges in Γ_τ and $\Gamma_{\mathcal{I}}$, respectively.

From now on we consider decompositions τ which approximate $d(\cdot)$. Furthermore let the number of elements N of τ be sufficiently large, i.e., $N \geq 10 \cdot p$ in practice. We are going to use triangular elements τ_i. However, the algorithms can be applied to other elements as well.

Remark 3.1 *(18) follows from (17) if the inner angles of the triangles are uniformly bounded from below.*

The value $1/d(x)$ can now be interpreted as the local element size of the discretization τ at the point x, whereas the coefficients β_i in (7) stand for the effort required for one element $\tau_j \subset \hat{\Omega}_i$. Since $d(\cdot)$ is continuous, it can be approximated by a discretization without hanging nodes. On the other hand, $\beta\, d(\cdot)$ cannot be approximated easily if β has jumps.

Let $\hat{\mathcal{I}}_j$ the set of indices such that $\bigcup_{i\in\hat{\mathcal{I}}_j} \overline{\tau}_i = \overline{\hat{\Omega}}_j$. $\hat{N}_j$ denotes the number of elements of $\hat{\mathcal{I}}_j$ for $j = 1,..,N_M$. Then the local computational effort $\hat{E}_i^C$ computes by

$$\hat{E}_i^C := \beta_i\, \hat{N}_i \quad i = 1,..,N_M \tag{19}$$

and the global computational effort E^C is obtained by

$$E_{glo}^C := \sum_{i=1}^{N_M} \hat{E}_i^C. \tag{20}$$

As above, let p_i $(i = 1,..,N_M)$ be positive integers with $\sum_{i=1}^{N_M} p_i = p$. For given p_i let

$$E^C := \max_{j=1,..,N_M} \frac{E_j^C}{p_j} \tag{21}$$

For each $\hat{\mathcal{I}}_i$ we define p_i disjoint subsets $\mathcal{I}_i^j$ such that $\bigcup_{j\in\mathcal{I}_i}\mathcal{I}_i^j = \hat{\mathcal{I}}_j$ where $\mathcal{I}_i$ has p_i elements. This defines a decomposition into disjoint subdomains as follows.

$$\overline{\Omega}_i := \bigcup_{k\in\mathcal{I}_j^i} \overline{\tau}_k,\ (i=1,..,p), \qquad \overline{\hat{\Omega}}_j = \bigcup_{i\in\mathcal{I}_j} \overline{\Omega}_i\ (j=1,..,N_M). \tag{22}$$

Let N_j^i be the number of elements of $\mathcal{I}_j^i$ and $N_j^{\max} := \max_{i\in\mathcal{I}_j} N_j^i$. Looking at the interfaces $\Gamma_{ij} := \overline{\Omega}_i \cap \overline{\Omega}_j$ let $\tilde{n}_{ij}$ denote the number of edges on Γ_{ij} for $i,j=1,..,p$. We define $n_{ij} := \beta_{ij}\tilde{n}_{ij}$ and

$$E_{loc}^{\Gamma} := \max_{i,j=1,..,p} n_{ij}, \quad E_{Loc}^{\Gamma} := \max_{i=1,..,p}\sum_{j=1}^{p} n_{ij}, \quad E_{glo}^{\Gamma} := \sum_{i=1}^{p}\sum_{j=i}^{p} n_{ij}. \tag{23}$$

Similar as in the previous section we define the cost functional

$$E^{\Gamma} := \delta_1\, E_{loc}^{\Gamma} + \delta_2\, E_{Loc}^{\Gamma} + (1-\delta_1-\delta_2)\, E_{glo}^{\Gamma} \tag{24}$$

for given parameters $0 \le \delta_1, \delta_2 \le 1$. For $\hat{\Omega}_j$ $(j=1,..,N_M)$ we define

$$\hat{E}_j := \alpha_{EC}\, N_j^{\max} + \alpha_{E\Gamma}\, E^{\Gamma}. \tag{25}$$

Now we define again a two step algorithm.

Algorithm 3.1

1. *Find* p_i $(i=1,..,N_M)$ *such that* $E^C \longrightarrow \min$.
2. *For* $j=1,..,N_M$ *find* p_i *index sets* $\mathcal{I}_j^i$ *such that* $\hat{E}_j \longrightarrow \min$.

Algorithm 3.1 realizes Step 1 of Algorithm 2.1. We are faced with two tasks. It turns out that Step 1 in Algorithm 3.1 can be realized quite simple by the following algorithm.

Algorithm 3.2

1. *Initialize* $p_i := 1$, $e_i^C := \hat{E}_i^C$ $(i=1,..,N_M)$ *and* $\hat{p} := N_M$.
2. WHILE *($\hat{p} < p$)* DO
   (a) *Find* j *such that* $e_j^C = \max_{i=1,..,N_M} e_i^C$.
   (b) *Set* $e_j^C := e_j^C * p_j/(p_j+1)$, $p_j := p_j+1$, $\hat{p} := \hat{p}+1$.

   END DO

Hence, it remains to propose some realization for Step 2 in Algorithm 3.1. Obviously $\hat{N}_j$ alone could be minimized easily. The challenge is to find a partition which minimizes $\hat{E}^j$ for each data region separately. We propose an algorithm (Algorithm 3.3) which is based on recursive bisections of domains or index sets, respectively.

Algorithm 3.3

DO $j = 1, .., N_M$

(a) $\mathcal{I}_j^1 := \hat{\mathcal{I}}_j,\ q_1 := p_j,\ k := 1$

(b) WHILE $(\max_{l=1,..,k} q_l > 1)$ DO

i. *Find* l *such that* $q_l > 1,\ r_1 := [q_l/2],\ r_2 := [(q_l+1)/2].$

ii. *Bisect* $\mathcal{I}_j^l$ *into* $\mathcal{J}_1,\ \mathcal{J}_2$ *such that* $|\mathcal{J}_1|/|\mathcal{J}_2| = r_1/r_2.$

iii. $\mathcal{I}_j^l := \mathcal{J}_1,\ q_l := r_1,\ \mathcal{I}_j^{k+1} := \mathcal{J}_2,\ q_{k+1} := r_1.$

iv. $k := k+1.$

END DO

END DO

It remains to realize (*ii*) of Algorithm 3.3. In fact, this task has been discussed extensively in the literature for the case $r_1 = r_2$, i.e., the elements will be equally distributed. Simon et al. [1, 17] proposed to apply the spectral bisection method. It works as follows. The discretization is represented by a graph where one node in the graph corresponds to one element of the discretization. Moreover, two elements of the discretization do have a common edge iff the corresponding nodes in the graph are connected by an edge. Having defined this graph with N nodes we define the corresponding matrix $A = (a_{ij})_{i,j=1}^N$ where

$$a_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and the nodes } i \text{ and } j \text{ are connected by an edge} \\ 0 & \text{if } i \neq j \text{ and the nodes } i,\ j \text{ are not connected by an edge} \\ -\sum_{j=1, j\neq i}^N a_{ij} & \text{if} i = j. \end{cases} \tag{26}$$

Then we compute the normalized eigenvector which belongs to the second largest non-zero eigenvalue of A, e.g., by the Lanczos algorithm [16]. This eigenvector has entries in the range $[-1, 1]$. Given some number $c \in [-1, 1]$ we can divide the nodes of the graph, i.e., the elements of the discretization, into two sets according to whether the corresponding entry of the eigenvector is less or equal, or greater than c.

Remark 3.2 *We obtain the classic bisection for* $c = 0$. *The idea of choosing the cut away from zero is due to Brand [2]. It works quite well for* c *being close to zero which is guaranteed in our case because of the recursive approach.*

In the case of non-convex domains the spectral bisection method together with its modification may lead to multiply connected domains. In particular small cells of a few isolated elements are obtained. If the number of elements of such a cell is beyond some threshold, we propose to modify the mapping for the elements of such a cell. So, the cell is assigned to one of its surrounding domains. Doing so, $\hat{N}_i^{\max}$ might be increased whereas E^Γ is decreased, i.e., $\hat{E}_j$ should not grow too much. We apply the following algorithm.

Algorithm 3.4

1. *Compute the matrix A according to (26).*
2. *Compute normalized eigenvector which belongs to the second largest eigenvalue of A.*
3. *Define the index set $\mathcal{J}_1$, $\mathcal{J}_2$ with respect to the cut at $c := \frac{r_1 - r_2}{r_1 + r_2}$.*
4. *Modify $\mathcal{J}_1$, $\mathcal{J}_2$ in order to remove small cells which are isolated.*

Thus, given a discretization τ which approximates the density function $d(\cdot)$ Algorithm 3.5 realizes Step 1 of Algorithm 2.1.

Algorithm 3.5

1. *Apply Algorithm 3.2 to find p_i, $i = 1, .., N_M$.*
2. *Apply Algorithm 3.3 to generate the sets of indices $\mathcal{I}_j^i$, $i = 1, .., p_j$, $j = 1, .., N_M$ using Algorithm 3.4 for the required bisection.*
3. *k:=1.*

   DO $j = 1, .., N_M$

   DO $i = 1, .., p_j$

   $\tilde{\mathcal{I}}_k := \mathcal{I}_j^i$, $\overline{\tilde{\Omega}}_k := \bigcup_{l \in \tilde{\mathcal{I}}_k} \overline{\tau}_l$.

   k:=k+1.

   END DO.

   END DO.

We conclude this section by pointing out the heuristics again. The introduction of a mesh is nothing but an approximation of the original problem. In order to obtain a conform decomposition any further partitioning has to start from the data regions. The numbers p_i can be chosen easily in an optimal way. Then we apply recursive bisection to the data regions and auxiliary domains obtained at intermediate steps. Hereby the spectral bisection minimizes automatically the interface between the two subdomains which are obtained after each bisection. We assume that this recursive approach leads to acceptable solutions of the global problem (the partition into p_i domains). For certain, we do not compute the optimal solution even for simple geometries, e.g. the L-shape with $d(x) = \text{const}$ and $p = 3$. Moreover, most of the numerical algorithms solving large systems of equations (resulting from partial differential equations) are node based whereas our decomposition is element based. Hence, the decomposition obtained from the algorithms above does not reflect the numerical load balance correctly. However, it works sufficiently good.

4 Generation of Subdomains

Now, we discuss some realization of Step 2 of Algorithm 2.1. Basically our approach consists in selecting certain nodes on the interfaces between the subdomains and to replace the edges between two of those nodes by one smooth segment. We describe the algorithm in detail.

Let e_i and P_i denote the edges and nodes of the discretization τ. We define

$$B_i := \{e^i_k : e^i_k = e_{f_i(k)} \subset \partial\tilde{\Omega}_i\}, \qquad B := \bigcup_{i=1}^{p} B_i,$$

i.e., B_i is the set of $n_{B,i}$ edges of τ which form the boundary of $\tilde{\Omega}_i$. The function $f_i(.)$ maps the local numbering to the global numbering. We assume an ordering in B_i according to the geometric position of the edges, i.e, subsequent edges in B_i do have a common node. The nodes are ordered as well such that if $e^i_k = \overline{P_1P_2}$ then $e^i_{k+1} = \overline{P_2P_3}$. For simplicity, we agree upon that any index k of $e^i_k \in B_i$ appearing below has to be taken modulo $n_{B,i}$.

Remark 4.1 *We assume implicitly that the subdomains are connected. However, the approach can be generalized easily to disconnected subdomains.*

Let $F(\cdot)$ denote a flag defined on the set of nodes of τ. Starting with $F(\cdot) = 0$ we are going to select nodes marked by this active flag. Based on those nodes we define a set G of *geometrical edges* g_i. Moreover we generate index sets $\mathcal{I}_{G,j}$ and define the domains by their boundaries $\partial\Omega_i := G_i := \{g_j \in G : j \in \mathcal{I}_{G,i}\}$.

Remark 4.2 *Taking* $G_i = B_i$ *is an admissible solution. However, the minimization with respect to* $\mathcal{S}^\Gamma$ *is not satisfying.*

The following Algorithm 4.1 generates geometrical edges together with sets of indices G_i. It contains the logical function *AcceptEdge*($\overline{P_kP_l}$) which decides whether $\overline{P_kP_l}$ is admissible as new element of G or not. The function itself will be explained later. We start with an empty set of geometrical edges in Step 1. Setting $F(\cdot) = 0$ is the default setting. However, it is also possible to pre-select certain points with $F(\cdot) = 1$ in order to control the algorithm. In Step 2, characteristic points are selected. They are determined by the original data regions as well as by the decomposition into $\tilde{\Omega}_i$. The following steps are partly based on the results of Step 2. Next we consider subsequently the domains $\tilde{\Omega}_i$ and check whether the corresponding domain Ω_i has already been defined completely which is the case if $B_i = \emptyset$. If $B_i \neq \emptyset$ we finish the definition of Ω_i, i.e., of the index set $\mathcal{I}_{G,i}$. For this purpose we try to collect as many subsequent elements of B_i as possible in order to "unify" them to one new geometrical edge. The function *AcceptEdge* controls the quality of the smoothness. The process results in the definition of a new geometrical edge which guarantees that the whole algorithm terminates. The new edge is added to G and the index sets are updated. Step 2 guarantees the compatibility between neighboured domains $\tilde{\Omega}_k$ of $\tilde{\Omega}_i$. In particular, the index set of Ω_k may have to to be updated as well. The edges which could be unified are finally removed from B_i. Since inner edges appear twice, the same sequence of edges has to be removed from some other set B_k.

Algorithm 4.1

1. *Define* B_i*,* B *as above and* $G := \emptyset$*,* $n_e := 0$*,* $\mathcal{I}_{G,i} = \emptyset$ *for* $i = 1,..,p$*. Set* $F(P_k) := 0$ *for all* P_k *of* τ*.*

2. *Set* $F(P_k) := 1$ *for all* P_k *of* τ *which are at the intersection of at least three domains* $\tilde{\Omega}_i$*. Interpret (different) boundary conditions as (different) virtual domains.*

3. DO $i = 1,..,p$

   WHILE $B_i \neq \emptyset$ DO

   i. *Choose* $\overline{P_0P_1} := e^i_k \in B_i$ *with* $F(P_0) = 1$*. If such an edge does not exist then choose any* $\overline{P_0P_1} := e^i_k \in B_i$ *and set* $F(P_0) := 1$*. Set l:=k.*

   ii. DO $l = l+1$, $\overline{P_2P_3} := e^i_l$ UNTIL $F(P_2) = 1$ OR $(AcceptEdge(\overline{P_0P_2}) = 1$ AND $AcceptEdge(\overline{P_0P_3}) = 0)$.

   iii. $F(P_2) := 1$.

   iv. $n_e := n_e{+}1$, $g_{n_e} := \overline{P_0P_2}$. $G := G \cup \{g_{n_e}\}$, $\mathcal{I}_{G,i} := \mathcal{I}_{G,i} \cup \{n_e\}$.

   v. *Remove* $B_{g_{n_e}} := \{e^i_k, .., e^i_{l-1}\}$ *from* B_i*.*

   vi. IF $e_{f_i(k)} = e_{f_j(m)}$, $m \in \{1,..,n_{B,j}\}$, *for* $j \neq i$ THEN $\mathcal{I}_{G,j} := \mathcal{I}_{G,j} \cup \{n_e\}$, *Find* $e^j_l \in B_j$ *with* $e^j_l = e_{f_j(m)}$ *and remove* $\{e^i_k = e^j_m, .., e^j_{m+l-k-1} = e^i_{l-1}\}$ *from* B_j*.* END IF.

   END DO

   END DO

4. DO $i = 1,..,p$

   Define Ω_i *by* $\partial\Omega_i := \{g_j \in G : \ j \in \mathcal{I}_{G,i}\}$*.*

   END DO.

Remark 4.3 *The objects called* edges *can be straight lines, arcs of circles or similar simple geometrical objects. In this sense, a polygonal curve with more than one component is not a simple object.*

It remains to discuss how the function *AcceptEdge* works. Obviously, the geometrical edges have to reproduce the interfaces between the data regions exactly. For this purpose we define the union $\hat{\Gamma}_I$ of such interfaces

$$\hat{\Gamma}_I := \bigcup_{i=1}^{N_M} \partial\hat{\Omega}_i.$$

Furthermore the distortion of the intermediate decomposition which is load balanced should be limited. We introduce a local measure for this distortion. In Algorithm 4.1, the definition of a new geometrical edge g corresponds to a set B_g of n_g edges of τ which is removed from index sets B_i. The edges in B_g form a polygonal curve with $(n_g - 1)$ inner nodes collected in N_g. For any node P_i of τ we can define the local neighbourhood by

$$S_{P_i} := \bigcup_{\overline{\tau}_j \cap P_i \neq \emptyset} \overline{\tau}_j .$$

The distortion caused by the new geometrical edge g is assumed to be small if g intersects all local neighbourhoods of the nodes in N_g. We can define the function AcceptEdge(g).

Algorithm 4.2 (AcceptEdge(g))

```
IF g ∩ Γ_I ≠ ∅ THEN
    IF g ⊂ Γ_I THEN RETURN(1) ELSE RETURN(0). END IF
ELSE
    IF g ∩ S_{P_i} ≠ ∅ for all P_i ∈ N_g THEN
        RETURN(1)
    ELSE
        RETURN(0)
    END IF
END IF
```

The result of Algorithm 4.1 is illustrated in Figure 2. We assume that the white and the gray triangles belong to the same data region, whereas the black triangles belong to a different data region. We observe that three domains meet at P_7. Hence Step 2 will cause $F(P_7) = 1$. Suppose, Algorithm 4.1 has chosen $\overline{P_0P_1}$ in Step $3i$ and $F(P_0) = 1$, $F(P_j) = 0$ for $j = 1,..,6$. Then we go along P_2, P_3, P_4, P_5 and check whether we have to stop or if we can go ahead. None of the edges intersects Γ_I. It turns out that $\overline{P_0P_2}$, $\overline{P_0P_3}$, $\overline{P_0P_4}$ will be accepted whereas $\overline{P_0P_5}$ is refused because it does not intersect the surrounding polygon of P_3. Thus, $3iii$ implies $F(P_4) = 1$ and $\overline{P_0P_4}$ is defined as new geometrical edge. We restart at P_4 and can go to P_7 with $F(P_7) = 1$ already. The new geometrical edge is $\overline{P_4P_7}$.

Remark 4.4 *We may control the results of Algorithm 4.1 by adding or changing criteria for defining the initial set of selected points in Step 2. Furthermore* AcceptEdge *may be modified, e.g., by defining an upper limit for n_g.*

Algorithm 4.2 realizes Step 2 of Algorithm 2.1 which completes its definition. When we started we have introduced a density function $d(\cdot)$ which describes somehow the local mesh size. We claimed that it is sufficient to define $d(\cdot)$ on Γ_I. We ended up by introducing geometrical edges. In order keep the full information we have to supply a method which allows to compute $d(\cdot)$ at least approximately along these geometrical edges. How this is to be done will be explained in the following section.

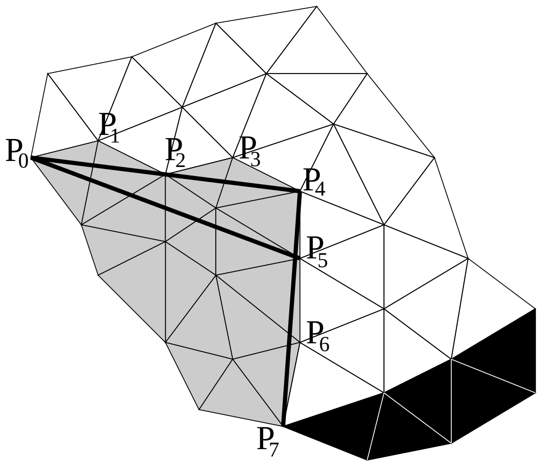

Figure 2: The smoothing of interfaces.

5 Refinement, Data Formats and Adaptivity

The algorithm presented above were concerned with two representations of the geometrical data including $d(\cdot)$. We considered a discretization τ and a decomposition into subdomains defined by geometrical edges together with selected nodes. These two representations are now reflected by two formats of describing data: dd-data and tri-data. Both of them describe the geometry of the problem (shape of the domain, material interfaces) as well as the discretization (density and grading of the mesh). Each of them consists of certain classes of objects which are defined hierarchically downwards. The main difference between the two data types is how the

type	object	definition
dd–data	cross points	coordinates
	geometrical edges	two cross points, (midpoint), refinement parameter, (boundary condition)
	faces	geometrical edges, material pointer
	subdomains	faces
tri–data	points	coordinates
	edges	two points, (midpoint), (boundary condition)
	triangles	edges, material pointer

Table 1: The data formats.

discretization is being represented. The dd-data describe the refinement only on the interfaces (as it is sufficient for a BEM discretization), whereas the tri-data describe the full 2d-discretization (as it is required for a FEM discretization) and thus, they are, in some sense, the 2d–realization of the refinement information contained in the dd-data. Both information are identical since we have assumed that $d(\cdot)$ is piecewise homogeneous (5).

Remark 5.1 *Using the dd-data format, the amount of data is one order lower than for the tri–data format whereas the information is the same.*

Hence, the definition of a new geometrical edge g_{n_e} in Step 3iv of Algorithm 4.1 has to involve the assignment of a refinement information to g_{n_e}. We have to define at least the number of (boundary) elements N_g on g_{n_e}. This can be done, e.g., by

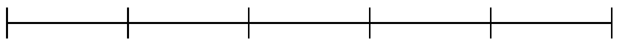

Figure 3: Uniform elements on a geometrical edge.

averaging the element size of edges in B_g and taking the best approximating value. In Figure 2, this gives $N_g = 4$ for $\overline{P_0P_4}$ and $N_g = 3$ for $\overline{P_4P_7}$.

Remark 5.2 *In our example we have $N_g = n_g$ but in general only $N_g \leq n_g$ holds.*

Since only integer values are meaningful for the number of element on the new edge we cannot guarantee that the refinement of τ can be reproduced exactly. However

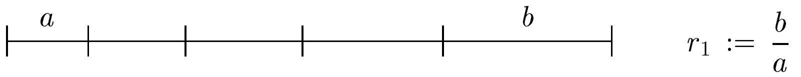

Figure 4: Graded elements on a geometrical edge.

it is possible to extend the approach to locally refined meshes. For this purpose we allow graded meshes on geometrical edges as well. In this case we have to define an additional parameter r_1 which describes the grading of the elements, i.e., the ratio between the last and the first element assuming a geometric sequence. Since the exact reproduction of the local refinement may be important, we may have to add criteria to the function *AcceptEdge*. In particular some new edge has to be refused if the refinement can no longer be reproduced exactly.

In the sense discussed above our algorithm supplies a quasi-static load balancing. We assume that the numerical effort which is to be expected is known a-priori and can be described by some function $\beta d(\cdot)$. A-priori techniques have been described, e.g., by Jung [9]. Alternatively, a-posteriori estimates as described in [18] can be used to refine existing meshes appropriately. Given $\beta d(\cdot)$ we generate a decomposition which is load–balanced with respect to $\beta d(\cdot)$. The algorithms presented above have been implemented. The tool is described in the next section.

6 The Pre-Processing Tool *ADDPre*

We can easily deduce the following algorithm for a DD pre-processing from above.

Algorithm 6.1

> 1. *Generate a mesh which approximates the density function* $d(\cdot)$
> 2. *Apply Algorithm 3.5 to generate a partition of the mesh.*
> 3. *Apply Algorithm 4.1 to generate subdomains.*

Thus, we should require three modules each for one of the three steps in Algorithm 6.1. These modules are

AdapMesh:	Generates a mesh approximating $d(\cdot)$.
Decomp:	Generates a partition of a given mesh applying Algorithm 3.5.
Tri2DD:	Generates subdomains from a partition of a mesh applying Algorithm 4.1.

Furthermore we introduce two types of files

dd-file:	Data in dd-format,
tri-file:	Data in tri-format.

The interaction of the components of the *A*daptive *D*omain *D*ecomposition *Pre*processor *ADDPre* is shown in Figure 5. We assume that the problem has been described

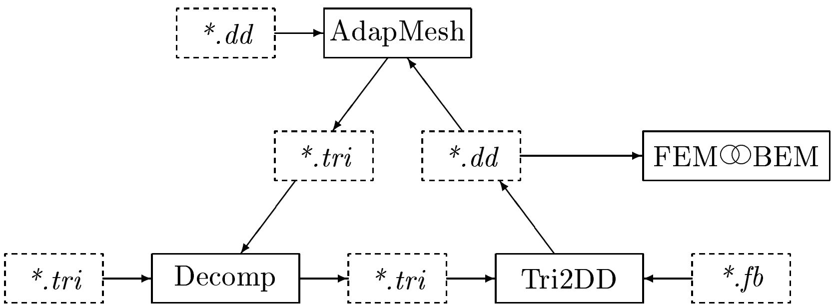

Figure 5: The components of the preprocessing.

either in dd– or tri–format. In the first case we start at the top of the diagram with applying *AdapMesh* to generate a $*.tri$–file. Alternatively, the process starts on the left of the diagram. We apply *Decomp* to the $*.tri$–file which is available now. The output of this process is also a $*.tri$–file which then is converted into a $*.dd$–file by applying the program *Tri2DD*. Additional information, e.g., pre-selected nodes, may be supplied by the $*.fb$–file. In our case, the resulting $*.dd$–file is the input for the parallel code FEM⊗BEM [5].

Remark 6.1 *The cross points occurring in the resulting dd–data are exactly the nodes of* τ *with* $F(\cdot) = 1$*. In FEM⊗BEM the cross–point–system is solved directly, i.e., the number of cross points should be small.*

Because of the simple structure of the dd–data format being used it is quite easy to add or to modify refinement information concerning, e.g., singularities in the $*.dd$–file. Those modifications correspond to a re-definition of the density function $d(\cdot)$.

Remark 6.2 *The new density function may have been generated, e.g., by a coarse-grid computation in connection with a-posteriori error estimators [18].*

Having re-defined $d(\cdot)$ we have to expect a bad load balance as a consequence. In order to recover the load balance we have to generate a new mesh which approximates the new density function, i.e., we apply *AdapMesh* using the mesh–generator *ParMesh* [4]. *AdapMesh* creates a $*.tri$–file which is the input for restarting the cycle with *Decomp.* In any case, the discretization τ cannot be reproduced exactly within the parallel code FEM⊗BEM since we have converted the data into dd-format. Hence we have to accept an imbalance of the distribution within the parallel code. The examples presented next will show that this imbalance is acceptably low.

The DD pre-processing using *ADDPre* can be realized by the following algorithm.

Algorithm 6.2

1. IF *tri-data are given* THEN

   GOTO *Step 3*

   ELSE

   Create (manually) a dd-data of the problem and state a-priori refinement information on the geometrical edges (see Figure 6.1).

   END IF

2. *Use AdapMesh to convert the dd-data into tri-data. (see Figure 6.2).*

3. *Use Decomp to partition the mesh, i.e., to assign each triangle to one of the domains (see Figure 6.3).*

4. *Use Tri2DD to generate subdomains, i.e., to convert the tri-data back into dd-data (see Figure 6.4).*

5. IF $d(\cdot)$ *is to be re-defined* THEN

   Re-define $d(\cdot)$*, modify dd-data,* GOTO *1.*

   ELSE

   STOP.

   END IF

The preprocessor allows us to start off with any data format and to decompose Ω into any number $p \geq N_M$ of subdomains. The only data which are fixed throughout the preprocessing are the definition of $\hat{\Gamma}_I$ and the a-priori given refinement information on $\hat{\Gamma}_I$.

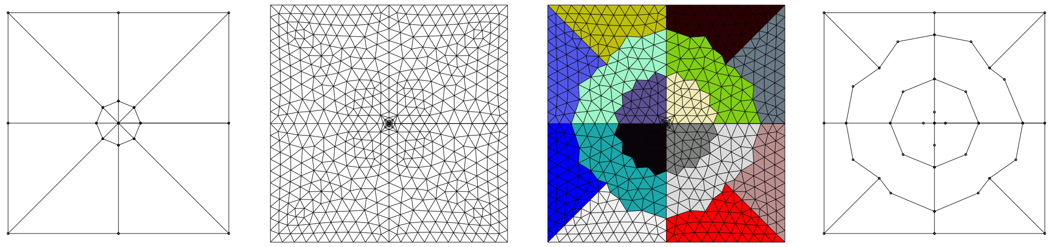

Figure 6: The 4 stages (1–4 from left to right) of preprocessing starting with an initial decomposition (D1) on the left and ending up with a load-balanced decomposition (D2) on the right.

7 Results and Experiments

7.1 The Unit Square

We present the simple example of the unit square with $N_M = 1$ and $d(\cdot) =$ const in order to demonstrate how the modified spectral bisection method works. For this purpose we consider a discretization τ with $N = 160$. First, the mesh is being

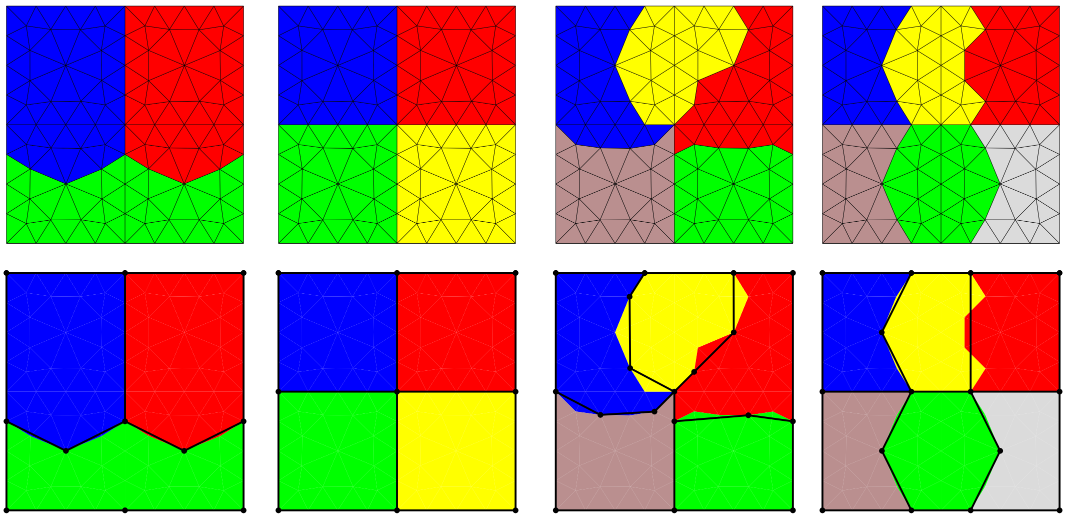

Figure 7: Decompositions of the unit square into $3, 4, 5, 6$ subdomains.

partitioned according to Algorithm 3.4. The results are shown in Figure 7 in the upper row. The number of elements are shown for each partition in Table 2. Second, we apply Algorithm 4.1 to generate subdomains. We obtain the results as shown in Figure 7 in the lower row. Geometrical edges are being introduced and the interfaces are smoothed. The highest number (9) of inner cross points is obtained for $p = 5$.

p	N_1^1	N_1^2	N_1^3	N_1^4	N_1^5	N_1^6
3	53	53	54			
4	40	40	40	40		
5	32	32	32	32	32	
6	26	26	27	27	27	27

Table 2: Results for the unit square.

7.2 The Slit Problem Using Graded Grids

As a further model problem we consider the Dirichlet boundary value problem

$$\Delta u(x) = 0 \quad \text{in } \Omega \quad \text{and} \quad u(x) = r^{1/2} \sin \phi/2 \quad \text{on } \partial\Omega \tag{27}$$

with Ω being the square $(-1,1) \times (-1,1)$ with a slit $([0,1],0)$ and (r,ϕ) are the polar coordinates with respect to the origin. Because of the singularity at $r = 0$ we intend to use an a-priori graded mesh. In order to describe the density function $d(\cdot)$ we introduce auxiliary geometrical edges defining subdomains of Ω, see Figure 6.1. The auxiliary edges allow to define density function which is monotone increasing for $r \to 0$. The corresponding mesh which approximates $d(\cdot)$ is shown in Figure 6.2. The grading is such that $h_{max}/h_{min} > 100$ where h_{min}, h_{max} are the minimum and maximum local element size, respectively. We apply *ADDPre* with $p = 16$ and obtain the decomposition shown in Figure 6.4. In order to use a more powerful computer we perform one uniform refinement step of the mesh shown in Figure 6.2 and apply *ADDPre* with $p = 64$. The result is shown in Figure 8 on the left. The resulting

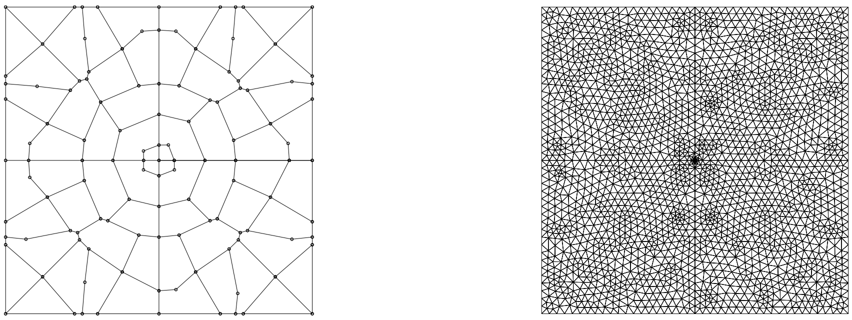

Figure 8: The decomposition (D3) into 64 subdomains (left) and the resulting FEM mesh of the first grid (right).

dd-data are the input for the parallel code FEM⊕BEM. We investigate the load balance after the generation of local finite or boundary element meshes. The finite element mesh is shown for $p = 64$ in Figure 8 on the right. Since the underlying discretization for (D3) is about as double as fine as for (D2) the local problem size should be constant, i.e., independent of p.
Looking at FEM in Table 3 we observe that *ADDPre* (AD) generates an optimal

	BEM				FEM					
	$\min_j \sum_{i=1}^{p} n_{ij}$		$\max_j \sum_{i=1}^{p} n_{ij}$		$\min_j N_1^j$			$\max_j N_1^j$		
p	AD	F⚭B	AD	F⚭B	opt	AD	F⚭B	opt	AD	F⚭B
16	25	25	26	26	68	68	63	68	68	78
64	20	20	25	25	65	65	41	66	66	89

Table 3: The minimum and maximum local number of boundary and finite elements being optimal (opt), after *ADDPre* (AD) and of the local meshes in FEM⚭BEM.

partition (opt) of the mesh with respect to the local number of elements. This optimal balance is distorted by the generation of subdomains, the approximation of the refinement information and the re-meshing within the parallel code FEM⚭BEM (F⚭B). In the case of BEM, the optimal number of local boundary elements cannot be pre-described. We observe that the approach designed for FEM works well for DD–BEM. The subdomains are generated such that the local number of boundary elements is well balanced. The resulting numbers of local finite and boundary

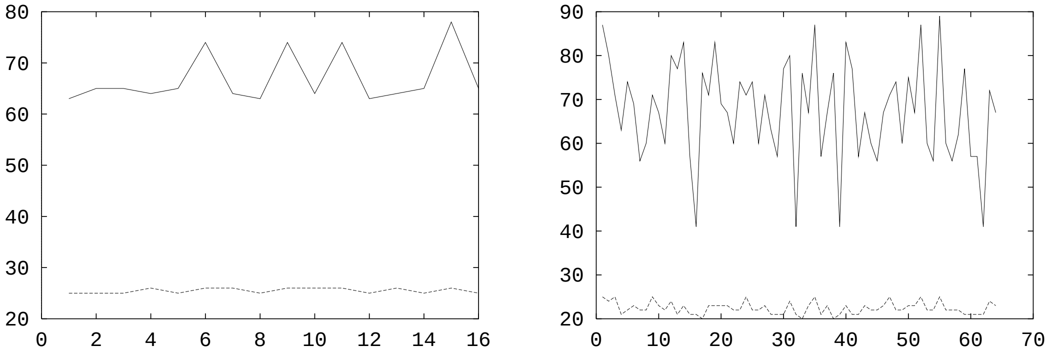

Figure 9: The local number of finite elements (upper curves) and boundary elements (lower curves) for $p = 16$ (D2) (left) and $p = 64$ (D3) (right) after the discretization in FEM⚭BEM.

elements within the parallel code FEM⚭BEM are shown in Figure 9. Obviously, the quality of the load balance with respect to FEM depends additionally on the local mesh generator. In particular it is important how the volume mesh finally corresponds to the refinement information given on the boundary.

7.3 The Asynchronous Motor

We consider the model of an asynchronous motor which is a is a real-life application. The cross–section is shown in Figure 10 on the left where different colors correspond to different data regions. Overall we have $N_M = 11$. The geometry is described in tri–format, see Figure 10 on the right. The number of local elements $\hat{N}_i$ is given in Table 4. The numbers p_i have been computed for $p = 64$ by Algorithm 3.2.

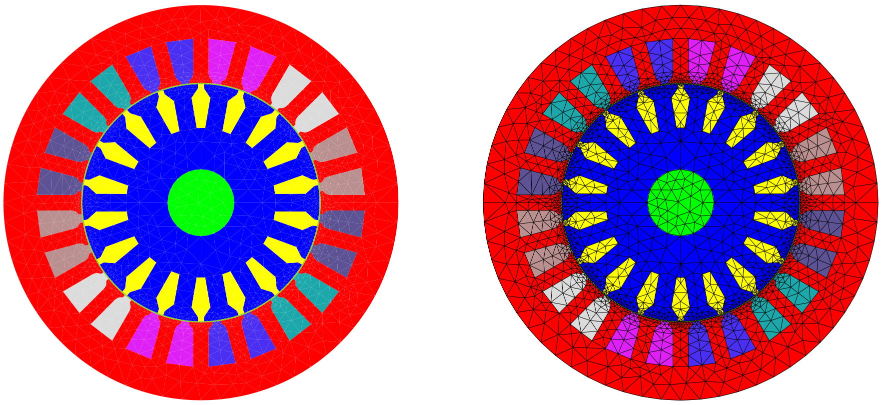

Figure 10: The data regions (left) and the geometry described in tri-format (right) for the asynchronous motor.

We consider again the maximal and minimal numbers of local elements N_i^j of the subdomains compared to the optimal values (opt). The latter are now local with respect to the data regions. A distortion of the load balance is also due to Step 4 of Algorithm 3.4. However, the overall performance of the numerical algorithm is improved ! The result of the partitioning of the mesh using *Decomp* is shown in

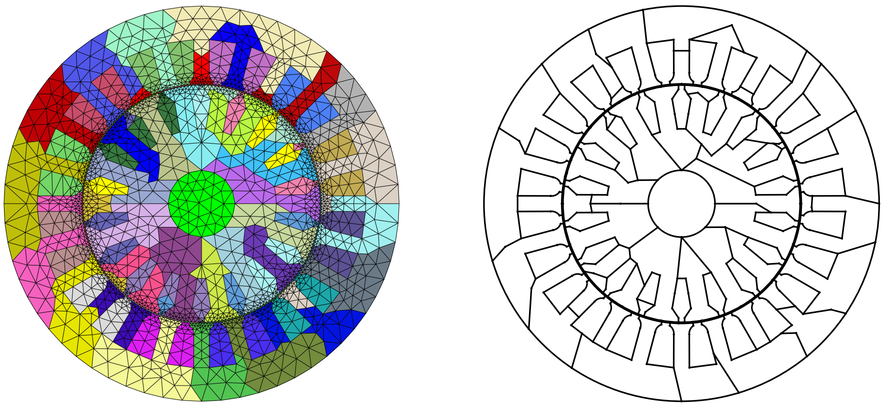

Figure 11: The partitioning of the mesh and the resulting subdomains for $p = 64$.

Figure 11 on the left. *Tri2DD* generates the subdomains as shown on the right. The decomposition does not reflect the symmetries of the original geometry. However, it has been generated automatically within seconds and does not require hours of manual work which may or may not minimize the cost functionals $\hat{E}_j$, $j = 1, .., N_M$.

8 Conclusions

We have presented a tool which generates automatically a decomposition of the computational domain into subdomains. Such a decomposition is required for domain decomposition algorithms using FE, BE, or FE–BE discretizations. Here, only the results of the preprocessing have been analyzed. Applications can be found in var-

i	$\hat{N}_i$	p_i	$\min_{j\in\mathcal{I}_i} N_i^j$			$\max_{j\in\mathcal{I}_i} N_i^j$		
			opt	ADDPre	F⚭B	opt	ADDPre	F⚭B
1	1416	21	67	61	47	68	72	81
2	60	1	60	60	60	60	60	60
3	1064	16	66	62	52	67	74	82
4	448	7	64	56	54	64	74	74
5	108	2	54	54	54	54	54	54
6	112	2	56	56	56	56	56	56
7	108	2	54	54	54	54	54	54
8	108	2	54	54	54	54	56	56
9	112	2	56	56	56	56	56	56
10	102	2	51	48	48	51	54	54
11	444	7	63	63	63	64	64	64

Table 4: The local number of finite elements n_i after *ADDPre* with $p = 64$ and within FEM⚭BEM for the asynchronous motor.

ious papers, e.g., [7, 12, 13, 14]. However, the approach used for generating the subdomains seems to be limited to the 2–dimensional case, i.e., new techniques have to be developed for 3–dimensional applications. Mesh partitioning algorithms for 3d are well known. The tool *Exdasy* [11] collects a number of such tools and allows a comparison of them with respect to different efficiency criteria.

References

[1] S. Barnard and H. D. Simon, *Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurrency: Practice and Experience, 6 (1994).

[2] C. Brand, *A modification of the rsb.* Personal communication at a workshop in Graz, Sept. 1995.

[3] C. Farhat and H. Simon, *TOP/DOMDEC – a software tool for mesh partitioning and parallel processing*, tech. rep., NASA Ames Research Center, Moffet Field, CA, 1993. Technical Report RNR–93–011.

[4] G. Globisch, *PARMESH - a parallel mesh generator*, Parallel Computing, 21 (1995), pp. 509–524.

[5] G. Haase, B. Heise, M. Jung, M. Kuhn, and O. Steinbach, *FEM⚭BEM - a parallel solver for linear and nonlinear coupled FE/BE-equations*, DFG-Schwerpunkt "Randelementmethoden", Report 94–16, University of Stuttgart, 1994.

[6] G. Haase, B. Heise, M. Kuhn, and U. Langer, *Adaptive domain decomposition methods for finite and boundary element equations*, in Reports from

the Final Conference of the Priority Research Programme Boundary Element Methods 1989–1995, (W. Wendland ed.), Stuttgart, October 1995, Heidelberg, 1997, Springer Verlag, pp. 121–147.

[7] B. Heise and M. Jung, *Parallel solvers for nonlinear elliptic problems based on domain decomposition ideas*, Parallel Computing, 22 (1997), pp. 1527–1544.

[8] B. Hendrickson and R. Leland, *The Chaco user's guide – version 2.0*, tech. rep., Sandia National Laboratories, Albuquerque, NM, 1992. SAND94–2692.

[9] M. Jung, *On adaptive grids in multilevel methods*, in Multigrid Seminar, Gosen 1992, S. Hengst, ed., Berlin, 1993, Institut für angewandte Analysis und Stochastik, pp. 67–80. Report No. 5, ISSN 0942–9077.

[10] G. Karypis and V. Kumar, *METIS: Unstructured graph partitioning and sparse matrix ordering system - version 2.0*, tech. rep., Department of Computer Science, University of Minnesota, 1995.

[11] R. Koppler, G. Kurka, and J. Volkert, *Exdasy - an user–friendly and extentable data distribution system*, Lecture Notes in Computer Science, 1300 (1997), pp. 118–127.

[12] M. Kuhn, *DD-BEM: Preprocessing and parallel solution*, in Boundary Elements: Implementation and Analysis of Advanced Algorithms, Proceedings of the 12th GAMM-Seminar, Kiel, January 19 to 21, 1996., W. Hackbusch and G. Wittum, eds., vol. 54 of Notes on Numerical Fluid Mechanics, Braunschweig, Wiesbaden, 1996, Vieweg, pp. 147–158.

[13] ——, *The application of coupled BE/FE formulations in technical magnetic field computations*, Comp. Methods Appl. Mech. Eng., 157 (1998), pp. 193–204.

[14] ——, *Efficient parallel numerical simulation of magnetic field problems*, Ph.D. thesis, Johannes Kepler University Linz, 1998.

[15] U. Langer, *Parallel iterative solution of symmetric coupled FE/BE- equations via domain decomposition*, Contemp. Math., 157 (1994), pp. 335–344.

[16] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, 1996.

[17] H. Simon, *Partitioning of unstructured problems for parallel processing*, Comput. System in Eng., 2 (1991), pp. 135–148.

[18] R. Verfürth, *A Review of A Posteriori Error Estimation and Adaptive Mesh–Refinement Techniques*, Wiley-Teubner Series in Advances in Numerical Mathematics, Wiley and Teubner, Chichester, Stuttgart, 1996.

[19] C. Walshaw, M. Cross, S. Johnson, and M. Everett, *JOSTLE: Partitioning of unstructured meshes for massively parallel machines*, in Proc. Parallel CFD'94, Kyoto, 1994.