TRANSCG

A Scientific Program for Testing

Parallel Algorithms

(Documentation of Version 2)

Gundolf Haase
Michael Kuhn
Johannes Kepler University Linz
Institute of Mathematics
Altenbergerstr. 69
A-4040 Linz

e–mail: ghaase@miraculix.numa.uni-linz.ac.at
kuhn@miraculix.numa.uni-linz.ac.at

1 Introduction

Today high effort is being made in developing and in optimizing parallel algorithms for solving linear systems which arise from discretizing partial differential equations. Such an algorithm for solving elliptic equations numerically is the PCCG-algorithm [7], which runs on, so called, MIMD-computers. It is based upon domain decomposition techniques and the use of multigrid methods. Currently the main interest is being paid in investigating new preconditioning strategies. The program $TRANSCG$ has been developed for testing new ideas and for investigating the influence of control parameters, e.g. the choice of the multigrid-parameters. Since no matrices are being stored the program allows us to use a higher number of meshes for a fixed size of the memory. Thus, the performance of the algorithms under consideration can be observed in more detail.
The main advantage of version 2 of the program $TRANSCG$ is that irregular meshes can be used now, which is desired if singularities occur or if problems over rectangular regions have to be solved. Of course, these irregularities require generalized routines and new strategies, especially for the choice of the preconditioning operators.
The remainder of this paper is organized as follows. In section 2 the capabilities of the program and the required input data are described. In section 3 we explain the geometrical input data and the standard input file. In section 4 the principle of virtual meshes and the generation of the mesh parameters are described. The subsequent section 5 is concerned with the structure of the program and data formats being used. A numerical example is presented in section 6 and, finally, section 7 gives an overview of the subroutines being used within the program.

2 A short description

2.1 Capabilities

The program $TRANSCG$ is mainly thought to be a scientific program for testing algorithms. That is why only a restricted class of equations given on rectangles can be solved. Only such a restriction allows us to avoid the storage of the matrices, arising after discretizing the problem, which is desired, because memory is being saved and the structure of the program is kept simple. Of course, this technique requires the calculation of the matrix elements at each time they are needed. This can be done if the mesh parameters, which are explained in detail in section 4, are known. The following class of equations can be solved:

$$\begin{aligned} -div(k(x)\nabla u(x)) &= f(x) \quad \forall x \in \Omega \\ u(x) &= g(x) \quad \forall x \in \partial\Omega, \end{aligned}$$

where Ω is a rectangle. Actually this condition can be released if more than one processor is being used. Then, the domain Ω will be decomposed into p subdomains, where p is the number of processors. In this case the rectangle-condition is, that all subdomains must be rectangles with its edges parallel to the x- and y-axis, respectively. The functions $k(x)$, $(k(x) > 0\, \forall x \in \Omega)$ are assumed to be constant over the subdomains, whereas the first-kind boundary conditions and the right hand side can be defined arbitrary. The function $g(x)$ is assumed to be linear over the edges with the values at the vertices given in the input file, whereas the function $f(x)$ is assumed to be constant over the subdomains. For test purposes one can rewrite the functions defining $f(x)$ and $g(x)$ which are contained in the file *func.f.*
The main advantage of the new version of the program is, that a-priori adapted meshes can be used now. Thus, the meshes can be adapted to singularities occurring and a wider range of domains can be included in to our considerations.

2.2 Input data - Running the program

The input data describing the discretization have to be provided in a standardized file which will be defined in section 3. These data determine uniquely the grading of the coarsest mesh. Finer meshes are obtained by dividing each element on the edges, such that a uniform grading is maintained. This division is performed for each level, that is $(L_{ev} - 1)$-times, where L_{ev} is the number of levels the user is asked for. Furthermore the components of the preconditioner within the cg-algorithm can be chosen, that is the components of the basis transformation B_I, as it occurs in algorithm 3 in [3], have to be defined. This is being done in dialog with the user.
Thus, the following data have to be provided after starting the program.

Name of the file:	Enter the name of the input file.
Number of levels:	Number of grids in the multigrid algorithm, the coarsest grid is defined by the data given in the input file - subsequent grids are obtained by dividing each element on the edges into two elements.
Basistranformation:	Kind of the multigrid cycle (1,2 $\widehat{=}$ V-cycle and 3,4 $\widehat{=}$ W-cycle).
Number of iterations:	Number of multigrid iterations (1 is sufficient).
Smoothing:	Kind of the pre-smoothing within the multigrid-algorithm (1 $\widehat{=}$ lexicographically forward Gauss-Seidel-smoothing, 2 $\widehat{=}$ ω-Jacobi-smoothing). The post-smoothing is chosen appropriately.
Number of pre-smoothing steps:	Number of pre-smoothing steps within the multigrid-algorithm.
Number of post-smoothing steps:	Number of post-smoothing steps within the multigrid-algorithm.
Full-multigrid:	Decision if the full-multigrid technique is to be used ($\widehat{=}$ 1) or not ($\widehat{=}$ 0).
Generalized cycle:	Decision if the generalized V-cycle (i.e. doubling the number of pre-smoothing steps at every coarser grid) is to be used ($\widehat{=}$ 1) or not ($\widehat{=}$ 0).

During the process of solving the equation every cg-iteration is confirmed by printing the current cg-rate. After finishing the calculations a short statistic, containing the number of iterations, the average cg-rate and several time-measurements, is printed. One can look at the solution on each subdomain by entering the number of the appropriate processor (a number less than zero stops this process). Finally the program can be restarted by typing "1", whereas typing "0" exits the program. If the program has been started within the *xrun*-environment [1] it is possible to generate pictures of the mesh and of the solution as they are presented in section 6.

3 Geometrical Data

3.1 Terminology

In order to solve the differential equation over a domain Ω in parallel via domain decomposition methods we have to find a way of describing the geometry globally (Ω) and locally (subdomains), which allows us, additionally, to control the communication between the processors, e.g. within the pccg-algorithm.

For this purpose we use the terminology of geometrical nodes, geometrical edges, geometrical faces and subdomains. The hierarchy is such that the **geometrical nodes** have to be defined directly by their coordinates whereas higher order structures are defined by lower order ones. That is the **geometrical edges** are defined by two or three geometrical nodes and the geometrical edges themselves define **geometrical faces**. In the 2d-case these faces coincide with the subdomains whereas in the 3d-case the **subdomains** are defined by their faces. The union of all subdomains gives the **domain** the differential equation is to be solved on.

$\boxed{\text{geometrical nodes}}$
$\Downarrow$
$\boxed{\text{geometrical edges}}$
$\Downarrow$
$\boxed{\text{geometrical faces}}$
$\Downarrow$
$\boxed{\text{subdomains}}$
$\Downarrow$
$\boxed{\text{domain}}$

For solving our problem numerically we have to descretize the domain or, more precisely, each subdomain. This process generates (ordinary) **nodes** at which the numerical solution is sought. Geometrical nodes which are starting points and end points of the edges belong to these ordinary nodes and they are called **crosspoints.** Geometrical nodes that have been defined as midpoints of edges do not belong necessarily to the set of ordinary nodes. The remaining nodes on the edges (which are no crosspoints) are called **pure coupling nodes.** The set of all nodes on the boundary of a subdomain is called **coupling nodes** and the set of all nodes remaining is called **inner nodes.** Globally we distinguish between the **boundary** of the domain Ω and the **inner boundary**, that consists of edges lying between two subdomains.

3.2 The standard input file

The communication between the program *TRANSCG* and other programs that may perform pre- and postprocessing routines is realized via a standardized file, which we are going to describe in this section. The file contains geometrical data as well as data defining the problem which is to be solved.

The design of the file is such that it can be used for more general applications than *TRANSCG* can cope with. Note that the following documentation contains all these options regardless of *TRANSCG* does use them or not.

3.2.1 The structure of the file

The standard input file contains several blocks each of them describing a certain part of the global geometry. The headline every block starts with must consist of the following data.

(a) Codeword (string, 8 letters).
The codeword determines which kind of data are contained in the block.

(b) Type (string, 1 letter).
Here, the type, independent of its actual length, of the data following the headline has to be defined. There are five options available:

I	–	integer data
R	–	real data
K	–	complex data
L	–	logical data
C	–	strings.

(c) Length (integer, 4 byte).
This number determines the length of the type named in (b).

(d) Size (integer, 4 Byte).
This number tells how many data of the type (b) and length (c) are contained in the block after the headline.

After the headline the block must contain exactly as many data as determined by the numbers stated above. The sequence of the blocks is not fixed, whereas each block has a special structure which is described next.

3.2.2 The structure of the blocks

The codewords the blocks start with are eight letters long, where (␣) stands for space bar.

(1) Codeword: NKONST␣␣
Type: I
Length: 4

This block contains global data in the sequence given below.

1.) Number nGeom of geometrical nodes.
2.) Number nEdges of geometrical edges.
3.) Number nFaces of geometrical faces.
4.) Number nDomain of subdomains.
5.) Dimension nDim of the coordinate system.
6.) Maximum number nPoly of geometrical edges describing a subdomain.
7.) Number of degrees of freedom per node.
8.) Number nDiri of edges with first kind boundary conditions.
9.) Number nBound of edges with boundary conditions.
10.) Number nMat of materials.

(2) Codeword: NODES␣␣␣
Type: R
Length: 4 (or 8)
Size: 4 $*$ number nGeom of nodes

This block contains the coordinates of the geometrical nodes in the following order.

number of the node
1st coordinate of the ith node
2nd coordinate of the ith node
3rd coordinate of the ith node

Here, i runs from 1 to nGeom.

(3) Codeword: EDGES␣␣␣
Type: I
Length: 4
Size: 6 $*$ number nEdges of the geometrical edges

The block defines the geometrical edges via the geometrical nodes.

Number of the edge
starting crosspoint of the ith edge
end crosspoint of the ith edge
midpoint of the ith edge
type of the edge
pointer

The variable i runs from 1 to nEdges. The following types of edges are defined:

1	:	straight line
2	:	arc of a circle
3	:	parabola.

(4) Codeword: FACES␣␣␣
Type: I
Length: 4
Size: (3+nPoly) $*$ number of subdomains

The block defines the geometrical faces via the geometrical edges.

number of the subdomain
number of the edges describing the subdomain
number edge(1) ,..., Number edge(nPoly)
type of the geometrical face

Here, i runs from 1 to nFaces. If the number of edges describing the subdomain is less than nPoly, then the remaining numbers are set to zero. The type is 1 for plain subdomains.

(5) Codeword: MAPPINGS
Type: I
Length: 4
Size: 3 $*$ number nDomain of subdomains

The block contains the mapping of the subdomains onto the array of processors and classifies the kind of the problem which is to be solved on the subdomain.

number of the processor
number of the subdomain
classification of the problem to be solved in the ith subdomain

Here, i runs from 1 to nDomain. The classification is defined by:

-2	:	BEM-computation, exterior domain
-1	:	BEM-computation, interior domain
1	:	FEM-computation.

(6) Codeword: BOUNDARY
Type: R
Length: 4 (or 8)
Size: (3+6*nFhg) $*$ number nBound of the geometrical edges with b.c.

The block contains the values which are necessary to describe the boundary conditions along the edges.

number of the edge (see EDGES) of the ith edge with b.c.
kind of the boundary conditions
pointer
value1 starting node, value1 end node, value1 midpoint
value2 starting node, value1 end node, value2 midpoint

The variable i runs from 1 to nBound. The value of kind ranges between 1 and 5 and is defined as:

1	:	essential boundary conditions
2	:	natural boundary conditions (flowing condition)
3	:	natural boundary conditions (transition condition)
4	:	periodical boundary conditions
5	:	anti-periodical boundary conditions.

The variable pointer can be used to define a more complicated behaviour of the values of the boundary conditions than the linear one, which is assumed as standard. The second values are the transition coefficients in the case of boundary conditions of the 3rd kind. Otherwise they are not being used. The values of the midpoint will be ignored if the edge is a straight line.

(7) Codeword: MATERIAL
Type: R
Length: 4 (or 8)
Size: 8 ∗ number nMat of materials

The block contains the coefficients of the differential equation, which are constant over the subdomains

$$-\nabla^T \underline{\mathbf{u}}(x) \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} \nabla \underline{\mathbf{u}}(x) + (b_1, b_2) \nabla \underline{\mathbf{u}}(x) + c\underline{\mathbf{u}}(x) \;=\; f \qquad .$$

k_{11}, k_{12}, k_{21}, k_{22}, b_1, b_2, c, f

Here, i runs from 1 to nMat (=nDomain).

(8) Codeword: REFINE0␣
Type: R
Length: 4 (or 8)
Size: 5 ∗ number nEdges of geometrical edges

This block defines the a-priori strategy for refining the edges.

number of the edge (see EDGES)
kind of discretization of the ith edge
number of inner points on the ith edge
ratio 1
ratio 2

Here, i runs from 1 to nEdges.

3.3 Global and local data

In order to compute a numerical solution over the domain Ω we need, of course, communication between the processors and thus global data, e.g. a global numbering of the geometrical nodes and edges. On the other hand the benefit of parallelism is based upon the fact that the majority of the data is local. This applies to the computational quantities as well as to the geometrical data. The latter are being stored in two common blocks *GeomGlo* (global data) and *GeomLoc* (local data).

3.3.1 The two common blocks

The common block *GeomGlo* (file geomglo.inc) contains several global constants and variables. The global constants are also being used in the common block *GeomLoc* (file geomloc.inc).
Constants :

constant	:	description
DimEdg	:	Length of an entry for describing the edges.
DimMap	:	Length of an entry for describing the mapping of the subdomains onto the array of processors and classification of the problem (FEM, BEM).
DimRef	:	Length of an entry for describing the integer values, which are used by mesh generator.
DimBC	:	Length of an entry for describing the number of an edge, the kind of the boundary conditions and additional information concerning the boundary conditions.
DimMat	:	Length of an entry for describing the material dependent coefficients on a subdomain.

Variables :

variable	:	description
nGeom	:	Number of global geometrical nodes including the midpoints of the edges.
nCrP	:	Number of global crosspoints.
nEdges	:	Number of geometrical edges.
nFaces	:	Number of geometrical faces (in the 2d case equal to nDomain).
nDomain	:	Number of subdomains.
nDim	:	Dimension of the domain (2 or 3).
nPoly	:	Maximum number of geometrical edges for describing one face.
nFhg	:	Number of degrees of freedom per node.
nBound	:	Number of geometrical edges with boundary conditions.

Temporary arrays :

array	:	description
XCross(nDim, nCross)	:	Coordinates of the geometrical nodes.
IEdges((DimEdg, nEdges)	:	Description of the edges.
IFace(nPoly+2, nFaces)	:	Description of the faces.
IMapping(DimMap, nDomain)	:	Mapping of the subdomains onto the array of processors.
IRefine(DimRef, nEdges)	:	Integer information for the mesh generator.
RRefine(nEdges)	:	Real information for the mesh generator.
IBound(DimBC, nBound)	:	Integer information concerning the boundary conditions.
RBound(2*3*nFhg, nBound)	:	Real information concerning the boundary conditions.
RMat(DimMat, nDomain)	:	Material dependent coefficients.

The common-block *GeomLoc* contains the local constants, variables and arrays for describing the local geometry, the initial data for the local mesh generator, the relations between the subdomains and the local coefficients of the differential equation.

Constants :

constant	:	description
MaxEdges	:	Maximum number of local geometrical edges.
MaxVertex	:	Maximum number of local geometrical nodes.
DimIE	:	Column dimension of the array IE.
MaxFhg6	:	Maximum number of degrees of freedom per node multiplied by 6.

Variables :

variable	:	description
NrDomain	:	Number of the subdomain handled by the processor.
NumVertex	:	Number of local geometrical nodes.
NumEdges	:	Number of local geometrical edges.
KindFace	:	Kind of the subdomain (3d only).
KindDisk	:	Kind of discretization (FEM=1 / BEM(int)=-1 / BEM(ext)=-2)
NV	:	Number of local crosspoints.
NE	:	Number of nodes on the local edges without the crosspoints.
NC	:	Number of local coupling nodes (= NV+NE).
NI	:	Number of inner nodes (locally).
nNode	:	Number of nodes in the subdomain (= NC+NI).
nElem	:	Number of elements (FEM) in the subdomain.

Arrays :

array	:	description
NrVerL2G(MaxVertex)	:	Global numbering of the local geometrical nodes.
NrEdgL2G(MaxEdges)	:	Global numbering of the local geometrical edges.
XCrole(2, MaxVertex)	:	Coordinates of the local geometrical nodes.
IEdgle(DimEdg, MaxEdges)	:	Description of the local edges.
IRefle(DimRef, MaxEdges)	:	Integer information for the local mesh generator.
RRefine(MaxEdges)	:	Real information for the local mesh generator.
IBoundle(MaxEdges)	:	Integer information concerning local boundary conditions.
RBoundle(MaxFhg6,MaxEdges)	:	Real information concerning local boundary conditions.
RMatle(DimMat)	:	Local material dependent coefficients.
IDirle(MaxVertex)	:	Kind of boundary conditions at the vertices.
RDirle(MaxVertex)	:	Contains the function value at the vertex in the case of essential boundary conditions.

3.3.2 The structure of the local data

In this section we describe the local arrays which define the geometry and the differential equation. All numbers, names and quantities mentioned are restricted to the subdomains only, that is they are local data.

Arrays describing vertices.

XCrole $\longrightarrow$ Coordinates of the geometrical nodes :

XCrole(2, i) [Real] = | x-coordinate | y-coordinate |

Note that the nodes must be ordered globally in such a way, that the numbers i=1,...,nCrP stand for the crosspoints and i=nCrP+1,...,nGeom stand for the midpoints that may be present.

IDirle, RDirle $\longrightarrow$ Essential boundary conditions at the vertices :

$$\text{IDirle(i) [Integer]} \quad = \quad \boxed{\begin{cases} 1 & \text{if essential b.c. are given at the vertex} \\ 0 & \text{otherwise} \end{cases}}$$

$$\text{RDirle(i) [Real]} \quad = \quad \boxed{\begin{cases} \text{value of b.c.} & \text{if IDirle(i) = 1} \\ 0.0 & \text{otherwise} \end{cases}}$$

Here, i runs from 1 to NV.

NrVerL2G $\longrightarrow$ Mapping onto the global numbering :

NrVerL2G(i) [Integer] = | global number of the ith crosspoint |

Here, i runs from 1 to NumVertex.

Arrays describing the edges.

IEdgle $\longrightarrow$ description of the edges :

IEdgle(5, i) [Integer] =

$$\underbrace{\boxed{\text{starting point}}\boxed{\text{end point}}\boxed{\text{midpoint}}}_{\text{geometrical nodes of the ith edge}}\boxed{\text{type of the edge}}\boxed{\text{pointer}}$$

Here, i runs from 1 to NumEdges. The midpoint is being ignored if the edge is a straight line. The following types of edges are defined :

1	:	straight line
2	:	arc of a circle
3	:	parabola.

More general types of edges can be defined using the variable pointer.

IE $\longrightarrow$ description of the edges :

IE(5, i) [Integer] =

$$\underbrace{\boxed{\text{starting node}}\boxed{\text{end node}}}_{\text{of the ith edge}}\boxed{\text{neighboured process}}\underbrace{\boxed{\text{starting vertex}}\boxed{\text{end vertex}}}_{\text{crosspoints}}$$

Here, i runs from 1 to NumEdges. It is necessary that the nodes along an edge (without the crosspoints) are numbered increasingly according to their distance (meassured along the edge) to the first node on the edge. By naming the starting crosspoint and end crosspoint a one-to-one mapping of the coupling edges between the subdomains is garanteed. The numbering of the processes starts with 0. If the entry "neighboured process" is less than zero, then the edge being considered belongs to the boundary of the domain and the entry multiplied by (-1) gives the kind of boundary conditions on that edge.

REdgle $\longrightarrow$ weighted coefficient along the edge :

REdgle(i) [Real] = $\boxed{\begin{cases} 0.5[Mat(process) + Mat(\text{IE(3,i)})] & \text{if IE(3,i)} \geq 0 \\ Mat(process) & \text{otherwise} \end{cases}}$

Here, i runs from 1 to NumEdges. These data are required for the preconditioning operator along the coupling edges. $Mat(process)$ is set equal to RMatle(1).

NrEdgL2G $\longrightarrow$ mapping onto the global numbering of the edges :

NrEdgL2G(i) [Integer] = $\boxed{\text{global number of the ith geometrical edge}}$

Here, i runs from 1 to NumEdges.

Arrays describing boundary conditions.

RBoundle $\longrightarrow$ boundary conditions along the edges :

RBoundle(6*nFhg, i) [Real] =

value1-starting point	value1-end point	value1-midpoint
value2-starting point	value2-end point	value2-midpoint

Here, i runs from 1 to NumEdges. The dimension of the array RBoundle depends on the number of degrees of freedom per node, here we assume nFhg = 1. These values are zero for real coupling edges (i.e. inner boundaries). The values of the midpoint are ignored of the edge is a straight line and the 2nd values are ignored in the case of 1st/2nd kind boundary conditions.

IBoundle $\longrightarrow$ boundary conditions along the edges :

IBoundle(i) = $\boxed{\text{pointer to computation routines}}$

This pointer is used if non-standard (i.e. not linear interpolated) values are to be used at the nodes on the edge.

Material dependent coefficients.

RMatle $\longrightarrow$ coefficients :

RMatle(8) =

k_{11}	k_{12}	k_{21}	k_{22}	b_1	b_2	c	f

The coefficients are assumed to be constant over the subdomain. They refer to the differential equation as follows:

$$-\nabla^T \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} \nabla u(x) + (b_1, b_2) \nabla u(x) + c\, u(x) \;=\; f \quad .$$

Arrays controlling the mesh generator.

IRefle, RRefle $\longrightarrow$ descretization of the edges :

IRefle(2, i) [Integer] =

kind of descretization	number of inner nodes

RRefle(2, i) [Real] =

ratio 1	ratio 2

Here, i runs from 1 to NumEdges.

3.4 The auxiliary program *Geommake*

In spite of the rather simple structure of the standard input file it is a hard and error producing work to edit such a file if a domain is to be subdivided into more than 8 subdomains. So we wrote a small and helpful program, called *Geommake,* which supports the user in doing this work. The program subdivides squares and rectangles into 2^k subdomains and creates all the blocks containing both, geometrical data and dats describing the problem which is to be solved. The latter is assumed to be the Poisson equation with homogeneous Dirichlet boundary conditions. The program has been written in such a way that any personal extensions for other domains, materials and boundary conditions can be made easily. Each subroutine is written in its own file. A detailed description of the subroutines can be found in the file *Geom.doc* and within the source code.

4 Virtual meshes and their generation

4.1 Data structures

The idea of this program is to avoid the storage of the matrices. Whereas the use of uniform meshes on quadratic subdomains (to which the first version was restricted) requires only the number of intervals on each edge for computing elements of the stiffness matrix, more information will be needed in the case of irregular meshes on rectangular subdomains. Still the triangulation will be obtained from partitioning the rectangle into small rectangles by horizontal and vertical lines and dividing each rectangle into two triangles.
Now a nonuniform partition of the edges is allowed. There are seven options for dividing an edge into elements as it is described in the next section. The desired partitioning has to be defined in the block *REFINE0* of the input file.
Throughout the program the number of inner points on each edge is known for every grid and is denoted by nx, ny, where nx stands for the number on the horizontals and ny for the number on the verticals. For two subsequent meshes the equations $(NX + 1) = (nx + 1)/2$ and $(NY + 1) = (ny + 1)/2)$, where $(n.)$ stands for the finer and $(N.)$ for the coarser grid, are fulfilled. The sizes of the elements on the edges are calculated a-priori for all meshes. They are stored in two vectors HX and HY, where the information regarding the finest mesh stand at the beginning of the appropriate vector, which is illustrated below.

h_1	$\cdots$	h_{n+1}	H_1	$\cdots$	H_{N+1}	$\cdots$	HG_1	$\cdots$	HG_{NG+1}

Here, $h_1, \ldots, h_n$ are the element sizes belonging to the finest grid, whereas $HG_1, \ldots, HG_{NG+1}$ belong to the coarsest grid. For two subsequent grids the following holds

$$\begin{aligned} N + 1 &= (n + 1)/2 \\ H_i &= h_{2i-1} + h_{2i}, \end{aligned}$$

where, again, the capital letters stand for the coarser grid.

4.2 The generation of the meshes

In this section we will explain how the meshes are obtained from the data given in the input file. For controlling the generation of the discretizing parameters the following values have to be defined in the file:

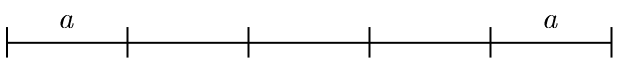
nn	$\longrightarrow$	Number of inner nodes on an edge for the coarsest grid,
kind	$\longrightarrow$	Kind of discretization,
r_1	$\longrightarrow$	Ratio 1 of element sizes for the coarsest grid,
r_2	$\longrightarrow$	Ratio 2, not being used.

For simplicity we define additionally:

$ne \quad \longrightarrow \quad$ Number of elements on an edge for the coarsest grid,

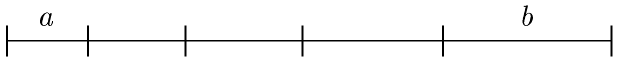
where $ne = nn + 1$ holds. The parameter *kind* has the range $[1, \ldots, 7]$, where the numbers define the coarsest grid as follows.

- kind = 1: Uniform discretization.



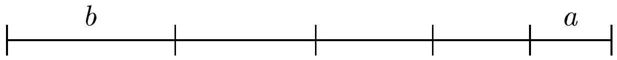
$r_1 := 1.0$

- kind = 2: Compression towards the starting point.



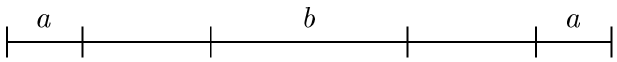
$r_1 := \frac{b}{a}$

- kind = 3: Compression towards the end point.



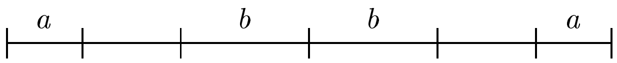
$r_1 := \frac{b}{a}$

- kind = 4: Compression towards the boundary (ne odd).



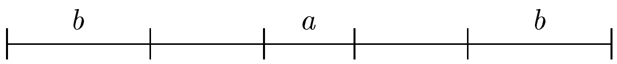
$r_1 := \frac{b}{a}$

- kind = 5: Compression towards the boundary (ne even).



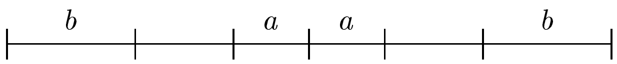
$r_1 := \frac{b}{a}$

- kind = 6: Compression towards the centre (ne odd).



$r_1 := \frac{b}{a}$

- kind = 7: Compression towards the centre (ne even).



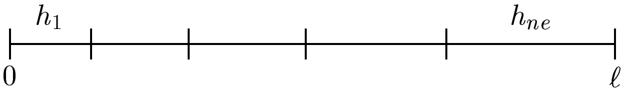
$r_1 := \frac{b}{a}$

These seven cases are converted internally such that the discretization can be described by two alternatives only. We distinguish between a grading into one direction, $k = 1$ for $kind = 1, 2, 3$, which involves the uniform refinement as a special case, and a grading towards the boundary or towards the centre, $k = 2$ for $kind = 4, 5, 6, 7$. For this purpose we have to redefine the values r_1 and r_2 in the following way:

$$\begin{aligned} r_1 &= r_1 \quad \text{for } kind = 1, 2, 4, 5 \\ r_1 &= 1/r_1 \quad \text{for } kind = 3, 6, 7 \\ r_2 &= 1/r_1 \quad (\text{new value of } r_1). \end{aligned}$$

Then the coarsest mesh, i.e. the mesh for the lowest level, is defined uniquely by the following algorithm with the input-data given above.

- <u>k = 1 : Grading into one direction.</u>

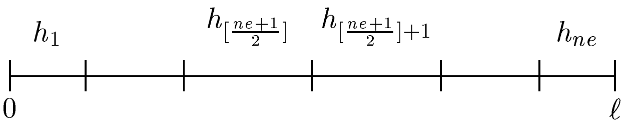


$$r_1 := \frac{h_{ne}}{h_1}$$

One obtains the following formulae for the length h_1 of the first element and for the ratio q_1 of neighboured elements

$$\begin{aligned} q_1 &= \sqrt[ne-1]{r_1} \\ h_1 &= \begin{cases} \ell/ne & \text{iff} \quad q_1 = r_1 = 1.0 \\ \ell \frac{q_1 - 1}{q_1^{ne} - 1} & \text{otherwise.} \end{cases} \end{aligned}$$

- <u>k = 2 : Grading from the centre towards the boundary.</u>



$$r_1 := \frac{h_{[\frac{ne+1}{2}]}}{h_1}$$

Here, we obtain the following formulae for the length h_1 of the first element and for the ratio q_1 and q_2 of neighboured elements, where q_1 is the ratio for the first $[\frac{ne+1}{2}]$ elements and q_2 the ratio for the elements $[\frac{ne+1}{2}], \ldots, ne$

$$\begin{aligned} q_1 &= \sqrt[[\frac{ne-1}{2}]]{r_1} \qquad q_2 = \frac{1}{r_1} \\ h_1 &= \ell \left(\frac{q_1^{[\frac{ne}{2}]} + q_1^{[\frac{ne+1}{2}]} - 2}{q_1 - 1} \right)^{-1} \end{aligned}$$

where [.] denotes the integer part and r is assumed not to be one, i.e. the uniform discretization is excluded.

Further grids are obtained by dividing each element into two subelements such that one gets a grading with the ratio which is the square root of the previous ratio. That is if two neighboured (old) elements do have the ratio q_1 the two subelements will have the ratio $\sqrt{q_1}$. This can be done for $kind = 1, 2, 3, 5, 7$. In the case that $kind = 4, 6$ the element in the middle will be halved, so that the $\sqrt{q_1}$-property mentioned above will only be valid for the elements away from the centre.

5 The main parts of the program

5.1 The algorithm

The basic algorithm being used for solving the linear system, which arises from discretizing our problem, has been described in [10]. The main interest is being paid in investigating the following components:

- choice of the basistransformation B_I
- choice of the operator C_I, compare [3]
- choice of the operator C_C (preconditioning along the edges and for the vertices)
- choice of the multigrid parameters.

Because of the simple data structure being used new components can be programmed easily and existing routines can be exchanged as desired. Note, that only piecewise linear trial functions on triangular elements can be used and that the multigrid algorithm is based upon bilinear interpolation and the appropriate restriction.

5.2 Data formats

<u>Vectors within the CG</u>

The following structure of numbering the nodes in each subdomain is being used within the cg-algorithm :

1	$\cdots$	NV

crosspoints

NV+1 = IE(1,1)	$\cdots$	IE(2,1)	$\cdots$	IE(1,NumEdges)	$\cdots$	IE(2,NumEdges)

'pure' coupling nodes

NC+1	$\cdots$	nNode

inner nodes.

That is, the vertices are numbered first followed by the remaining nodes on the boundary, edge by edge. Finally the inner nodes of the rectangular subdomain are numbered line by line parallel to the x-axis.

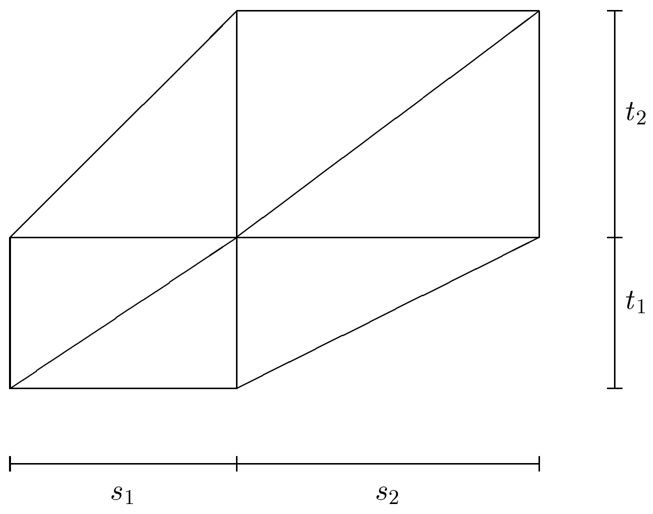
<u>The local coarse-grid matrix</u>

In opposition to uniform meshes on squares the meshes under consideration do not lead to a one-point coarse grid system, which has to be solved at each multigrid step on the coarsest grid, in general. That is why this system, that is the corresponding stiffness matrix (which is symmetric and positiv definite) is computed and decomposed once before the cg-iteration starts. Then, only the back-substitution has to be performed within the multigrid routine. The matrix is stored in vbs-format (variable band-width, only the upper triangle is being stored) with a vector the components of which point to the last elements of the rows.

<u>The matrix K_E</u>

Since irregular meshes can be used a generalized preconditioning has been implemented. This technique requires the assembled diagonal of the stiffness matrix for the points on the coupling edges. Because this computation requires communication between the processors the diagonal is computed before the cg-iteration, too. The values are stored in the vector $SKC4$.

Elements of K



The picture above shows the triangles sourrounding an inner node. One obtains the values s_1, s_2 and t_1, t_2 of the element sizes from the vectors HX and HY (see section 4). From these values we can compute the elements of the stiffness matrix in the line belonging to the node at the centre. For the diagonal element we have

$$s_{ii} = \frac{1}{2}\left[\frac{s_1+s_2}{t_1} + \frac{s_1+s_2}{t_2} + \frac{t_1+t_2}{s_1} + \frac{t_1+t_2}{s_2}\right].$$

On the other hand one obtains for a non-diagonal element, e.g. for the one belonging to the right neighbour, the formula

$$s_{ii+1} = -\frac{t_1+t_2}{2\,s_2}.$$

Other non-diagonal elements are being computed in a similar way by switching the values of s_i and t_i.

5.3 Statistical and graphical output

After finishing the cg-iteration the following statistic is printed out (here a 2x2-example has been chosen).

```
Times for waiting+communication [s]:

processor
log. /    phys.  input :  in % : output:  in % : overall:
   0    0  0  0    0.03    0.07    0.03    0.07   41.93
   1    0  0  1    3.56    8.48    0.03    0.06   41.93
   2    0  1  0    3.56    8.50    0.03    0.08   41.93
   3    0  1  1    3.56    8.49    0.03    0.07   41.93

pure arithmetic-time (max):      41.87
```

```
(w,r)-defect              :      0.32E-D06 after 10 iterations
average cg-rate           :      0.2242

coarse-grid system solved in     0.03 sec. solved    (    0.06 % )
pure coupling edges              0.56 sec. needed    (    1.32 % )
preconditioning                 35.84 sec. needed    (   85.46 % )
entire algorithm                41.93 sec.
```

It shows global measurements as the duration of the algorithm, the average cg-rate and the time-shares of the single components of the algorithm. Additionally the time-shares for communication on each processor can be seen from the table at the beginning of the output. For example, the uniform load balancing can be observed from these data. Provided that the program has been started from the *xrun*-tool it is possible to generate pictures of the mesh and a 2d-representation of the solution as it can be seen in section 6.

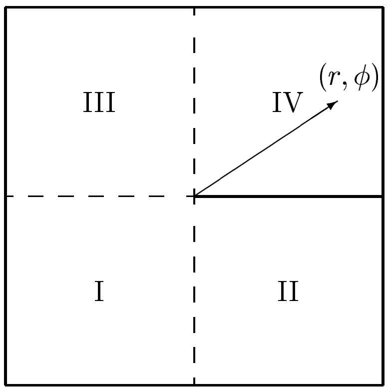
6 A model problem

6.1 The problem

In this section we are going to demonstrate the performance of the program and the influence of the mesh-grading on both, the accuracy of the solution and the efficiency of the preconditioners being used. For this purpose we choose the following model problem:

$$\begin{aligned} -\Delta u(x) &= 0 \quad x \in \Omega \\ u(x) &= r^{0.5} \sin \frac{\phi}{2} \quad x \in \partial\Omega, \end{aligned}$$

where Ω is the square $(0,2)\times(0,2)$ with a slit $[(1,1),(2,1)]$ and (r,ϕ) are the polar-coordinates with respect to the origin $(1,1)$ as shown in figure 1. The roman numbers and the dashed lines



Solution:

$$u = r^{0.5} \sin \tfrac{\phi}{2}.$$

Regularity:

$$u \in H^{1.5}(\Omega).$$

Figure 1: Model problem.

in the figure indicate the subdomains as they are handled by the 4 processors being used. We know that the analytical solution has a singularity at the point $(1,1)$. One can expect that a mesh, which is adapted to this singularity, will lead to a more accurate numerical solution. On the other hand we will observe, that the preconditioners loose their efficiency if the mesh becomes strongly irregular.

6.2 Numerical results

The results and, especially, the computation times presented in this section have been obtained with a PARSYTEC MultiCluster 2. For solving our problem numerically we use two different meshes (figure 6.2), a regular one and an adapted mesh, which is compressed towards the singularity, that is towards the point $(1, 1)$. Of course, the adapted mesh shown is not

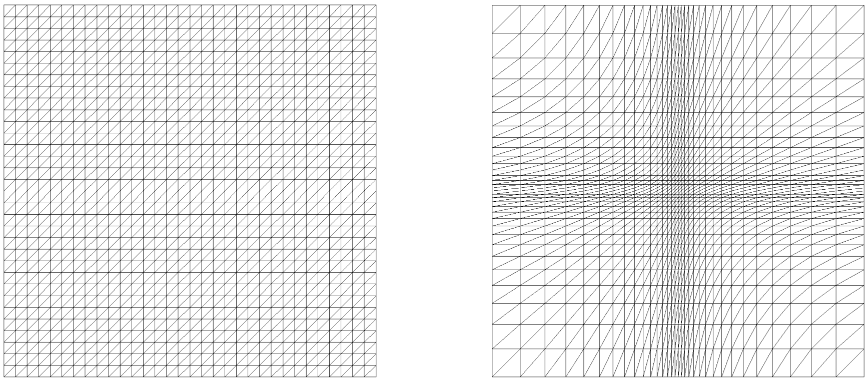


Figure 2: Regular mesh (left) and adapted mesh (right).

graded in the optimal way and that is why we cannot expect the recovery of the quadratic convergence. What we can expect is a more accurate numerical solution for the same number of unknowns. Indeed, these facts are confirmed by the numerical results presented in table 1, where u stands for the analytical and u_h for the numerical solution. The results printed

Table 1: Numerical results obtained for the meshes shown above.

it.	$\|u - u_h\|_{\infty,\Omega}$	$\|u - u_h\|_{0,\Omega}$	grid	nodes	$\|u - u_h\|_{0,\Omega}$	$\|u - u_h\|_{\infty,\Omega}$	it.
7	4.62e-2	1.10e-2	2	945	3.22e-3	2.41e-2	14
8	3.34e-2	5.70e-3	3	3937	1.54e-3	1.70e-2	16
8	2.39e-2	2.89e-3	4	16065	7.46e-4	1.20e-2	19
9	1.70e-2	1.46e-3	5	64897	3.67e-4	8.44e-3	20
10	1.20e-2	7.31e-4	6	260865	1.80e-4	5.96e-3	22

on the left have been obtained for a regular mesh, whereas the graded mesh as drawn in figure 6.2 leads to the results on the right hand side. In both cases we observe almost the same rate of convergence (determined by the order of the singularity), which is one for the $H^0(\Omega)-$Norm and 1/2 for the $L^\infty(\Omega)-$Norm. As expected, the absolute values of the norms are much lower in the case that the adapted mesh is being used. Looking at the pictures 6.2 this phenomenon can be observed too. Here, the behaviour of the numerical solution obtained for different mesh-types is shown. In the left picture there one can see clearly, that the numerical solution is distorted near the singular point. The reason for this distortion is the insufficient refinement of the mesh near that point. Even the adapted mesh shown above is not compressed optimally. This fact is underlined by the results shown in table 2, where the factor of grading is being increased further. The table shows the norms of the error for a fixed number of unknowns. Here, the growing irregularity of the meshes is the reason for the

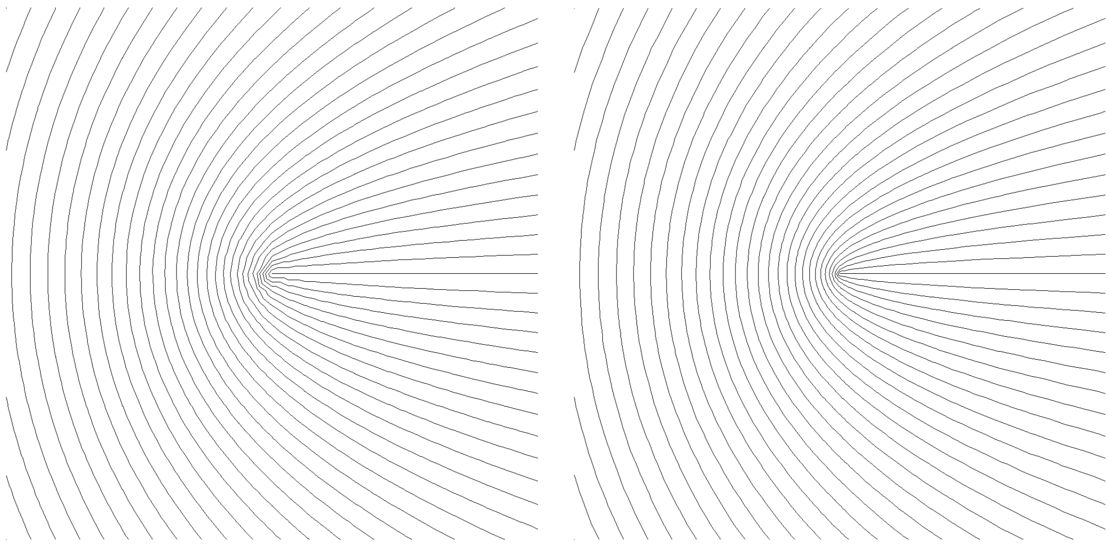


Figure 3: Solution obtained for two meshes shown above.

Table 2: Dependence of the error and the number of iterations on the grading of the mesh.

ratio	it.	$\|u-u_h\|_{\infty,\Omega}$	$\|u-u_h\|_{0,\Omega}$
1.0	7	4.62e-2	1.10e-2
8.0	14	2.41e-2	3.22e-3
32.0	29	1.38e-2	1.20e-3
128.0	48	7.44e-3	5.24e-4

increasing iteration-numbers. This fact indicates, that the preconditioners being used require a certain regularity of the mesh in order to work efficiently. Table 3 shows the positive influence of using strongly irregular meshes. The absolut time which is required in order to reach a certain error-bound is decreasing if the ratio of grading is being increased. That is why we get closer to the optimal grading by increasing the ratio and thus the analytical solution can be approximated numerically by less unknowns that is by a lower number of grids. The latter mainly determines the computation time for one iteration and thus for the whole algorithm.

Table 3: Computation time for reaching an error-bound.

ratio	it.	grid	$\|u-u_h\|_{0,\Omega}$	time (s)
1.0	10	6	7.31e-4	900
8.0	20	5	3.67e-4	200
32.0	38	3	5.34e-4	52
128.0	48	2	5.24e-4	17

Summarizing we can say, that the use of irregular meshes can improve the accuracy of the numerical solution on the one hand side but has brought up the need of new preconditioning techniques on the other hand.
The program *TRANSCG* is the tool for developing and testing new preconditioners and it

can also be used for solving the Poisson problem on a rectangular domain. An extension towards parabolic problems is being developed at present.

7 The routines of the program

7.1 Structure of TRANSCG

TRANSCG

- ↪ INIGEOM
  - ↪ GETNXNY
- ↪ ORIEDG
- ↪ ABFRAGE
- ↪ INIH
  - ↪ DIS1D
- ↪ INIMGCONST
- ↪ INICOARSE
- ↪ INIKOPKE
  - ↪ WADD
- ↪ INILOCGROB
- ↪ SEITE
- ↪ RSINT
- ↪ CG
  - ↪ VDCOPY
  - ↪ VDAXPY
  - ↪ INIBC1
  - ↪ KMALS
    - ↪ MULTI
    - ↪ MULTC
  - ↪ VORKOND
    - ↪ WIAUFW
    - ↪ BASISTRAFO
      - ↪ STARTMG
    - ↪ INKOP
    - ↪ LINADEV
    - ↪ ADDKOPPEL
    - ↪ VKKOPPEL
      - ↪ GETDIR
      - ↪ FFTVKG

↪ VKECK
↪ LINADDVE
↪ KOPIN
↪ WISTAUCH

↪ GRAFIK

↪ DRAWNETZ
↪ DRAWNIVEAUS

↪ GEOMMAKE (external)

↪ GETKONST
↪ GETINDEX
↪ DEFDOM
↪ DEFMAP
↪ DEFMAT
↪ DEFREF
↪ DEFDIR0
↪ PUTDAT
↪ PRINTDAT2

7.2 Description of the Routines

ABFRAGE	Compute the number of intervals over all grids, define the error bound for the cg-algorithm.
ADDKOPPEL	Add up the values across the coupling boundaries.
BASISTRAFO	Perform the basistransformation.
CG	Perform the cg-algorithm.
DEFDIR0	Sets all boundary conditions to homogeneous Dirichlet boundary conditions.
DEFDOM	Defines the cross points, edges, faces (subdomains).
DEFMAP	Defines the mapping of the subdomain onto the hypercube topology.
DEFMAT	Defines the material dependent parameters of the p.d.e..
DEFREF	Defines the user net information on the edges.
DIS1D	Discretize a 1D-interval for given parameters.
DRAWNETZ	Draw the grid (under xrun only).
DRAWNIVEAUS	Draw the niveaus of the solution (under xrun only).
FFTVKG	Perform the preconditioning along an edge via FFT.

GETDIR	Auxiliary routine for determining the step-size along an edge.
GETINDEX	Index calculation in the huge auxiliary array.
GETKONST	Defines several global constants and variables.
GETNXNY	Compute nx, ny, the number of inner points in x-/y-direction.
GRAFIK	Runs the graphic-routines under the xrun-environment.
INIBC1	Initialize the solution-vector with the boundary conditions.
INICOARSE	Compute and decompose the global stiffness matrix for the coarse grid.
INIGEOM	Read data from the input file, generate the discretization parameters HX, HY.
INIH	Compute HX, HY from the input data.
INKOP	Compute $W_C = W_C - K_{CI}Y_I$.
INIKOPKE	Compute the diagonal of that part of the stiffness matrix that belongs to the local crosspoints.
INILOCGROB	Compute and decompose the local stiffness matrices for the coarsest grid.
INIMGCONST	Read and initialize the controlling parameters for the multigrid algorithm.
KMALS	Compute the product K times s, where K is the stiffness matrix (each element is calculated at the moment it is actually needed, no elements are being stored).
KOPIN	Compute $W_I = W_I - K_{IC}Y_C$.
LINADDEV	Compute $W_V = W_V - K_{VE}K_E^{-1}W_E$.
LINADDVE	Compute $W_E = W_E - K_E^{-1}K_{EV}W_V$.
MULTC	Perform the matrix multiplication concerning the inner points.
MULTI	Perform the matrix multiplication concerning the points on the (coupling) boundary.
ORIEDG	Re-ordering of the edges and the corresponding data such that a positively oriented polygon is obtained.
PRINTDAT2	Controls the geometry.
PUTDAT	Writes the data describing the geometry to a file.
RSINT	Integrate the values of the right hand side over the elements.
SEITE	Initialize the right hand side.
STARTMG	Start the multigrid algorithm.
VDAXPY	Basic vector operation ($u + \alpha * v$).
VDCOPY	Basic vector operation ($u = v$).
VKECK	Perform the preconditioning for the crosspoints.
VKKOPPEL	Perform the preconditioning for the points on the coupling boundary.
VORKOND	Perform the preconditioning-step of the cg-algorithm.
WADD	Add up values across the boundaries.
WIAUFW	Uncompress a vector to a vector with boundary points.
WISTAUCH	Compress a vector to a vector without boundary points.

References

[1] Falk Bräuer. Grafische Ergebnisaufbereitung auf Parallelrechnern. Jahresarbeit, TU Chemnitz-Zwickau, FB Mathematik, 1993.

[2] G. Haase, U. Langer, and A. Meyer. The approximate dirichlet domain decomposition method. Part I: An algebraic approach. Part II: Applications to 2nd-order elliptic boundary value problems. *Computing*, 47:137–151 (Part I), 153–167 (Part II), 1991.

[3] Gundolf Haase. Die nichtüberlappende Gebietszerlegungsmethode zur Parallelisierung und Vorkonditionierung des CG-Verfahrens. Report 92-10, IWR Heidelberg, 1992.

[4] Gundolf Haase, Thomas Hommel, Arnd Meyer, and Matthias Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. Preprint-Reihe der Chemnitzer DFG-Forschergruppe "'Scientific Parallel Computing"' SPC 93-1, TU Chemnitz-Zwickau, 1993.

[5] Gundolf Haase, Beate Junghans, and Michael Kuhn. TRANSCG – Ein paralleles Programm zur Lösung der Potentialgleichung in speziellen Gebieten. Programmdokumentation, TU Chemnitz, FB Mathematik, 1993.

[6] Gundolf Haase and Ulrich Langer. The non-overlapping domain decomposition multiplicative Schwarz method. *International Journal of Computer Mathemathics*, 44:223–242, 1992.

[7] Gundolf Haase, Ulrich Langer, and Arnd Meyer. A new approach to the Dirichlet domain decomposition method. In Sabine Hengst, editor, *Proceedings of the "5-th Multigrid Seminar" held at Eberswalde, GDR, May 14-18, 1990*, pages 1–59, Berlin, 1990. Academy of Science. Report-Nr. R-MATH-09/90.

[8] Gundolf Haase, Ulrich Langer, and Arnd Meyer. The approximate dirichlet decomposition method. part I : An algebraic approach. *Computing*, 47:137–151, 1991.

[9] Gundolf Haase, Ulrich Langer, and Arnd Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. In *Parallele Algorithmen auf Transputersystemen*, Teubner-Scripten zur Numerik III, Stuttgart, 1992. Teubner. Tagungsbericht der GAMM-Tagung, 31. Mai- 1. Juni 1991, Heidelberg.

[10] Ulrich Langer. Parallel iterative solution of symmetric coupled fe/be- equations via domain decomposition. *Contemporary Mathematics*, 8:173–177, 1994.