Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

Documentation for the HDD method

by

Alexander Litvinenko

Technical Report no.: 5 2006

Documentation for the HDD Method

A.Litvinenko
litvinen@mis.mpg.de
Max Planck Institute for Mathematics in the Sciences

Abstract

The hierarchical domain decomposition method (HDD method) for solving elliptic differential equations, whose L^∞ coefficients may contain a multiscale parameter, was presented in my dissertation work [3]. This technical report describes the main data structures and procedures of the HDD package as well as some examples. The main idea of the HDD method is to build a large scale solution without computing the solution on the small scale. The $\mathcal{H}$-matrix technique yields the efficient $\mathcal{H}$-matrix arithmetic. It is shown that the storage of HDD is $\mathcal{O}(k\sqrt{n_h n_H}\log^2\sqrt{n_h n_H})$ and the complexity $\mathcal{O}(k^2\sqrt{n_h n_H}\log^3\sqrt{n_h n_H})$, where k is a small rank, n_h and n_H are the numbers of degrees of freedom on fine and coarse grids respectively. In the case of homogeneous right-hand side HDD has linear storage and complexity $\mathcal{O}(k^2\sqrt{n_h n_H})$.
The method was tested on the so-called skin problem with jumping coefficients and on problems with oscillatory coefficients.

Contents

1 Problem Setup

Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, be a polygonal domain. We consider the elliptic boundary value problem with oscillatory coefficients with the Dirichlet boundary conditions:

$$\begin{cases} Lu = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{1}$$

whose coefficients may contain a non-smooth parameter, e.g.,

$$L = -\sum_{i,j=1}^{3} \frac{\partial}{\partial_j} \alpha_{ij} \frac{\partial}{\partial_i} \tag{2}$$

with $\alpha_{ij} = \alpha_{ji}(x) \in L^\infty(\Omega)$ such that the matrix function $A(x) = (\alpha_{ij})_{i,j=1,2,3}$ satisfies $0 < \underline{\lambda} \leq \lambda_{\min}(A(x)) \leq \lambda_{\max}(A(x)) \leq \overline{\lambda}$ for all $x \in \Omega \subset \mathbb{R}^3$. This setting allows us to treat oscillatory coefficients as well as jumping ones.
Let h be a step size, associated with the triangulation $\mathcal{T}_h$ of the domain Ω. We assume $\Omega = \bigcup_{t \in \mathcal{T}_h} \overline{t}$.
For simplicity we consider the P^1-Lagrange finite element discretization of the elliptic problem (1). The vertices of the triangulation $\mathcal{T}_h$ form the set $\{x_i : i \in I\}$, indexed by I.

2 Steps of the Implementation

2.1 The idea of HDD

HDD does not compute the direct inversion of the stiffness matrix L_h. HDD performs some recursive inverting process. This is done step by step by eliminating interior degrees of freedom of the discrete problem. HDD represents the inverse of the stiffness matrix as a set of boundary_to_interface mappings (Φ^g_ω, $\omega \in \Omega$) and domain_to_interface mappings (Φ^f_ω, $\omega \in \Omega$). The solution can be evaluated in the following way:

$$u_h(f_h, g_h) := \Phi^f_\omega f_h + \Phi^g_\omega g_h, \tag{3}$$

where $u_h(f_h, g_h) \in V_h$ is the FE solution of $L_h u_h = f_h - \overline{L}_h \cdot \overline{u}_h$, $\overline{u}_h = g_h$ on $\partial\Omega$, f_h is the discrete right-hand side, g_h is the discrete Dirichlet boundary condition, L_h and $\overline{L}_h$ are discrete operators on Ω and $\overline{\Omega}$.
HDD allows the efficient computation of different functionals of the solution. If Λ_h is such a functional, HDD evaluates $\Lambda_h(\Phi^g, \Phi^f, f_h, g_h)$. Examples are a) Neumann data $\frac{\partial u_n}{\partial n}$ at the boundary, b) u_h at some point, c) a mean value $\int_\omega u_h dx$ for some $\omega \subset \Omega$. HDD also allows to compute $u_h(f_h, g_h)$ for f_h in a smaller space $V_H \subset V_h$.
Due to the oscillatory character of the coefficients we are forced to use a rather small step size h. Since we make no assumptions about a periodic structure of the problem, analytic homogenization methods do not apply. Instead we want to perform a "numerical homogenization".
HDD consists of two algorithms. The first algorithm "Leaves to Root" computes the domain-to-boundary Ψ^f_ω and boundary-to-boundary Ψ^g_ω mappings and after, using the Schur complement, the domain-to-interface Φ^f_ω and boundary-to-interface Φ^g_ω mappings, $\omega \in T_{\mathcal{T}_h}$. The second algorithm "Root to Leaves" computes the solution consistently applying the mappings Φ^f_ω, Φ^g_ω to the right-hand side and to the Dirichlet data.

2.2 Notations

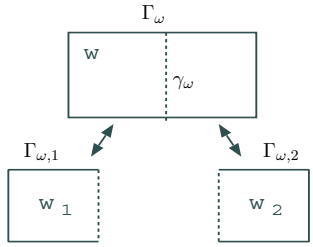
The hierarchical decomposition of the domain Ω results in the tree $T_{\mathcal{T}_h}$.

Definition 2.1 *The tree $T_{\mathcal{T}_h}$ has to satisfy the following properties:*

- Ω *is the root of the tree,*
- $T_{\mathcal{T}_h}$ *is a binary tree,*
- *if $\omega \in T_{\mathcal{T}_h}$ has two sons $\omega_1, \omega_2 \in T_{\mathcal{T}_h}$, then $\omega = \omega_1 \cup \omega_2$ and ω_1, ω_2 intersect at most by their boundaries,*
- $\omega \in T_{\mathcal{T}_h}$ *is a leaf, if and only if $\omega \in \mathcal{T}_h$.*

Let $\{x_i : i \in I\}$, be the set of all nodal points in $\overline{\Omega}$ (including nodal points on the boundary). We define $I(\omega) = \{i \in I;\ x_i \in \overline{\omega}\}$. Similarly, we define $I(\partial\omega)$. The (external) boundary $\partial\omega$ of ω splits into

$$\Gamma_{\omega,1} := \partial\omega \cap \overline{\omega}_1, \quad \Gamma_{\omega,2} := \partial\omega \cap \overline{\omega}_2.$$



Let $\omega \in T_{\mathcal{T}_h}$ be a sub-domain and

$$d_\omega := \Big((f_i)_{i \in I'}, (g_i)_{i \in I(\partial_\omega)} \Big) = (f_\omega, g_\omega) \tag{4}$$

be the values of the right-hand side f and the boundary values g at $x_i \in \partial\omega$. Here $I' := \{i : \mathrm{supp}(f_i) \cap \overset{\circ}{\omega} \neq \oslash\}$. Assuming that the boundary value problem restricted to ω is solvable, we can define the local FE solution by solving the following discrete problem in the variational form:

$$\begin{cases} a_\omega(u_h, b_j) = (f_\omega, b_j)_{L^2(\omega)}, & \forall\ j \in I(\overset{\circ}{\omega}), \\ u_h(x_j) = g_j, & \forall\ j \in I(\partial\omega). \end{cases} \tag{5}$$

Here,

$$b_j(x,y) = \frac{(x-x')(y''-y') - (y-y')(x''-x')}{(x_j-x')(y''-y') - (y_j-y')(x''-x')}$$

is the P^1-Lagrange basis function at (x_j, y_j). (x', y'), (x'', y''), (x_j, y_j) are vertices of a triangle $T \in \mathcal{T}_h$. $a_\omega(\cdot,\cdot)$ is the bilinear form with integration restricted to ω, $a_\omega(b_i, b_j) = \int\limits_\omega \alpha(x) \nabla b_i \cdot \nabla b_j \, dx$ and $(f_\omega, b_j) = \int\limits_\omega f_\omega(x)\, b_j(x)\, dx$.

2.3 Mappings Ψ_ω and Φ_ω

Let us define the linear mapping Ψ_ω which maps the data d_ω given by (4) to the external boundary data $\partial\omega$. The result of the mapping Ψ_ω is defined by its components:

$$\Psi_\omega(d) = (\Psi_\omega(d_\omega))_{i\in I(\partial\omega)} \text{ with } (\Psi_\omega(d_\omega))_i = a_\omega(u_h, b_i) - (f_\omega, b_i)_{L^2(\omega)}\,. \tag{6}$$

By definition, Ψ_ω is linear in (f_ω, g_ω) and can be written as $\Psi_\omega d_\omega = \Psi^f_\omega f_\omega + \Psi^g_\omega g_\omega$.

Let us consider $\omega \in T_{\mathcal{T}_h}$ with two sons ω_1, ω_2. The internal boundary of ω is denoted by γ_ω. Consider again the data d_ω from (4) and define u_h by (5). Then, $\Phi_\omega(d_\omega)$ is defined by the components

$$(\Phi_\omega(d_\omega))_i := u_h(x_i) \quad \forall i \in I(\gamma_\omega). \tag{7}$$

Hence, $\Phi_\omega(d_\omega)$ is the trace of u_h on γ_ω.

2.4 Algorithms "Leaves to Root" and "Root to Leaves"

"Leaves to Root" (see Fig. 1, left)

1. Compute Ψ_ω on all leaves (triangles of $\mathcal{T}_h$). Since the leaves are triangles, we have to compute stiffness matrices from $\mathbb{R}^{3\times 3}$.
   For triangles we rewrite (5) as a system of linear equations
   $Au = c - Au|_{\partial\Omega}$, where $A_{ij} = \int\limits_\Omega \alpha(x)\langle\nabla b_i \nabla b_j\rangle dx$, $c_j = (f, b_j)_{L^2(\omega)} = \int\limits_{\operatorname{supp} b_j} f(x) b_j(x) dx$ and $\operatorname{supp} b_j = \{\overline{T} : T \in \mathcal{T}_h \text{ has } x^j \text{ as a vertex}\}$. To evaluate the integral $\int\limits_T h(x_i)dx$ we use the following trapezoidal rule $\int\limits_T h(x)dx = \frac{1}{3}|T| \sum\limits_{i=1}^{3} h(x_i)$, where x_i, $i = 1..3$, are vertices of the triangle T. For a better precision we use also a 12-point quadrature formula (8) on triangles (see [3]).

2. Recursion from the leaves of $T_{\mathcal{T}_h}$ to the root:
   (a) Compute and store Φ_ω and Ψ_ω from $\Psi_{\omega_1}, \Psi_{\omega_2}$ (see Sections 4.4, 4.5).
   (b) Delete $\Psi_{\omega_1}, \Psi_{\omega_2}$.

The result of this algorithm will be a collection of mappings $\{\Phi_\omega : \omega \in T_{\mathcal{T}_h}\}$. The maps Ψ_ω are only of auxiliary purpose and need not be stored.

Remark 2.1 *To calculate* $\int_T \alpha(x)\langle\nabla b_j, \nabla b_i\rangle dx$ *and* $\int\limits_T f(x) b_j(x) dx$ *numerically we use the basic 3-point quadrature formula on a triangle (see Table 1).*
If $b_i \in P^1$, *then* $\nabla b_i = const, \nabla b_j = const$ *and*

$$\int_T \alpha(x)\langle\nabla b_j, \nabla b_i\rangle dx = \langle\nabla b_j, \nabla b_i\rangle \cdot \int_T \alpha(x)dx = \sum_k a(v_k) w_k, \quad \textit{where}$$
$$v_k = v_k(x,y) \quad \textit{from (9)}, \quad w_k \quad \textit{from Table (1) or (2).} \tag{8}$$

In the case of a higher order of $\alpha(x)\langle\nabla b_j, \nabla b_i\rangle$ one can use 12-point quadrature formula on triangle (see Table (2)). This 12-point rule gives exact values of integrals for a polynomial from P^6 on a triangle.

i	weights w_i	d_{i1}	d_{i2}	d_{i3}
1	0.33(3)	0.5	0.5	0.0
2	0.33(3)	0.0	0.5	0.5
3	0.33(3)	0.5	0.0	0.5

Table 1: The coefficients of the basic 3-point quadrature rule for a triangle (used in (8) and (9)). This rule calculates exactly the value of integral for a polynomial from P^2.

Remark 2.2 *It makes sense to apply the 12-point rule if the discretisation error is comparable with the quadrature rule error. If the discretisation error is larger than the quadrature error, it is reasonable to apply the simple 3-point quadrature rule.*

i	weights w_i	d_{i1}	d_{i2}	d_{i3}
1	0.050844906370207	0.873821971016996	0.063089014491502	0.063089014491502
2	0.050844906370207	0.063089014491502	0.873821971016996	0.063089014491502
3	0.050844906370207	0.063089014491502	0.063089014491502	0.873821971016996
4	0.116786275726379	0.501426509658179	0.249826745170910	0.249826745170910
5	0.116786275726379	0.249826745170910	0.501426509658179	0.249826745170910
6	0.116786275726379	0.249826745170910	0.249826745170910	0.501426509658179
7	0.082851075618374	0.636502499121399	0.310352451033785	0.053145049844816
8	0.082851075618374	0.636502499121399	0.053145049844816	0.310352451033785
9	0.082851075618374	0.310352451033785	0.636502499121399	0.053145049844816
10	0.082851075618374	0.310352451033785	0.053145049844816	0.636502499121399
11	0.082851075618374	0.053145049844816	0.310352451033785	0.636502499121399
12	0.082851075618374	0.053145049844816	0.636502499121399	0.310352451033785

Table 2: The coefficients of the basic 12-point quadrature rule for a triangle (used in (8) and (9)). This rule calculates exactly the value of integral for a polynomial from P^6.

If $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ are coordinates of the vertices of triangle, then we define the points by the following formula (the coefficients d_{ij} are defined in Table 2):

$$v_i(x, y) = (d_{i1}x_1 + d_{i2}x_2 + d_{i3}x_3, d_{i1}y_1 + d_{i2}y_2 + d_{i3}y_3), \quad i = 1, 2, 3. \tag{9}$$

Given $\{\Phi_\omega : \omega \in T_{\mathcal{T}_h}\}$, we can determine the solution u_h for the data $d_\Omega = (f_\Omega, g_\Omega)$ by the following Algorithm:

"Root to Leaves" (see Fig. 1, right)

1. Given $d_\omega = (f_\omega, g_\omega)$, compute the solution u_h on the interior boundary γ_ω by $\Phi_\omega(d_\omega)$.
2. Build the data $d_{\omega_1} = (f_{\omega_1}, g_{\omega_1})$, $d_{\omega_2} = (f_{\omega_2}, g_{\omega_2})$ from $d_\omega = (f_\omega, g_\omega)$ and $g_\gamma := \Phi_\omega(d_\omega)$.

The set of the values $(g_\omega)_{\omega \in T_{\mathcal{T}_h}}$ gives the solution of (5) in all of the domain Ω.

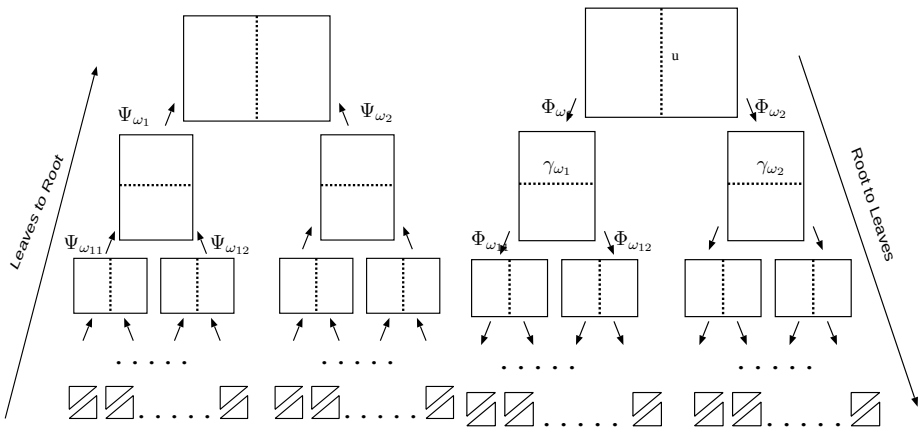


Figure 1: (**left**) Recursive process "Leaves to Root". The mapping Ψ_{ω_1} is a linear function of the mappings $\Psi_{\omega_{11}}$, $\Psi_{\omega_{12}}$. (**right**) Algorithm 'Root to Leaves'. Application of Φ_{ω_i} for computing the solution on the interior boundary γ_{ω_i}.

3 Start of the Program

The result of the implementation is a package of programs which uses the following libraries: HLIB, LAPACK, BLAS (see [5], [6], [7]). The implementation is done in C language (ANSI/ISO standard). The hierarchical matrix library HLIB (see [5]) is used for the $\mathcal{H}$-matrix arithmetic. HLIB uses the linear algebra packages LAPACK (see [6]) and BLAS (see [7]) for the standard matrix-vector arithmetic. The scheme of the implementation is shown in Fig. 2.

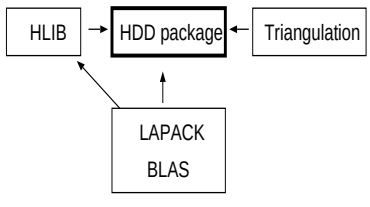


Figure 2: The libraries needed for the HDD package.

3.1 Installation

First, one should download and install HLIB (see more [5]). The successful installation of the HDD package requires the successful installation of HLIB. HLIB requires the presence of LAPACK and BLAS. After the successful installation of HLIB, one should initialize in `makefile` of the HDD directory the following parameters:

```
prefix =
exec_prefix =
CFLAGS =
LDFLAGS =
LIBS =
CC =
```

One can find the needed values in HLIB directory in file `../Examples/Makefile.examples`.

Remark 3.1 *The settings of parameters in the original version of makefile is only an example.*

The HDD package was tested in the UNIX family of operating systems. After successful installation of the LAPACK, BLAS and HLIB in WINDOWS operating systems, the HDD package will work also.

3.2 Input Parameters and Constants

The input parameters (cf. Table 3) and coefficients can be changed within the C source code. The right-hand side $f(x)$, the Dirichlet data $g(x)$, the coefficient function $a(x)$, the analytical solution (for test purposes) are specified in the procedures from Table 3 (see `mylib.c`).

Function	Description
double function_f (double x, double y)	The right-hand side $f(x)$
double function_g (double x, double y)	The Dirichlet data $g(x)$
double function_a (double x, double y, ..)	The coefficients $a(x)$
double function_u (double x, double y)	The test solution $u(x)$

Table 3: Input functions.

The main file is `main.c`. The procedure `init()` initializes all variables and defines constants. The procedures `read_tlist(..)`, `read_edges(..)` (see `laplace.c`) read the lists of triangles, vertices and edges. As an alternative to an external triangulation, one might generate tensor grids by procedures
`grid_generation_eq()` and `two_scale_grid_generation_eq()`. If at the beginning the numbers of vertices and triangles are unknown, we cannot define the sizes of the arrays for storing vertices and triangles. This is why, during the reading of input data, we use the dynamical C structure `list`. To improve the speed of the HDD method we copy all data from `lists` to `arrays` (see procedures in `laplace.c`).
The procedure `build_fine_grid(..)` divides each triangle into four triangles i times. All input parameters for the $\mathcal{H}$-matrix arithmetic and their default values are defined in Table 4.

Constant	Description
nmin=32	The maximal size of a fully-populated block.
global_k=5	The maximal rank of an admissible block for the standard admissibility criteria
global_3k=3·global_k	The maximal rank of an admissible block for the weak admissibility criteria
eps=1e-5	For the adaptive rank arithmetic, rank $k = \min\{i : \sigma_i \leq \varepsilon_a \sigma_1\}$, where $\sigma_1, .., \sigma_n$ are singular values of a matrix.
cg_eps=1e-10	:=Ax-b, residual for the CG method.

Table 4: Input parameters for the $\mathcal{H}$-matrix arithmetic.

3.3 Data Structures

After a triangulation is done we perform the hierarchical domain decomposition of Ω. The set of boundary vertices with their coordinates is the input data for building a triangulation. The triangulation includes the following information: the list of internal and boundary vertices, the list of triangles, the list of edges and for each vertex the list of the adjoint triangles.
Since the HDD method is developed for two and more scales the hierarchy of grids is needed. To build a new grid we divide each triangle of the coarse grid into four triangles. Then we repeat such division and stop when the size of finite elements is small enough. This process provides the hierarchy of grids. Below we describe the data structures which are used in the HDD package. The Diagram in Fig. 3 shows the connection between these structures.

Program 3.1

```
typedef struct _edge edge;
typedef _edge* pedge;

struct _edge{
  int index;     /* Index of the edge *\
  int a,b;       /* Verticed describing the edge *\
  int father;    /* Father of the edge *\
  int son[2];    /* Two sons of the edge *\
  int property;  /* =2 if the edge belongs to boundary, =1 otherwise *\
}
```

Program 3.2

```
typedef struct _telement telement;

struct _telement{
  int    index;             /* Index of the triangle *\
  int    property;          /* Attribute (external, internal, etc) *\
  struct _vertex **ver;     /* Array of pointers to vertices *\
  double area;              /* Area of the triangle *\
  double xmit, ymit;        /* Middle of the triangle *\
  struct _telement **neigh; /* Array of pointers to the neighbouring triangles *\
  struct _telement *father; /* Father of the triangle *\
  double* bij;              /* Stiffness matrix for the triangle *\
  int    edges[3];          /* Edges of the triangle *\
  int    flag;              /* Auxiliary parameter *\
}
```

Program 3.3

```
typedef struct _vertex vertex;

struct _vertex{
  int    index, cindex; /* Indices of the node on a fine grid and a coarse grid *\
  int    coarse;        /* =1 belong to the coarse grid, =0 only to the fine grid *\
  struct _tlist *list;  /* list of adjoining triangles *\
  double* koor;         /* Coordinates of the vertex *\
  struct _telement *tcoarse; /* Father triangle *\
  int    flag;          /* Auxiliary parameter *\
  int    randattr;      /* Attribute (external, internal, etc) *\
}
```

We combine the lists of vertices and triangles in $\omega \in T_{\mathcal{T}_h}$ to the new structure `grid`.

Program 3.4

```
typedef struct _grid grid;
typedef _grid* pgrid;

struct _grid{
  int         deep;
  tlist*      tl;
  vertexlist* vl;
  pedge*      edges;
  int         edgesize;
}
```

For describing a subdomain $\omega \in T_{\mathcal{T}_h}$ the structure `domain` is used.

Program 3.5

```
typedef struct _domain domain;
typedef _domain* pdomain;

struct _domain{
  long        index;  /* Index of the domain */
  tlist*      tl;     /* List of triangles at the fine scale */
  tlist*      ctl;    /* List of triangles at the coarse scale */
  vertexlist* vl;     /* List of vertices at the fine grid */
  vertexlist* cvl;    /* List of vertices at the coarse grid */
  double      area;   /* Area of the domain */
  double      minx,maxx,miny,maxy; /* Describe the boundary box */
}
```

For describing the external $\partial\omega$ and internal γ_ω boundaries the structure `boundary` is used.

Program 3.6

```
typedef struct _boundary boundary;
typedef _boundary* pboundary;

struct _boundary{
  vertexlist*  vl;   /* List of vertices at the fine grid   */
  vertexlist*  cvl;  /* List of vertices at the coarse grid */
  tlist*       tl;   /* List of triangles at the fine scale */
  psupermatrix frhs; /* To store the corresponding hierarchical matrix  */
}
```

For describing the HDD tree $T_{\mathcal{T}_h}$ the structure `DDTree` is used.

Program 3.7

```
typedef struct _DDTree DDTree;
typedef _DDTree* pDDTree;

struct _DDTree{
  long index;        /* Index of the subdomain *\
  pDomain clus;      /* Pointer to the corresponding domain *\
  pDDTree leftTree;  /* Pointer to the left son *\
  pDDTree rightTree; /* Pointer to the right son  *\
  pDDTree father;    /* Pointer to father *\
  pDDTree brother;   /* Pointer to brother *\

  psupermatrix invA22;
  prkmatrix    phi_g;
  psupermatrix psi;

  double *functional_g, *functional_f;
  int *father2sonL, *father2sonR;
  int  ind_removerow[2], ind_insertrow[2];
  int *dof2idx;
  int  compute;      /* =1 if for this domain matrices are computed, =0 else *\
  int  simple;       /* strategy of building H-matrix (=1 or =2) *\
```

```
  pclustertree interct;  /* Cluster tree for the internal boundary (fine grid) */
  pclustertree cinterct; /* Cluster tree for the domain (coarse grid)*/
  pclustertree ect;      /* Cluster tree for the external boundary (fine grid) */
  pclustertree cect;     /* Cluster tree for the external boundary (coarse grid) */
  pclustertree ct;       /* Cluster tree for the domain (fine grid) */
  pclustertree cct;      /* Cluster tree for the domain (coarse grid) */
  pclustertree cl_Gamma; /* Auxiliary cluster tree */
  pclustertree cl_gamma; /* Auxiliary cluster tree */

  pdomain   clus;        /* Pointer to the domain*/
  pboundary eclus;       /* Pointer to the external boundary */
  pboundary interclus;   /* Pointer to the internal boundary */
  int       *cf_index;   /* Auxiliary array. Used for the mesh refinement*/

}
```

To store the inverse of the mapping $\Psi^g_\omega|_{I(\gamma)} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)}$ the field `invA22` is used, to store the mapping $\Phi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}$ the field `phi_g` is used. To store the mapping $\Psi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\partial\omega)}$ the field `psi` is used. The fields `functional_g`, `functional_f` are needed to store the functionals λ^g_ω and λ^f_ω. The fields `father2sonL`, `father2sonR` are used for storing the mappings $I(\omega) \to I(\omega_1)$ and $I(\omega) \to I(\omega_2)$. The fields `ind_removerow[2]`, `ind_insertrow[2]` store indices from $I(\partial\omega)$ and define which rows should be removed from an $\mathcal{H}$-matrix. The field `dof2idx` maps the set of degrees of freedom on $\partial\omega \cup \gamma$ to the set of indices $I(\partial\omega \cup \gamma)$.

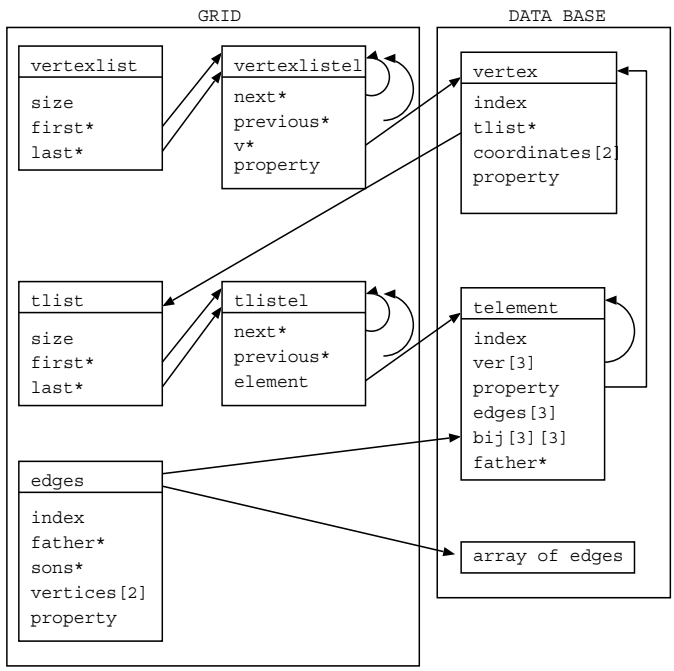


Figure 3: Implementation of the structures for storing vertices, triangles and edges.

3.4 Compilation and Starting of the Program

The HDD package does not need to be configured. To compile the program one should type:
`make`
and to start the program:
`./elaplace`

3.5 Main Steps of the Program

Let $\omega, \omega_1, \omega_2 \in T_{\mathcal{T}_h}$, $\omega = \omega_1 \cup \omega_2$. The main steps of the HDD program are:

1. read the coarse grid $\mathcal{T}_H$ (procedures `read_tlist`, `read_edges`),
2. refine $\mathcal{T}_H$ i_{max} times (procedure `build_fine_grid(..)`),
3. build the HDD tree $T_{\mathcal{T}_h}$ (procedure `divide_tree(..)`),
4. execute "Leaves to Root" (procedure `recursive_process(..)`),
   (a) build Ψ^g_ω and Ψ^f_ω for all leaves of $T_{\mathcal{T}_h}$ (`build_Psi_g_full(..)`),
   (b) build Ψ^g_ω from $\Psi^g_{\omega_1}$ and $\Psi^g_{\omega_2}$ (`build_Psi_g(..)` and `build_Psi_g_fast(..)`),
   (c) build Ψ^f_ω and Φ^f_ω from $\Psi^f_{\omega_1}$ and $\Psi^f_{\omega_2}$ (`build_Psi_f(..)` and `build_Psi_f_fast(..)`),
   (d) build Φ^g_ω, $\omega \in T_{\mathcal{T}_h}$ (`Schur_complement(..)`),
   (e) compute the functionals $\lambda_\omega := (\lambda^g_\omega, \lambda^f_\omega)$ (`build_father_functional(..)`, `build_father_functional_after(..)`),
   (f) repeat steps (a)-(f) for sons ω_1 and ω_2 of $T_{\mathcal{T}_h}$.
5. execute "Root to Leaves":
   (a) compute $u_{\gamma_\omega} := \Phi^g_\omega \cdot g|_{I(\partial\omega)} + \Phi^f_\omega \cdot f|_{I(\omega)}$, $\omega \in T_{\mathcal{T}_h}$, (`root_to_leaves(..)`),
   (b) compute a functional, e.g., the mean value $\lambda_\omega(d_\omega) = (\lambda^f_\omega, f) + (\lambda^g_\omega, g)$, $\omega \in T_{\mathcal{T}_h}$, (`compute_functional(..)`),
   (c) repeat steps (a)-(c) for sons ω_1 and ω_2 of $T_{\mathcal{T}_h}$.
6. compute the solution by the CG method and compare it with the solution obtained earlier by HDD (`solve_by_cg_method(..)`).

4 Other Important Algorithms

4.1 Generation of a Hierarchy of Grids

It is not enough to have either one or two scales for solving multiscale problems. Below we discuss how to implement the hierarchy of grids $\mathcal{T}_h \subset \mathcal{T}_{h/2} \subset ... \subset \mathcal{T}_{h/2^q}$. All grids must be connected with each other. Each finite element has to know his predecessor and each father has to know his descendants. We build a grid $\mathcal{T}_h$ with the step size h, refine it, obtain a grid $\mathcal{T}_{h/2}$, refine it again and so on.
In the dissertation [3] two grids $\mathcal{T}_{h/2^i}$ and $\mathcal{T}_{h/2^j}$, $0 \leq i, j \leq q$, are used, but for more difficult problems more grids (scales) should be used. If we are only interested in two scales with $H/h > 2$, we refine the given scale recursively and do not store intermediate grids. After each recursive step, we reorganize the connections predecessor↔descendant.

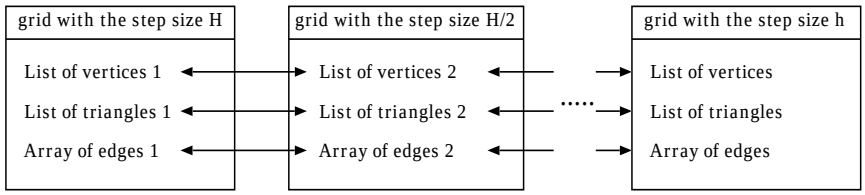


Figure 4: Connection of the grids $\mathcal{T}_H$, $\mathcal{T}_{H/2}$,..., $\mathcal{T}_h$.

The following procedures are used for the grid refinement:

procedure	description
build_fine_grid(..)	Refines the given grid
new_grid()	Allocates memory for a new grid
refine_grid(..)	Performs one refinement of the given grid
copy_links(..)	Copies all links predecessor↔descendant

Table 5: The procedures which are applied for the grid refinement.

4.2 New $\mathcal{H}$-matrix procedures

In the HDD package some useful $\mathcal{H}$-matrix procedures were implemented (see Table 6).

procedure	description
extract_col()	get a column from an $\mathcal{H}$-matrix
extract_row()	get a row from an $\mathcal{H}$-matrix
remove_colrow()	remove a row and a column from an $\mathcal{H}$-matrix and then copy the rest to the new $\mathcal{H}$-matrix
remove_col()	remove a column from an $\mathcal{H}$-matrix
remove_row()	remove a row from an $\mathcal{H}$-matrix
add_col_toHmatrix()	Add a rank-1 matrix to a column of an $\mathcal{H}$-matrix
add_row_toHmatrix()	Add a rank-1 matrix to a row of an $\mathcal{H}$-matrix
test_permute_f()	permutation of rows in a dense matrix
test_permute_rk()	permutation of rows in a low-rank matrix
h2r()	Conversion an $\mathcal{H}$-matrix to an low-rank matrix
h2r_fast()	Fast conversion an $\mathcal{H}$-matrix to an low-rank matrix
h2f()	Conversion an $\mathcal{H}$-matrix to an dense matrix
h2h()	Conversion of an $\mathcal{H}$-matrix to another $\mathcal{H}$-matrix
h2h_fast()	Fast conversion of an $\mathcal{H}$-matrix to another $\mathcal{H}$-matrix

Table 6: New $\mathcal{H}$-matrix procedures.

4.3 $\mathcal{H}$-matrix Conversion

Let I, J, I', J', I'' and J'' be given index sets such that $I', I'' \subseteq I$, $J', J'' \subseteq J$, and $M \in \mathcal{H}(T_{I\times J}, k)$. The sum of $M_1 \in \mathcal{H}(T'_{I'\times J'}, k_1)$ and $M_2 \in \mathcal{H}(T''_{I''\times J''}, k_2)$ with the result matrix M is defined as follows (see Fig. 5):

$$M = M' \oplus M'', \quad \text{where } M' := M_1|^{I\times J} \text{ and } M'' := M_2|^{I\times J}.$$

The adding procedure applies the list of procedures from Table 7.

Remark 4.1 *Note that $M_1|^{I\times J}$ and $M_2|^{I\times J}$ have the same block cluster structures. To compute $M' := M_1|^{I\times J}$ and $M'' := M_2|^{I\times J}$ we apply the procedure* `h2h(..)`.

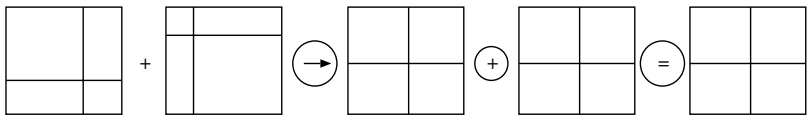


Figure 5: Transformation of $\mathcal{H}$-matrices M_1, M_2 to $\mathcal{H}$-matrices $M_1|^{I\times J}$, $M_2|^{I\times J}$ and their addition.

Consider the more difficult case. Suppose

$$I \supseteq I' = \bigcup_{i=1}^{p} I_i, \quad I_j \cap I_k = \varnothing, j \neq k, \tag{10}$$

$$J \supseteq J' = \bigcup_{j=1}^{q} J_j, \quad J_i \cap J_k = \varnothing, i \neq k \tag{11}$$

and $n = \max\{|I|, |J|\}$, $n' = \max\{|I'|, |J'|\}$. Let $\tilde{M} \in \mathcal{H}(T'_{I'\times J'}, k)$, $M \in \mathcal{H}(T_{I\times J}, k)$, $R \in \mathcal{R}(k, I', J')$, where I, J, I', J' are from (10, 11). The problem is to convert $\tilde{M}$ to M. Algorithm 4.1 performs this conversion.

Algorithm 4.1 *(Conversion $M \in \mathcal{H}(T_{I\times J}, k)$ to $M' := M|_{I'\times J'} \in \mathcal{H}(T'_{I'\times J'}, k)$)*

h2h$(M,\ M',\ \bigcup_{i=1}^{p} I_i,\ \bigcup_{j=1}^{q} J_j)$
begin
  if *(M' is a dense matrix)* ***then***
    h2f(M, M', $\bigcup_{i=1}^{p} I_i$, $\bigcup_{j=1}^{q} J_j$);
  if *(M' is a low-rank matrix)* ***then***
    *h2r(M, M', $\bigcup_{i=1}^{p} I_i$, $\bigcup_{j=1}^{q} J_j$);/*see Algorithm 4.2*/*
  if *(M' is an $\mathcal{H}$-matrix)* ***then***
    for each *subblock $b = (t, s)$ of M'* ***do***
      h2h(M, $M'|_b$, $\bigcup_{i=1}^{p} I_i \cap \widehat{t}$, $\bigcup_{j=1}^{q} J_j \cap \widehat{s}$);
  end if*;*
end*;*

Algorithm 4.2 *(Converting $M \in \mathcal{H}(T_{I\times J}, k)$ to $R \in \mathcal{R}(k, I', J')$)*

h2r$(M,\ R,\ \bigcup_{i=1}^{p} I_i,\ \bigcup_{j=1}^{q} J_j)$
begin
  if *(M is a dense matrix)*
    Create a new dense matrix $F \in \mathbb{R}^{I'\times J'}$;
    for all *i,j* ***do***
      copy $M|_{I_i\times J_j}$ to F;
    convert F to R;
  end if*;*
  if *(M is a low-rank matrix)* ***then***
    Create a new low-rank matrix $R \in \mathcal{R}(k, I', J')$;
    for all *i,j* ***do***
      copy $M|_{I_i\times J_j}$ to R;
  end if*;*
  if *(M is an $\mathcal{H}$-matrix)* ***then***
    for each *subblock $b = (t, s)$ of M* ***do***
      $R[l]$:=h2r($M|_b$, $\bigcup_{i=1}^{p} I_i \cap \widehat{t}$, $\bigcup_{j=1}^{q} J_j \cap \widehat{s}$);
    end for
    R:=$(R[0] \oplus_k (R[1] \oplus_k ... \oplus_k (R[l-2] \oplus_k R[l-1])..)$;
  end if*;*
end*;*

procedure	description
add_fullmatrix(F, F_1, F_2)	Adding two dense matrices
add_rkmatrix(R, R_1, R_2)	Adding two low-rank matrices
addfullpart2_rkmatrix(F,R)	Addition of a dense matrix to a low-rank matrix
addrk2_fullmatrix(R,F)	Addition of a low-rank matrix to a dense matrix
addfull2_supermatrix(F, H)	Addition of a dense matrix to an $\mathcal{H}$-matrix
addrk2_supermatrix(R, H)	Addition of a low-rank matrix to an $\mathcal{H}$-matrix
add_supermatrix(H, H_1, H_2)	Adding of $\mathcal{H}$-matrices $H := H_1 \oplus H_2$
h2r(H, R)	Conversion an $\mathcal{H}$-matrix to an low-rank matrix
h2r_fast(H, R)	Fast conversion an $\mathcal{H}$-matrix to an low-rank matrix
h2f(H, F)	Conversion an $\mathcal{H}$-matrix to an dense matrix
h2h$(M \in \mathcal{H}(T_{I'\times J'}, k), M\|^{I\times J})$	Conversion of an $\mathcal{H}$-matrix M to the $\mathcal{H}$-matrix $M\|^{I\times J}$
h2h_fast$(M \in \mathcal{H}(T_{I'\times J'}, k), M\|^{I\times J})$	Fast conversion of an $\mathcal{H}$-matrix M to the $\mathcal{H}$-matrix $M\|^{I\times J}$

Table 7: The procedures which are applied for adding two $\mathcal{H}$-matrices with different block structures. H, H_i are $\mathcal{H}$-matrices, R, R_i are low-rank matrices, F, F_i are dense matrices, $i = 1, 2$.

4.4 Building of Ψ^g_ω from $\Psi^g_{\omega_1}$ and $\Psi^g_{\omega_2}$

Denote

$$H_1 := (\Psi^g_{\omega_1})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_1)\times I(\partial\omega_1)}, k), \quad H_2 := (\Psi^g_{\omega_2})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_2)\times I(\partial\omega_2)}, k), \tag{12}$$

$$\tilde{H} \in \mathcal{H}(T_{I(\partial\omega\cup\gamma)\times I(\partial\omega\cup\gamma)}, k), \quad H := (\Psi^g_{\omega})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega)\times I(\partial\omega)}, k), \tag{13}$$

where $I(\partial\omega \cup \gamma) = I(\partial\omega_1 \cup \partial\omega_2)$ (see Figures 6, 7). We want to construct the matrix H from H_1 and H_2. First, we build a new cluster tree $T_{I(\partial\omega\cup\gamma)}$ from the clusters $T_{I(\partial\omega)}$ and $T_{I(\gamma)}$. There are many variants of how to build it, but we want such a cluster tree, which makes it easier to eliminate the unknowns x_i, $i \in I(\gamma)$, i.e, one of the sons of the cluster $I(\partial\omega \cup \gamma)$ should coincide with the index set $I(\gamma)$. As soon as the cluster tree $T_{I(\partial\omega\cup\gamma)}$ is built, we build the block cluster tree $T_{I(\partial\omega\cup\gamma)\times I(\partial\omega\cup\gamma)}$. The block cluster tree $T_{I(\partial\omega\cup\gamma)\times I(\partial\omega\cup\gamma)}$ defines the block structure of $\tilde{H}$.

Further the external boundary of ω we denote by $\partial\omega$ and for brevity we write Γ_i besides $\Gamma_{\omega,i}$. We consider two variants of the block structures:

$$I(\Gamma_i) \times I(\Gamma_i), I(\gamma) \times I(\gamma) \in T_{I(\partial\omega_i)\times I(\partial\omega_i)}, i = 1, 2. \tag{14}$$

$$I(\Gamma_i) \times I(\Gamma_i) \notin T_{I(\partial\omega_i)\times I(\partial\omega_i)} \text{ or } I(\gamma) \times I(\gamma) \notin T_{I(\partial\omega_i)\times I(\partial\omega_i)}, i = 1, 2. \tag{15}$$

Building algorithm in case (14):
Let H_1 and H_2 be defined as in (12) and H as in (13).

Algorithm 4.3 *Building* $H := (\Psi_\omega^g)^{\mathcal{H}}$ *from* $H_1 := (\Psi_{\omega_1}^g)^{\mathcal{H}}$ *and* $H_2 := (\Psi_{\omega_2}^g)^{\mathcal{H}}$
build_*Ψ^g*(*H_1*,* H_2*,* H*)
begin
create $\tilde{H}$*;*
$\tilde{H}|_{I(\Gamma_1)\times I(\Gamma_1)} := H_1|_{I(\Gamma_1)\times I(\Gamma_1)}$*;*
$\tilde{H}|_{I(\Gamma_2)\times I(\Gamma_2)} := H_2|_{I(\Gamma_2)\times I(\Gamma_2)}$*;*
$\tilde{H}|_{I(\Gamma_1)\times I(\Gamma_2)} := 0$*;*
$\tilde{H}|_{I(\Gamma_2)\times I(\Gamma_1)} := 0$*;*
/* *in Fig. 6 denoted by* $d + h$ */
$\tilde{H}|_{I(\gamma)\times I(\gamma)} := H_1|_{I(\gamma)\times I(\gamma)} \oplus H_2|_{I(\gamma)\times I(\gamma)}$*;*
$\tilde{H}|_{I(\gamma)\times I(\Gamma_1)\cup I(\Gamma_2)} := (H_1|_{I(\gamma)\times I(\Gamma_1)} \oplus H_2|_{I(\gamma)\times I(\Gamma_2)})$*;* /* *Sum of two low-rank matrices* */
/* *in Fig. 6 denoted by* $b + f$ */
$\tilde{H}|_{I(\Gamma_1)\cup I(\Gamma_2)\times I(\gamma)} := H_1|_{I(\Gamma_1)\times I(\gamma)} \oplus H_2|_{I(\Gamma_2)\times I(\gamma)}$*;* /* *Sum of two low-rank matrices* */
$\tilde{H}$*:=extract_rows(*$\tilde{H}$*,* r_1*,* r_2*,* i_1*,* i_2*);* /* *The output is* r_1*,* r_2 */
$\tilde{H}$*:=extract_columns(*$\tilde{H}$*,* c_1*,* c_2*,* j_1*,* j_2*);* /* *The output is* c_1*,* c_2 */
$\tilde{H}$*:=add_rows(*$\tilde{H}$*,* r_1*,* r_2*,* i_3*,* i_4*);*
$\tilde{H}$*:=add_columns(*$\tilde{H}$*,* c_1*,* c_2*,* j_3*,* j_4*);*
H*:=elimination(*$\tilde{H}$*,* $I(\partial\omega_1 \setminus \partial\omega)$*);* /* *see Algorithm 4.4*/
return H*;*
end*;*

Here i_1, i_2, j_1, j_2 are the positions of two rows and two columns which have to be extracted, i_3, i_4 and j_3, j_4 are positions of two rows and two columns to which the four rank-1 matrices r_1, r_2, c_1 and c_2 have to be added.

The elimination of unknowns x_i, $i \in I$, is done by Algorithm 4.4.

Algorithm 4.4 *(Elimination of* u_i*,* $i \in I(\gamma)$*)*
elimination*(block matrix* M*,* I*)*
begin
$M_{11} := M[0]$*;*
$M_{21} := M[1]$*;*
$M_{12} := M[2]$*;*
$M_{22} := M[3]$*;* /* *Corresponds to the set* I */
$\tilde{M}_{11} := M_{11} \ominus M_{12} \odot M_{22}^{-1} \odot M_{21}$*;*
return $\tilde{M}_{11}$*;*
end*;*

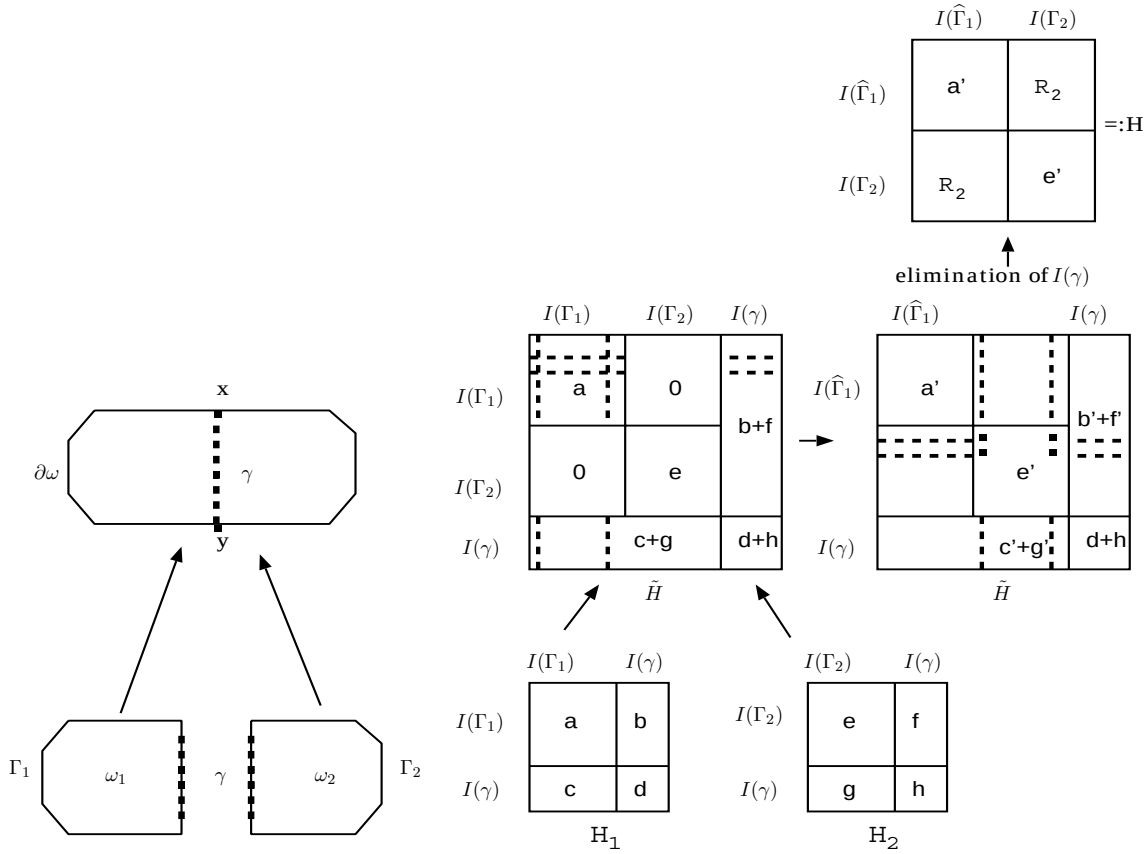


Figure 6: Building $H := (\Psi^g_\omega)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\partial\omega)}$ from $H_1 := (\Psi^g_{\omega_1})^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\partial\omega_1)}$ and $H_2 := (\Psi^g_{\omega_2})^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\partial\omega_2)}$, $\Gamma_i \cup \gamma = \partial\omega_i$, $i = 1, 2$, $\partial\omega = \Gamma_1 \cup \Gamma_2$, $\widehat{\Gamma}_1 = \Gamma_1 \setminus \{x, y\}$. The small letters define the corresponding blocks in different matrices. The dotted lines in $\tilde{H}$ present 2 rows and 2 columns.

Building algorithm in case (15):
Let H_1 and H_2 be defined as in (12), H as in (13) and $I := I(\partial\omega)$.

Algorithm 4.5 *Building* $H := (\Psi^g_\omega)^{\mathcal{H}}$ *from* $H_1 := (\Psi^g_{\omega_1})^{\mathcal{H}}$ *and* $H_2 := (\Psi^g_{\omega_2})^{\mathcal{H}}$

build_*Ψ^g*(*H_1*,* H_2*,* H*)
begin
H'*:=copy_block_structure(*H*);*
H''*:=copy_block_structure(*H*);*
h2h(H_1*,* H'*,...);*$/*$*$ *Convert* H_1 *to* H' $*$*/*
h2h(H_2*,* H''*,...);*$/*$*$ *Convert* H_2 *to* H'' $*$*/*
$\tilde{H} := H'' \oplus H'$*;* $/*$ *See (13)* $*/$
H*:=elimination(*$\tilde{H}$*,* $I(\partial\omega_1 \setminus \partial\omega)$*);* $/*$ *see Algorithm 4.4* $*/$
return H*;*
end*;*

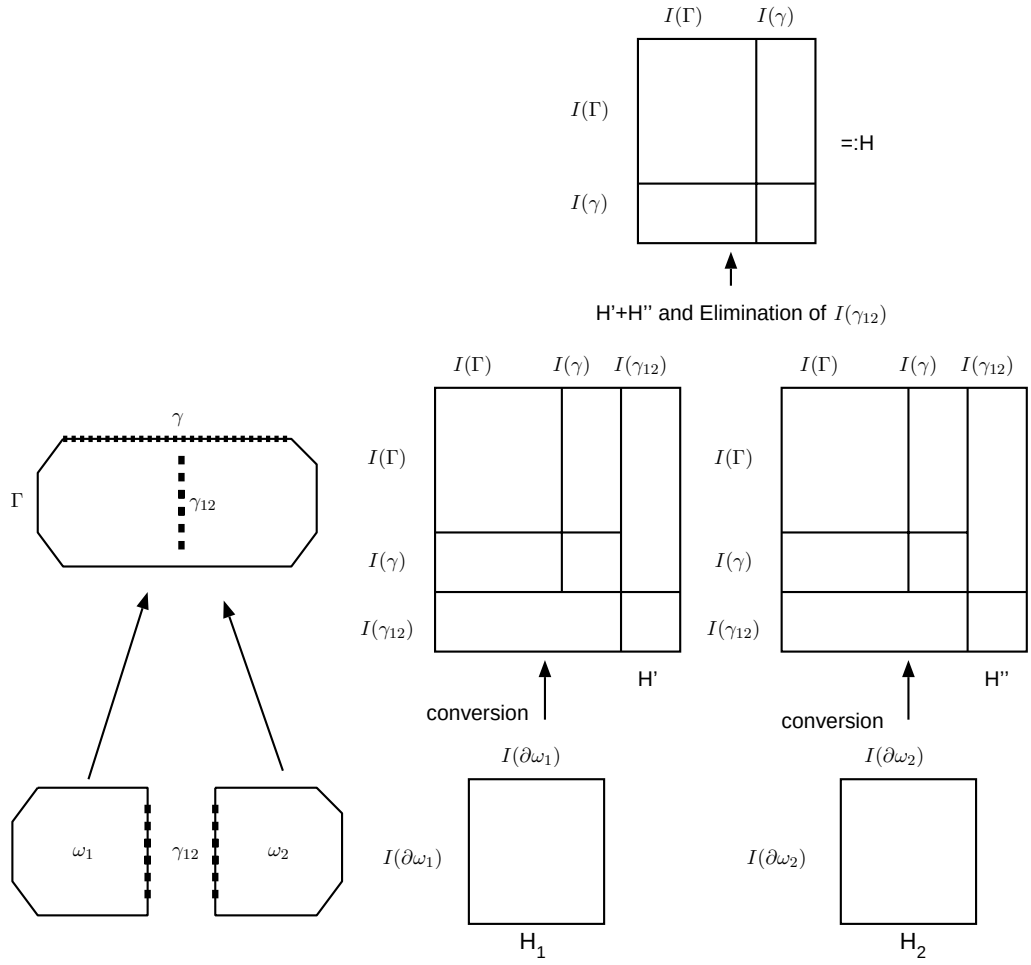


Figure 7: Building $(\Psi^g_\omega)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\partial\omega)}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\partial\omega_1)}$ and $(\Psi^g_{\omega_2})^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\partial\omega_2)}$, $\Gamma_i \cup \gamma_{12} = \partial\omega_i$, $i = 1, 2$, $\partial\omega = \Gamma \cup \gamma$. $H = H' + H''$, $H' := H_1|^{I(\Gamma)\cup I(\gamma)\cup I(\gamma_{12})}$, $H'' := H_2|^{I(\Gamma)\cup I(\gamma)\cup I(\gamma_{12})}$, $H := \tilde{H}_{11} \ominus \tilde{H}_{12} \odot \tilde{H}_{22}^{-1} \odot \tilde{H}_{21}$.

Example 4.1 *In Fig. 8 we show an example of building* $(\Psi^g_\omega)^{\mathcal{H}} \in \mathbb{R}^{512\times512}$ *from* $(\Psi^g_{\omega_1})^{\mathcal{H}} \in \mathbb{R}^{384\times384}$ *and* $(\Psi^g_{\omega_2})^{\mathcal{H}} \in \mathbb{R}^{384\times384}$*. The construction is performed in three steps. First, we build* $H' := (\Psi^g_{\omega_1})^{\mathcal{H}}|^{I\times I}$ *and* $H'' := (\Psi^g_{\omega_2})^{\mathcal{H}}|^{I\times I}$*,* $I := I(\partial\omega \cup \gamma)$*. Second, we sum* H' *and* H''*. Third, we eliminate* x_i*,* $i \in I(\gamma)$*. Note that* H'*,* H''*,* $\tilde{H}$ *have the same block structures.*

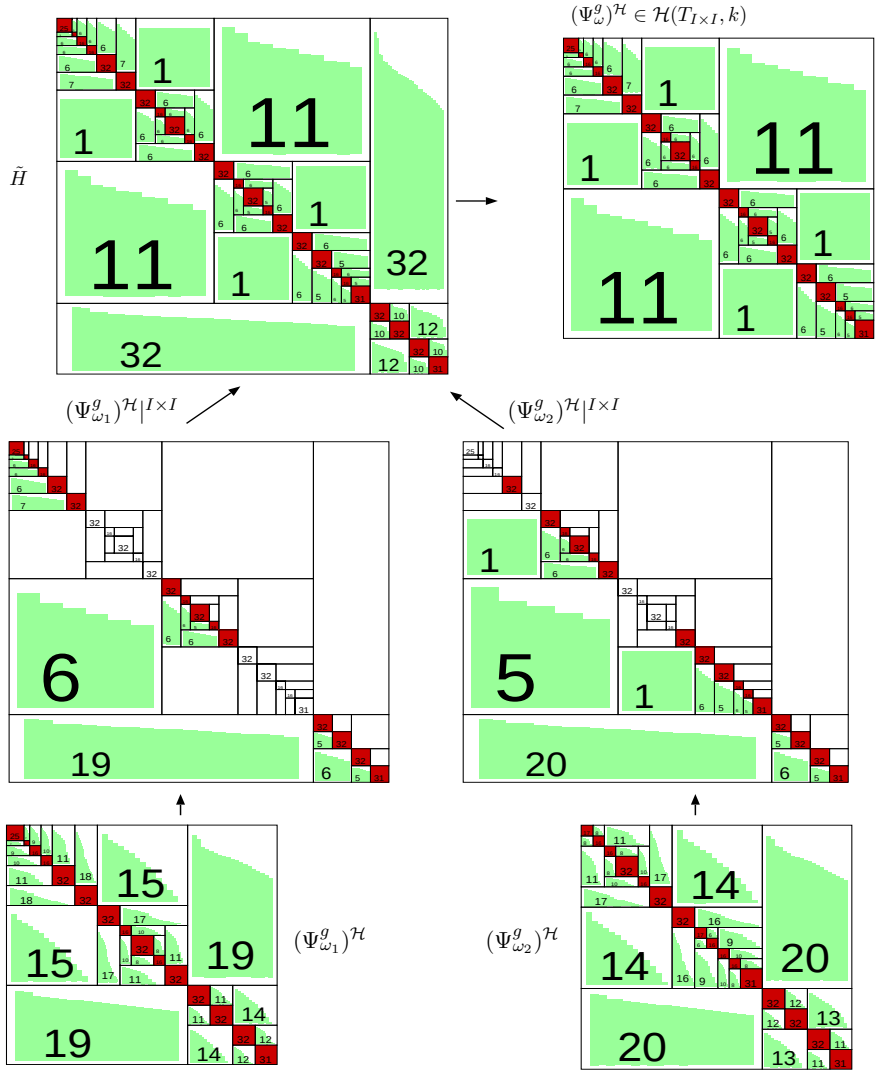


Figure 8: Building $(\Psi^g_\omega)^{\mathcal{H}} \in \mathbb{R}^{512\times512}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}}$ and $(\Psi^g_{\omega_2})^{\mathcal{H}}$ from $\mathbb{R}^{384\times384}$. The temporary matrix is $\tilde{H} \in \mathbb{R}^{639\times639}$. The maximal size of the diagonal blocks is 32×32. The grey blocks indicate low-rank matrices. The steps inside the grey blocks show an exponential decay of the corresponding singular values. The white blocks indicate zero blocks. For the acceleration of building the symmetry of Ψ^g_ω is used.

4.5 Building of Ψ_ω^f from $\Psi_{\omega_1}^f$ and $\Psi_{\omega_2}^f$

Denote

$$H_1 := (\Psi_{\omega_1}^f)^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_1)\times I(\omega_1)}, k), \quad H_2 := (\Psi_{\omega_2}^f)^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_2)\times I(\omega_2)}, k) \tag{16}$$

and

$$\tilde{H} \in \mathcal{H}(T_{I(\partial\omega\cup\gamma)\times I(\omega)}, k), \quad H := (\Psi_\omega^f)^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega)\times I(\omega)}, k). \tag{17}$$

Let $T := T_{I(\partial\omega\cup\gamma)\times I(\omega)}$. We want to construct the matrix H from H_1 and H_2. Note that $\partial\omega \cup \gamma = \partial\omega_1 \cup \partial\omega_2$, $I(\partial\omega_i) = I(\Gamma_i) \cup I(\gamma)$, $\Gamma_1 \cup \Gamma_2 = \partial\omega$. To build the matrix H we need two cluster trees $T_{I(\partial\omega\cup\gamma)}$ and $T_{I(\omega)}$. The first cluster tree was already built for $(\Psi_\omega^g)^{\mathcal{H}}$. There are many possibilities of how to build $T_{I(\omega)}$, but we want a tree which makes a further elimination of unknowns x_i, $i \in I(\gamma)$ easier, i.e., one of the sons of the block cluster tree T has to coincide with the block $I(\gamma) \times I(\gamma)$. Therefore we choose the following decomposition:

$$I(\omega) = I(\omega_1 \setminus \gamma) \cup I(\omega_2 \setminus \gamma) \cup I(\gamma).$$

There are two cases:

$$I(\Gamma_i) \times I(\omega_i \setminus \gamma) \in T_{I(\partial\omega_i)\times I(\omega_i)} \text{ and } I(\gamma) \times I(\gamma) \in T_{I(\partial\omega_i)\times I(\omega_i)}, i = 1, 2. \tag{18}$$

$$I(\Gamma_i) \times I(\omega_i \setminus \gamma) \notin T_{I(\partial\omega_i)\times I(\omega_i)} \text{ or } I(\gamma) \times I(\gamma) \notin T_{I(\partial\omega_i)\times I(\omega_i)}, i = 1, 2. \tag{19}$$

Building algorithm in case (18):
Let H_1 and H_2 be defined as in (16) and H as in (17).

Algorithm 4.6 *Build* $H := (\Psi_\omega^f)^{\mathcal{H}}$ *from* $H_1 := (\Psi_{\omega_1}^f)^{\mathcal{H}}$ *and* $H_2 := (\Psi_{\omega_2}^f)^{\mathcal{H}}$

build_*Ψ^f*(*H_1*,* H_2*,* H*)
begin
create $\tilde{H}$*;*
$\tilde{H}|_{I(\Gamma_1)\times I(\omega_1\setminus\gamma)} := H_1|_{I(\Gamma_1)\times I(\omega_1\setminus\gamma)}$*;*
$\tilde{H}|_{I(\Gamma_2)\times I(\omega_2\setminus\gamma)} := H_2|_{I(\Gamma_2)\times I(\omega_2\setminus\gamma)}$*;*
$\tilde{H}|_{I(\Gamma_1)\times I(\omega_2\setminus\gamma)} := 0$*;*
$\tilde{H}|_{I(\Gamma_2)\times I(\omega_1\setminus\gamma)} := 0$*;*
/ in Fig. 9 denoted by [cg] */*
$\tilde{H}|_{I(\gamma)\times I(\omega_1\setminus\gamma)\cup I(\omega_1\setminus\gamma)} := H_1|_{I(\gamma)\times I(\omega_1\setminus\gamma)} \oplus H_2|_{I(\gamma)\times I(\omega_2\setminus\gamma)}$*;*
/ in Fig. 9 denoted by b + f */*
$\tilde{H}|_{I(\Gamma_1)\cup I(\Gamma_2)\times I(\gamma)} := H_1|_{I(\Gamma_1)\times I(\gamma)} \oplus H_2|_{I(\Gamma_2)\times I(\gamma)}$*; /*sum of two low-rank matrices*/*
$\tilde{H}$*:=extract_rows(*$\tilde{H}$*,* r_1*,* r_2*,* i_1*,* i_2*);/* The output is* r_1*,* r_2 **/*
$\tilde{H}$*:=extract_columns(*$\tilde{H}$*,* c_1*,* c_2*,* j_1*,* j_2*);/* The output is* c_1*,* c_2 **/*
$\tilde{H}$*:=add_rows(*$\tilde{H}$*,* r_1*,* r_2*,* i_3*,* i_4*);*
$\tilde{H}$*:=add_columns(*$\tilde{H}$*,* c_1*,* j_3*,* c_2*,* j_4*);*
H*:=elimination(*$\tilde{H}$*,* $I(\partial\omega_1 \setminus \partial\omega)$*); /* see Algorithm 4.4*/*
return H*;*
end;

Here i_1, i_2, j_1, j_2 are the positions of two rows and two columns which have to be extracted, i_3, i_4 and j_3, j_4 are the positions of two rows and two columns to which the four rank-1 matrices r_1, r_2, c_1 and c_2 have to be added.

Remark 4.2 *Note that if the coefficients $\alpha(x)$ in ω_1 and ω_2 are equal and triangulations are the same, then it is possible to get $H_1|_{I(\gamma)\times I(\gamma)} = H_2|_{I(\gamma)\times I(\gamma)}$ and $H|_{I(\gamma)\times I(\gamma)} = 2H_1|_{I(\gamma)\times I(\gamma)}$.*

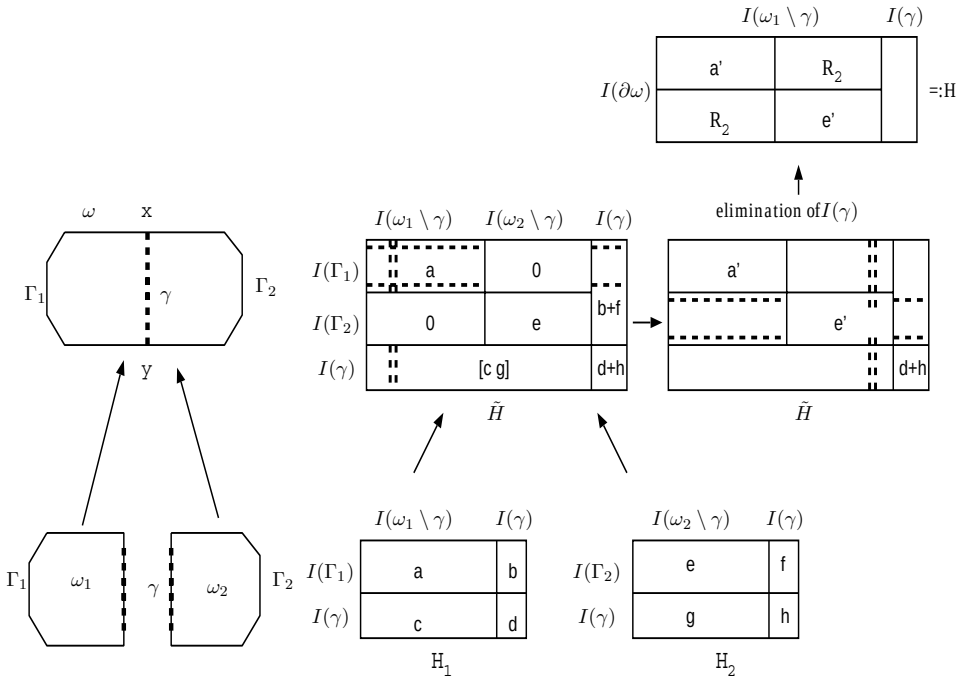


Figure 9: Building $(\Psi^f_\omega)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\omega)}$ from $(\Psi^f_{\omega_1})^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\omega_1)}$ and $(\Psi^f_{\omega_2})^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\omega_2)}$, $\Gamma_i \cup \gamma = \partial\omega_i$, $i = 1, 2$, $\Gamma_1 \cap \Gamma_2 = \{x, y\}$, $I(\partial\omega) = I(\Gamma_1) \cup I(\Gamma_2)\setminus I(\{x, y\})$. The small letters show the appearance of blocks in different matrices. The dotted lines in $\tilde{H}$ correspond to 2 rows and 2 columns.

Building algorithm in case (19):
Let $I := I(\partial\omega)$, $J := J(\omega)$ be two index sets. H_1 and H_2 are defined as in (16) and H as in (17).

Algorithm 4.7 *Build* $H := (\Psi_\omega^f)^{\mathcal{H}}$ *from* $H_1 := (\Psi_{\omega_1}^f)^{\mathcal{H}}$ *and* $H_2 := (\Psi_{\omega_2}^f)^{\mathcal{H}}$

build_*Ψ^f*(*H_1*,* H_2*,* H*)
begin
H':=copy_block_structure(H);
H'':=copy_block_structure(H);
*h2h(H_1, H',...); /*convert H_1 to H' by Algorithm 4.1*/*
h2h(H_2, H'',...);
$\tilde{H} := H'' \oplus H'$;
H:=elimination($\tilde{H}$, $I(\partial\omega_1 \setminus \partial\omega)$); / see Algorithm 4.4*/*
return *H;*
end*;*

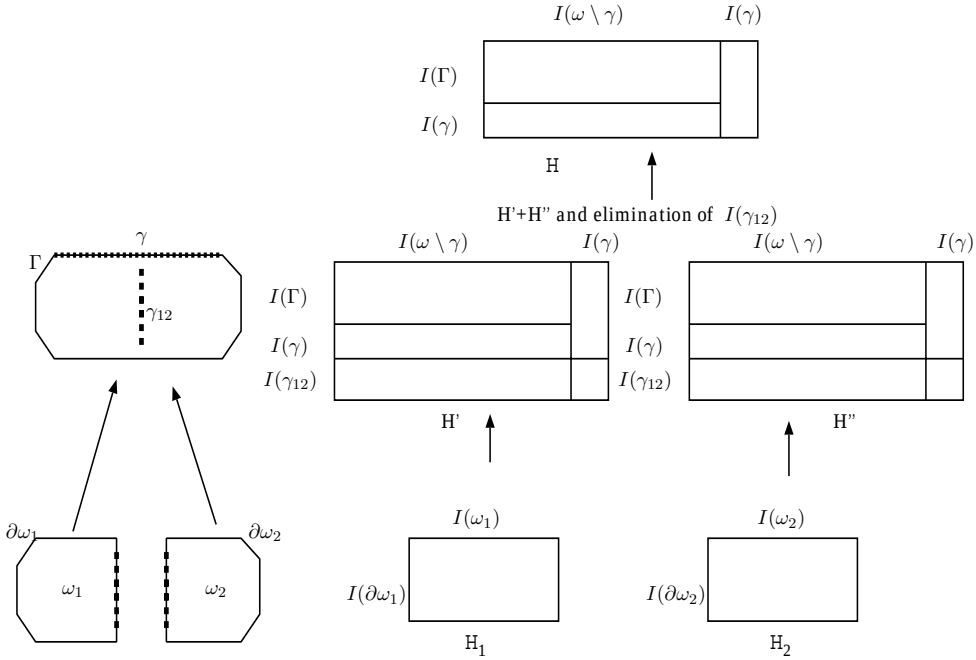


Figure 10: Building $H := (\Psi_\omega^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\omega)}$ from $H_1 := (\Psi_{\omega_1}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\omega_1)}$ and $H_2 := (\Psi_{\omega_2}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\omega_2)}$. $\tilde{H} = H_1|^{I\times J} \oplus H_1|^{I\times J}$, $I = I(\Gamma) \cup I(\gamma)$, $J = J(\omega \setminus \gamma) \cup J(\gamma)$, $H := \tilde{H}_1 \ominus A_{12} \odot A_{22}^{-1} \odot \tilde{H}_2$.

Building $(\Psi^f_\omega)^{\mathcal{H}}$ from $(\Psi^f_{\omega_1})^{\mathcal{H}}$ and $(\Psi^f_{\omega_2})^{\mathcal{H}}$ for a two-scale problem

The index $_h$ indicates the quantities of the fine grid and the index $_H$ of the coarse grid. Denote

$$H_1 := (\Psi^f_{\omega_1})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_{1,h})\times I(\omega_{1,H})}, k), \tag{20}$$

$$H_2 := (\Psi^f_{\omega_2})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_{2,h})\times I(\omega_{2,H})}, k). \tag{21}$$

We want to construct the matrix

$$H := (\Psi^f_{\omega})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_h)\times I(\omega_H)}, k). \tag{22}$$

Note that $\partial\omega_h \cup \gamma_h = \partial\omega_{1,h} \cup \partial\omega_{2,h}$, $I(\partial\omega_{i,h}) = I(\Gamma_{i,h}) \cup I(\gamma_h)$, $\Gamma_{1,h} \cup \Gamma_{2,h} = \partial\omega_h$. We construct the tree $T_{I(\omega_H)}$ so that the further elimination of the unknowns x_i, $i \in I(\gamma_H)$ becomes easier, i.e., we want that $I(\gamma_h) \times I(\gamma_H) \in T_{I(\partial\omega_h)\times I(\omega_H)}$. We choose the following decomposition

$$I(\omega) = I(\omega_{1,H} \setminus \gamma_H) \cup I(\omega_{2,H} \setminus \gamma_H) \cup I(\gamma_H),$$

$$I(\partial\omega_h) = I(\Gamma_{1,h}) \cup I(\Gamma_{2,h}).$$

There are two cases:

$$I(\Gamma_{i,h}) \times I(\omega_{i,H}\backslash\omega_H), I(\gamma_h) \times I(\gamma_H) \in T_{I(\partial\omega_{i,h})\times I(\omega_H)}, i = 1, 2, \tag{23}$$

$$I(\Gamma_{i,h}) \times I(\omega_{i,H}\backslash\omega_H) \notin T_{I(\partial\omega_{i,h})\times I(\omega_H)} \text{ or } I(\gamma_h) \times I(\gamma_H) \notin T_{I(\partial\omega_{i,h})\times I(\omega_H)}, i = 1, 2. \tag{24}$$

Algorithms 4.6, 4.7 with small modifications are used for cases (23) and (24) accordingly. The scheme of building $(\Psi^f_\omega)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_h)\times I(\omega_H)}$ for case (23) is shown in Fig. 11.

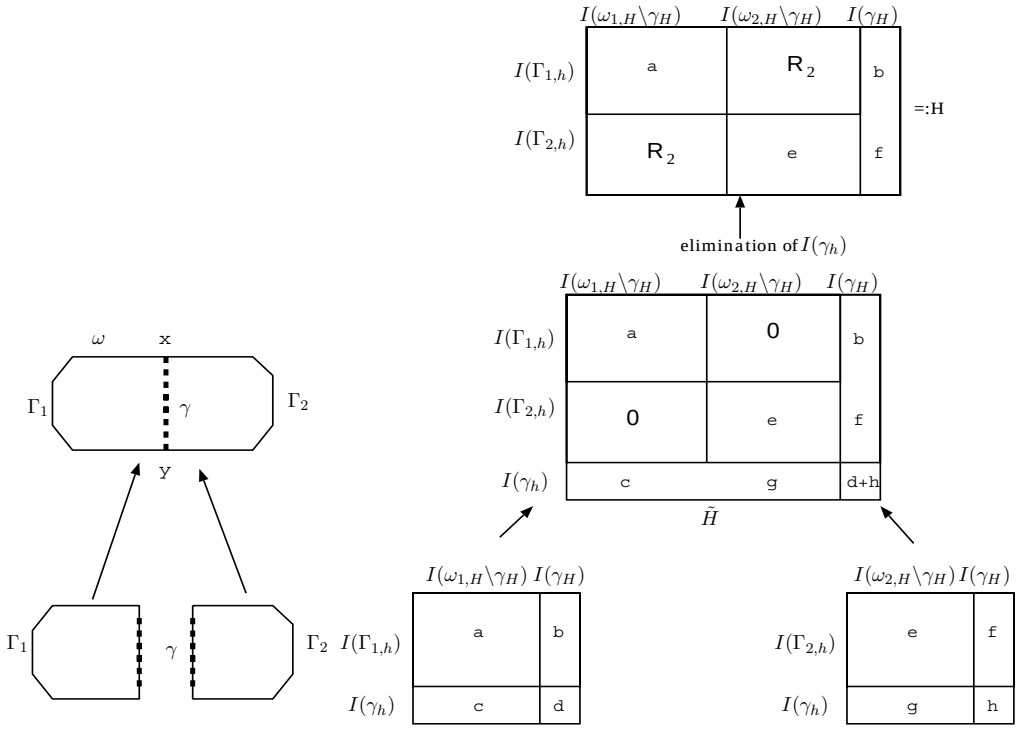


Figure 11: Building $(\Psi_\omega^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_h)\times I(\omega_H)}$ from $(\Psi_{\omega_1}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_{1,h})\times I(\omega_{1,H}}$ and $(\Psi_{\omega_2}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_{2,h})\times I(\omega_{2,H})}$ for two scales H and h. $\Gamma_{1,h}\cap\Gamma_{2,h} = \{x, y\}$, $I(\partial\omega_h) = I(\Gamma_{1,h})\cup I(\Gamma_{2,h})\backslash I(\{x, y\})$, $I(\omega_H) = I(\omega_{1,H}\backslash\gamma_H)\cap I(\omega_{2,H}\backslash\gamma_H)\cap I(\gamma_H)$. The dotted lines in $\tilde{H}$ indicate 2 rows and 2 columns.

4.6 CG method

Often, the exact solution is unknown and to estimate the solution obtained by the HDD method one may use the procedure `solve_by_cg_method(..)` (`myextlib.c`). This procedure builds the stiffness matrix sp (stored in a sparse matrix format) for the whole domain Ω, converts it to the $\mathcal{H}$-matrix s, performs the hierarchical Cholesky decomposition of s by `choleskydecomposition_supermatrix(..)` and then calls the procedure `solve_conjgrad_supermatrix(..)`. The last procedure produces the CG solution ('exact' solution) which is compared with the solution obtained by the HDD method.

4.7 Test procedures and output procedures

To test different parts of the HDD methods one can use procedures from `test.c`. There are procedures which test $\mathcal{H}$-matrix conversion, permutation of rows (columns) in a rank-k and dense matrices, removing and inserting of rows (columns) etc.
There are following output procedures (see files `mylib.c`, `myextlib.c`):

procedure	description
print_tlist()	print out a list of triangles
print_vl()	print out a list of vertices
print_grid2()	print out a grid
print_matrix()	print out a full matrix
print_Rkmatrix()	print out a low-rank matrix
mywrite_supermatrix()	print out an $\mathcal{H}$-matrix
print_svd() print_cond_number()	print out the spectrum of A and cond(A)
print_solution	print out the solution

Table 8: Output procedures.

4.8 3D case

The data structures in HDD package are suitable for the 3D case. For further 3D implementation one should rewrite the following procedures: division procedure (`divide.c`) which produces $T_{\mathcal{T}_h}$, mesh refinement procedures (see `laplace.c`) and start to use the standard admissibility criteria besides the weak admissibility criteria. The changes in the third point yield the differences in building Ψ_ω^g, Ψ_ω^f, Φ_ω^g and Φ_ω^f.

4.9 Parallel HDD method

To implement the parallel HDD method one should: build the domain decomposition tree $T_{\mathcal{T}_h}$ in parallel (see multilevel graph partitioning in [10]) and start to use the parallel $\mathcal{H}$-matrix library. Note, that the hierarchical base of the HDD method is very suitable for the parallel implementation (see Parallel Computing Section in [3]). There is possibility to apply large parts of the sequential code on each processor.

5 Examples

In Table 9 we give the list of different variations of the HDD method. To recompile and to start these programs one should type

```
$ make example_XXX
$ ./example_XXX
```

example_functional	Computes the solution on all internal boundaries γ_ω,$diam(\omega) \geq H$, and the mean values of the solution inside all domains ω with $diam(\omega) < H$.
example_trunc	Algorithm "Root to Leaves" works only for domains $diam(\omega) \geq H$.
example_homogen	For problems with the homogeneous right-hand side. The mapping Ψ_ω^f and Φ_ω^f are not computed.
example_2scales	For the right-hand side from $V_H \subset V_h$.

Table 9: Variations of the HDD method.

6 Files and Their Contents

file (its header file)	description
main.c (.h)	Main file
laplace.c (.h)	Initialization and start procedures
myextlib.c (.h)	Auxiliary procedures (e.g., output)
mylib.c (.h)	Auxiliary procedures
vertex.c (.h)	All procedures for vertices
telement.c (.h)	All procedures for finite elements
divide.c (.h)	Procedures for the hierarchical division of Ω
matrix_arithmetic.c (.h)	Exact matrix arithmetic
aprox_arithmetic.c (.h)	Approximate matrix arithmetic
matrix_operations.c (.h)	Operations with columns and rows of an $\mathcal{H}$-matrix
apr_arithm_fast.c (.h)	Approximate matrix arithmetic
h2h.c (.h)	Procedures for the $\mathcal{H}$-matrix conversion
mycluster.c (.h)	Modified procedures from HLIB/cluster.c
twoscale.c (.h)	Procedures for sparse and prolongation matrices
test.c (.h)	Different test procedures
makefile	Make file
ver_tri*.txt	Files with the lists of triangles and vertices
edges*.txt	Files with the list of edges

Table 10: The list of files in the HDD package.

References

[1] W.Hackbusch, *Elliptic Differential Equations, Springer Series in Computational Mathematics 18, 1992.*

[2] S.Börm, L.Grasedyck, W.Hackbusch: *Hierarchical Matrices*, Lecture Note 21/2003. Max-Planck-Institute for Mathematics, www.mis.mpg.de.

[3] A.Litvinenko: *Application of $\mathcal{H}$-matrices for solving multiscale problems.* PhD thesis, University Leipzig, Germany, 2006.

[4] Software and description of the algorithms for triangulation in 2D case:
http://www.-2.cs.cmu.edu/∼quake/triangle.html

[5] Web resource about HLIB:
http://www.hlib.org

[6] Linear Algebra Package (LAPACK):
http://www.netlib.crg/lapack/

[7] Basic Linear Algebra Subroutines (BLAS):
http://www.netlib.crg/blas/

[8] W.Hackbusch: *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part 1: Introduction to $\mathcal{H}$-matrices.* Computing, 62: 89-108, 1999.

[9] M.Bebendorf and W.Hackbusch: *Existence of $\mathcal{H}$-Matrix approximants to the inverse FE-matrix of elliptic operators with L^∞ - coefficients.* Numerische Mathematik, 95:1-28, 2003.

[10] G. Karypis: Software and theory for 2D and 3D graph partitioning:
http://glaros.dtc.umn.edu/gkhome/views/metis