A Domain–Oriented Multilevel Algorithm — Implementation and Parallelization

Michael Griebel and Tilman Neunhoeffer [1]

Abstract. We present a multilevel algorithm which is well suited for parallelization. Splitting the overall domain into subdomains and separators, we can treat the subdomains independently such that parallelization is straightforward. Regarding communication, the number of setup steps is only dependent on the number of processors but not on the number of levels like for a parallelized multigrid method. This paper is an extension of [6], [8]. Here, we focus especially on an efficient implementation of the parallel algorithm.

Key words. partial differential equations, multilevel methods, semidefinite system, iterative methods, domain decomposition, parallelization

AMS(MOS) subject classifications. 65F10, 65N22, 65N55, 65Y05, 65Y10

1 Introduction

In this report, we consider a domain-oriented multilevel method. It is based on the generating system approach (see [5], [6], [7]). In contrast to standard multilevel algorithms, such as the multigrid method or BPX–type preconditioners where an approximation of the solution of an elliptic PDE is computed on the finest grid and the basic iteration is accelerated by the use of coarse grid functions, we use a generating system which contains the nodal basis functions of *all* levels under consideration. Now, the Galerkin approach leads to a semidefinite system with unknowns on all levels and the associated matrix has a generalized condition number (i.e. the quotient of the largest and smallest *non-vanishing* eigenvalue) of the order $O(1)$. Standard iterative methods on the semidefinite system (Gauss-Seidel, Jacobi preconditioned conjugate gradients) turn out to be equivalent to modern multilevel methods (multigrid, BPX), if we use a levelwise ordering of the unknowns.

But the semidefinite system allows us to abandon the level-oriented view without a significant loss in the convergence properties [6]. For example, we can group together all unknowns which belong to the same part of the overall domain. We use a recursive substructuring process, dividing each (sub)domain into two smaller subdomains and a separator along a middle line [4]. This results in a method where the unknowns are not coupled, if they are associated to different subdomains or separators on the *same* level of the substructuring process. We only have a coupling with the unknowns related to adjacent separators. Thus, we get a simple domain decomposition technique which exhibits multigrid-like convergence properties.

The parallelization of the domain-oriented multilevel algorithm is straightforward. In contrast to the parallelization of a multigrid method where communication has to take place on *all* levels to maintain good convergence rates, our approach needs substantially less communication steps and therefore less setup time, due to its domain decomposition qualities. In this sense, our new method is superior to other parallel multigrid and multilevel methods, especially for parallel systems with high setup times, such as networks of workstations. Because of

[1] Institut für Informatik, Technische Universität München, D-80290 München,
e-mail griebel@informatik.tu-muenchen.de, neunhoef@informatik.tu-muenchen.de

the recursive substructuring approach, our algorithm is well suited for networks which have a configuration as a binary tree. Here, the subdomains on the finest level of the substructuring process are treated in the leaves of the tree, whereas the separators are treated in the nodes of the tree. Then, data exchange can take place in parallel on each level of the tree (see also [8]).

But all the advantages concerning communication time would be useless if the implementation of our new algorithm would cost much more operations than that of a standard multigrid algorithm. In our previous implementations of domain-oriented multilevel algorithms, nine or at least four times as many operations were needed. Now, we present a new implementation of the domain-oriented multilevel algorithm which cuts down this factor substantially. In the subdomains, it is as efficient as a standard multigrid technique and on the separators, we need about twice as many operations. Altogether, we now obtain a factor between 1 and 2, depending on the number of subdomains. So, for realistic examples with a large number of unknowns but a small number of available processors, our algorithm is superior to parallelized multigrid methods, because of the reduced number of setup steps.

We present the times needed for computation, communication and setup and parallel efficiencies of our algorithm on a tree-structured network with 16 workstations and a clustered network with 64 workstations.

2 The Generating System

We consider a two-dimensional, second order partial differential equation with a linear, symmetric and elliptic operator $Lu = f$ in $\Omega := (0,1)^2$ with Dirichlet boundary conditions. The associated weak formulation is:

$$\text{Find } u \in V \text{ such that } a(u,v) = f(v) \quad \forall v \in V.$$

For the discretization on some grid Ω_k with uniform mesh width $h_k = 2^{-k}$, usually, a nodal basis $B_k = \{\phi_{i,k}, i = 1,..,n_k\}$, $n_k = (2^k - 1)^2$ is used. Let $\mathcal{N}_k$ be the set of inner grid points of Ω_k. Then, the bilinear basis functions $\phi_{i,k}$ are defined by $\phi_{i,k}(x_j) = \delta_{ij}, x_j \in \mathcal{N}_k, i,j = 1,..,n_k$, and they span the corresponding space $V_k := \operatorname{span}\{\phi_{i,k}, i = 1,..,n_k\}$. Here, n_k denotes the number of interior grid points and thus the dimension of V_k. Any function $v \in V_k$ can be denoted by

$$v = \sum_{i=1}^{n_k} v_{i,k} \cdot \phi_{i,k}$$

with corresponding coefficient vector $v_k^B = (v_{i,k})_{i=1,..,n_k}$ of nodal values, that is $v(x_i) = v_{i,k}$. Now, the Galerkin approach leads to the linear system

$$L_k^B u_k^B = f_k^B \tag{1}$$

with the vector u_k^B of unknowns.

It is well known that the condition number of L_k^B behaves like $O(h_k^{-2})$. So, with an increasing number of levels and unknowns, an iterative method for the solution of (1) needs more iteration steps. This problem can be overcome by multilevel methods.

Here, a sequence $\Omega_1 \subset \Omega_2 \subset ... \subset \Omega_k$ of grids, with an associated sequence $B_1, B_2, ..., B_k$ of nodal bases and corresponding spaces $V_1 \subset V_2 \subset ... \subset V_k$ with dimensions $n_1, n_2, ..., n_k$, is employed. Inspired by that, we now will directly use the generating system

$$E_k := B_1 \cup B_2 \cup \ldots \cup B_k = \bigcup_{l=1}^{k} B_l$$

for the representation of functions in V_k and for the discretization process. Compare also [6], [7]. Since E_k is only a generating system and not a basis for V_k, the representation of any function $v \in V_k$ by

$$v = \sum_{l=1}^{k} \sum_{i=1}^{n_l} v_{i,l} \cdot \phi_{i,l}$$

with the enlarged vector $v_k^E = (v_1^{B^T}, v_2^{B^T}, \ldots, v_k^{B^T})^T$ is no more unique.
For the 1D case, Figure 1 shows the functions contained in E_3 and an example of a multilevel representation of a function $u_3 \in V_3$.

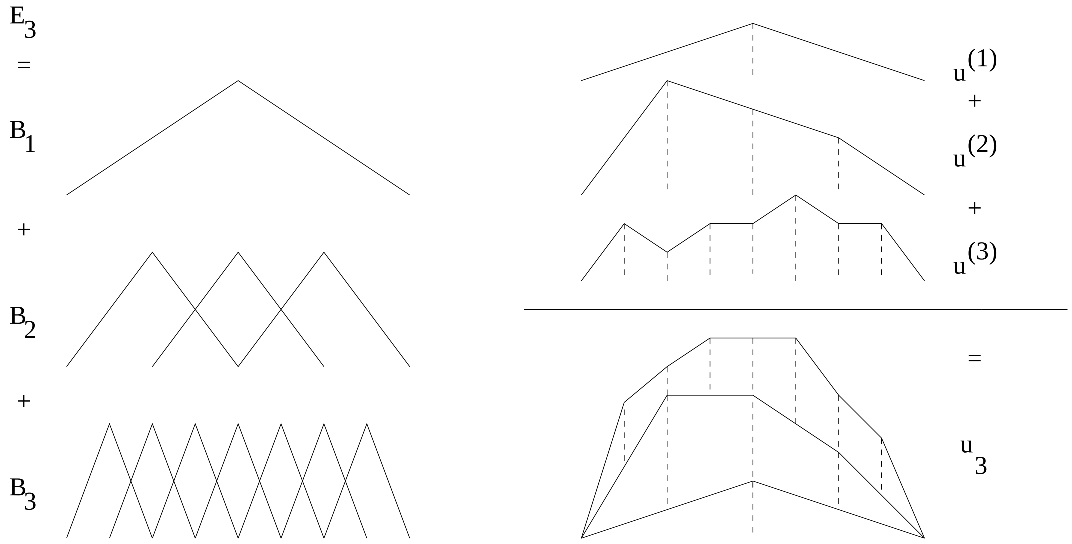

Fig. 1. *Generating system E_3 (left) and multilevel representation of u_3 by E_3 (right) in 1D.*

If we use the generating system E_k directly in the Galerkin discretization process, we obtain the enlarged linear system

$$L_k^E u_k^E = f_k^E \tag{2}$$

where

$$(L_k^E)_{i_1,i_2,l_1,l_2} = a(\phi_{i_1,l_1}, \phi_{i_2,l_2}) \quad \text{and} \quad (f_k^E)_{i_2,l_2} = f(\phi_{i_2,l_2})$$

for $i_1 = 1,..,n_{l_1}$, $i_2 = 1,..,n_{l_2}$ and $l_1, l_2 = 1,..,k$. This linear system is of size $n_k^E = \sum_{l=1}^{k} n_l$, which is in 1D about 2 times, in 2D about 4/3 times and in 3D about 8/7 times larger than n_k, i.e. the size of (1). Note that the matrix L_k^E is only positive semidefinite, because we used the generating system instead of a basis.
Assuming a level-oriented ordering of the unknowns, we obtain the following structure for L_k^E (here for the simple example of $k = 3$):

$$L_k^E = \begin{pmatrix} R_3^1 L_3^B P_1^3 & R_3^1 L_3^B P_2^3 & R_3^1 L_3^B \\ R_3^2 L_3^B P_1^3 & R_3^2 L_3^B P_2^3 & R_3^2 L_3^B \\ L_3^B P_1^3 & L_3^B P_2^3 & L_3^B \end{pmatrix} = \begin{pmatrix} R_3^1 \\ R_3^2 \\ I_3 \end{pmatrix} \cdot L_3^B \cdot \begin{pmatrix} P_1^3 & P_2^3 & I_3 \end{pmatrix} \tag{3}$$

where $R_i^j = P_j^{i^T}$ and P_j^i denotes the interpolation/prolongation from V_j to V_i, $j < i$, and I_i denotes the identity in V_i.
Note that $P_j^i = \prod_{q=1}^{i-j} P_{i-q}^{i-q+1}$. Thus, we see that, with help of the matrix

$$S_k := \begin{pmatrix} P_1^k & P_2^k & . & . & . & P_{k-1}^k & I_k \end{pmatrix},$$

our enlarged system (2) can be written as

$$S_k^T L_k^B S_k u_k^E = S_k^T f_k^B. \tag{4}$$

The discrete Galerkin operators $L_l^B = R_k^l L_k^B P_l^k, l = 1, .., k$, i.e. the stiffness matrices of each level of discretization, are contained as diagonal blocks. The couplings between different levels are contained in the outer diagonal blocks.
Note that our enlarged system is consistent, i.e. $\text{rank}(L_k^E) = \text{rank}(L_k^E, f_k^E)$, and therefore solvable. However, there exists not only one unique solution but many different solutions due to the semidefiniteness of L_k^E. But, since the unique solution u_k^B of (1) can be gained from *any* solution u_k^E of (2) by $u_k^B = S_k u_k^E$, the idea is now to apply S_k to some solution u_k^E of (2) produced by any convergent iterative method.

3 The Domain–Oriented Algorithm

In the previous example (3), we employed a level-wise ordering of the unknowns u_k^E that resulted in a level-block partitioning of the matrix L_k^E which was associated to the splitting $V_k = \sum_{l=1}^k V_l$. It can be easily seen that traditional iterative methods for (2) are equivalent to modern multilevel methods for (1), c.f. [6], [7]. For instance, the simple Jacobi-preconditioner for (2) resembles just the BPX-preconditioner [3] for (1). So, if we define the generalized condition number κ of a positive semidefinite matrix to be the quotient of the largest and non-vanishing smallest eigenvalue, we obtain directly

$$\kappa((D_k^E)^{-1} \cdot L_k^E) = \kappa(BPX_k \cdot L_k^B) = O(1),$$

compare [11], [12], [13], [14]. Consequently, the Jacobi-preconditioned CG-method for (2) converges to some solution within a number of iterations that is independent of k.
Furthermore, we can consider Gauss-Seidel-type methods for (2). They are equivalent to multigrid methods with Gauss-Seidel smoother, c.f. [6], [7]. For example, the symmetric Gauss-Seidel-iteration on the semidefinite system corresponds to the multigrid V-cycle with one pre- and one post-smoothing step. Here, an outer iteration switches from level to level and an inner iteration operates on the specific grids.
The convergence rate ρ of the Gauss-Seidel iteration on (2) can be estimated by

$$\rho = 1 - O(1) < 1.$$

Interestingly, this holds for *all* possible Gauss-Seidel traversal orderings, c.f. [7], [13]. Consequently, we obtain a k-independent convergence rate not only for the Gauss-Seidel method for (2) with some level-wise traversal ordering that corresponds to a multigrid method, but for *any other* traversal ordering as well.

Therefore, we can also use a domain–oriented approach. Here, we may allow for an arbitrary domain decomposition of Ω into non-overlapping subdomains. Then, we group together all the unknowns of the semidefinite system that are associated to basis functions whose center points are situated in the respective subdomain. We switch from subdomain to subdomain in an outer iteration and we treat all the unknowns which may belong to different levels but which are related to the same subdomain in an inner iteration. Within the subdomains and separators, we use a level-oriented ordering of the unknowns.

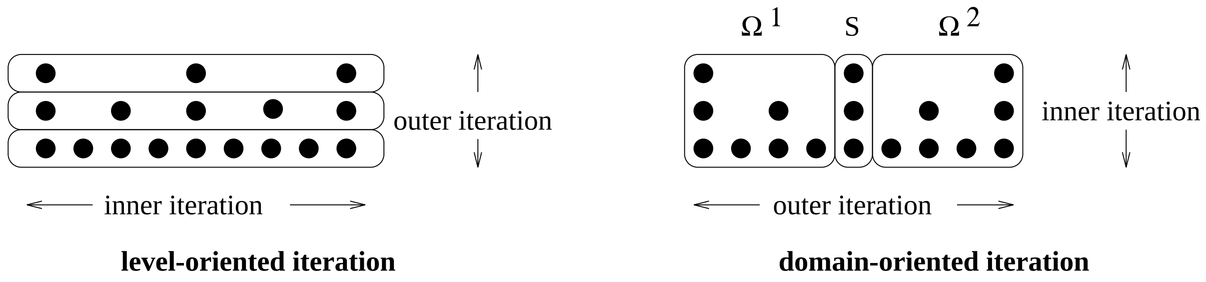

Fig. 2. *Different traversal orderings in 1D.*

A decomposition into subdomains Ω^1 and Ω^2 and a separator S along a middle line, such that the functions of the generating system E_k related to Ω^1 and Ω^2 have disjoint support, is especially well suited for parallelization. Thus, the unknowns belonging to Ω^1 are independent of the unknowns of Ω^2 and the two subdomains can be treated in parallel. A simple 1D example is shown in Figure 2.

Using a nested dissection strategy [4], each of the subdomains $\Omega^{q-1,i}, i = 1, \dots, 2^{q-1}$ itself can be divided into a separator $S^{q-1,i}$ and two smaller subdomains $\Omega^{q,2i-1}$ and $\Omega^{q,2i}$ in step q of the recursive substructuring process. In 2D, this can either be done using parallel lines or lines with alternating directions (c.f. Figure 3). Thus, we obtain a simple domain decomposition method with a stripe- or box-wise decomposition of Ω which exhibits multigrid-like convergence properties.

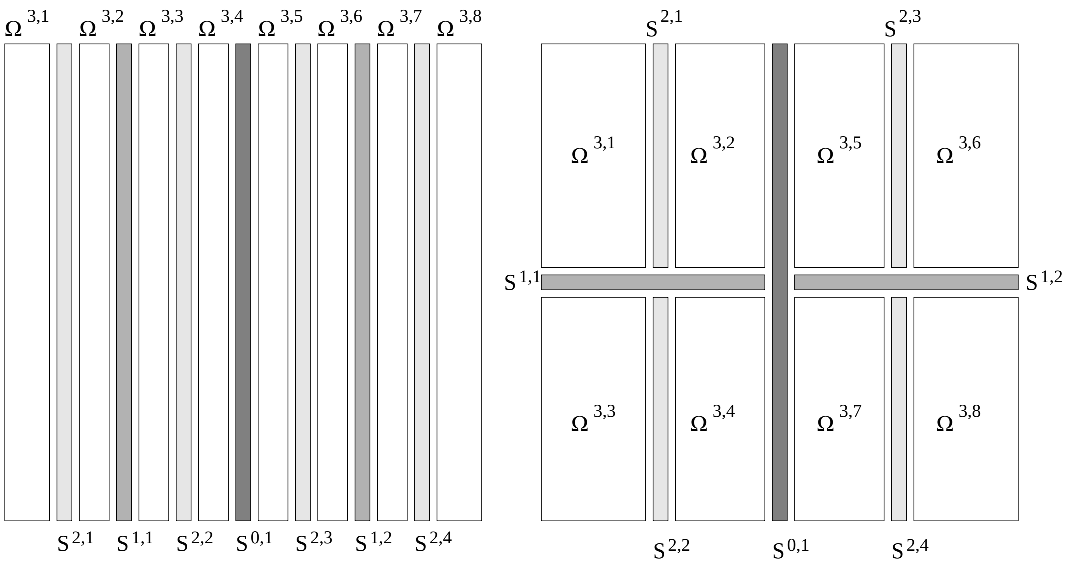

Fig. 3. *Stripe- and box-wise nested dissection decompositions of* Ω.

Note that it is not necessary to assemble the matrix L_k^E and the right-hand side f_k^E explicitly. Similar to (4), it is possible to use prolongation- and restriction-operators and the fine grid discretizations L_k, f_k to express L_k^E and f_k^E in a certain product form. Furthermore, by storing and updating specific parts of the current residual, it is possible to implement the domain-oriented Gauss-Seidel method to need only $O(n_k^E) = O(n_k)$ operations per iteration step. However, an efficient implementation is quite complicated. It will be described in detail in section 5.
Note that the convergence rates for the domain-oriented method are slightly worse than for levelwise orderings, although the rates are independent of the number of levels k.

4 Parallelization

We will see that the computational effort for both methods, standard multigrid and the domain-oriented approach, is in the same range. Thus, we can pose the question whether we need the domain-oriented method at all. The answer gets clear if we consider the parallelization properties of the two methods on most presently available MIMD computers and especially on networks of workstations. There, the time necessary to set up communication is often relatively large. Thus, it is more advantageous to exchange a larger amount of data collectively in one step than to exchange only fractions of the data in many different steps. Otherwise it can happen that the overall execution time is mainly dominated by the setup time.
In this aspect, the level- and the domain-oriented methods are different. For the parallel version of the level-oriented method, we need communication on each level. Splitting the overall domain into two subdomains, the number of communication setup steps is of the order $O(k)$ and the number of data to be exchanged on level l is of the order $O(n_l^{(d-1)/d})$, $l = 1, \ldots, k$. Here, n_l denotes the number of unknowns on level l, k denotes the number of levels and d denotes the dimension of the domain Ω. For details on the parallelization of multigrid methods, see [1], [2], [9], [10].
On the other hand, for the domain-oriented method with *two* subdomains, we can assign each of the subdomains and the separator to one process. Then, within a symmetric Gauss-Seidel iteration step we only need two communication steps between the subdomains and the separator, independent on the number of levels: one in the first and one in the second halfstep. The number of data to be exchanged in each step is of the order $O(\sum_{l=1}^{k} n_l^{(d-1)/d})$. The communication steps needed for the level- and domain-oriented ordering are illustrated in Figure 4.

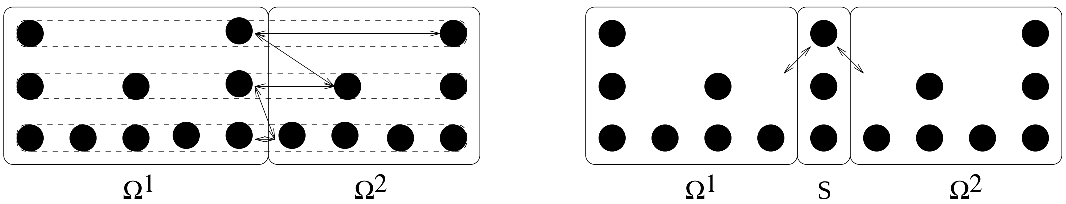

Fig. 4. *Communication ($\longleftrightarrow$) for the level- and domain-oriented ordering with two subdomains in 1D*

If we use more than two subdomains, resulting from a nested dissection process, we obtain a structure of processes like a binary tree. Here, the subdomains are treated in the leaves of the tree and the separators are treated in the nodes. For example the unknowns related to the line in the middle of the overall domain (the separator S) are treated in the root of the tree, c.f. Figure 5, left.

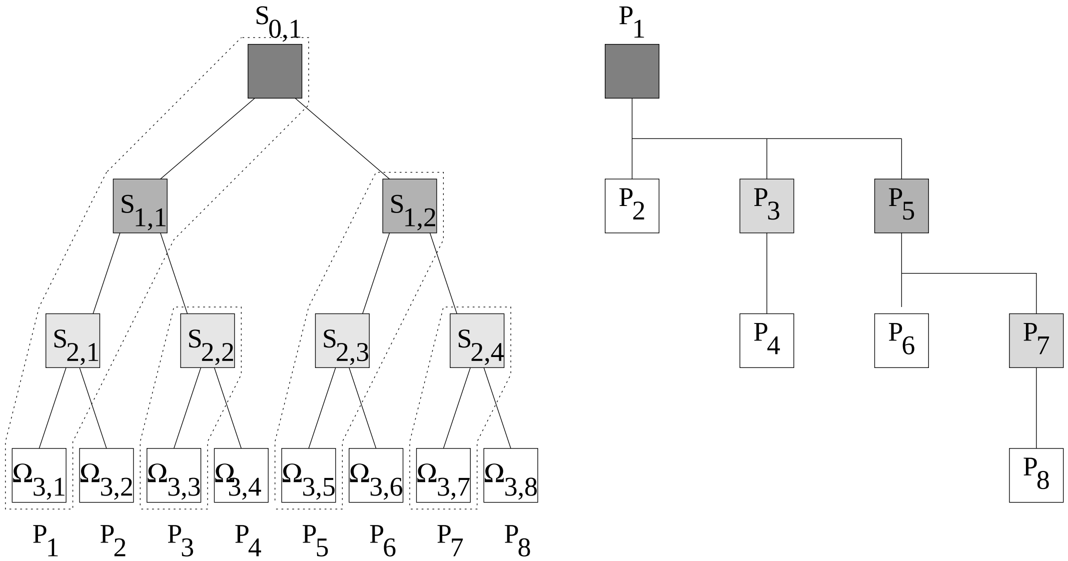

Fig. 5. *Process tree (left) and network configuration (right), example with* $|P| = 8$.

Thus, processes on one level of the tree can operate in parallel whereas processes on different levels must work sequentially. This is the reason why we can map several processes of different levels to one processor, namely each process and its left sons (compare Figure 5, right), such that the separator $S^{q,i}$ is treated by processor P_m with $m = (i-1)2^{p-q} + 1$ and $p = \log_2 |P|$. Then, the number $|P|$ of processors used is only one half of the number of processes. The number of communication steps is of the order $O(p)$ and communication between processes on adjacent levels can be executed in parallel.

Of course, level- and domain oriented multilevel methods show the same asymptotic behaviour as the number of processors $|P|$ tends to the number of unknowns n_k. In both cases, the number of setup steps is of the order $O(\log n_k)$. But for small numbers $|P|$, the domain oriented approach needs substantially less communication steps.

A short analysis of the data transport for the 2D-case shows, that the box-wise decomposition is superior to the stripe-wise decomposition. For the box-wise method, the amount of data which must be communicated sequentially is of the order $O(\sqrt{n_k})$ whereas, for the stripe-wise method, it is of the order $O(\sqrt{n_k}\,\log_2 |P|)$ (see [7]). Also, the time spent for the computation of the unknowns associated to the separators, where some of the processors are idle, is less for the box-wise decomposition.

5 Implementational Aspects

Now we want to discuss some details of an efficient implementation of the symmetric Gauss-Seidel iteration for the semidefinite system

$$L_k^E u_k^E = f_k^E \tag{5}$$

with an ordering of the unknowns that is adapted to the boxwise nested dissection decomposition of Ω as shown in Figure 3 (right).

First, we specify the ordering of the unknowns. Then, we describe a decomposition of the residuals, needed for the computation of the new iterates in the Gauss-Seidel process. In a third step, we demonstrate how those different parts of the residuals can be computed in the first and the second halfstep. At last, we consider the data which must be exchanged in the communication steps. We confess that the subsections 5.2 and 5.3 contain very technical information on our algorithm and implementation. The reader, who is only interested in the main principles of our method, might skip them to some extent.

5.1 The Overall Algorithm

First, we need some further notation. The set $\mathcal{N}_l$, $l = 1, \ldots, k$, of grid points on level l can be divided into subgroups with respect to the subdomains and separators

$$\begin{aligned} \mathcal{N}_l^{p,i} &:= \mathcal{N}_l \cap \Omega^{p,i}, && i = 1, \ldots, 2^p, \\ \mathcal{N}_l^{q,i} &:= \mathcal{N}_l \cap S^{q,i}, && i = 1, \ldots, 2^q, \quad q = 1, \ldots, p-1 \end{aligned}$$

where the number of processors is assumed to be $|P| = 2^p$. By

$$\mathcal{N}_l^q := \bigcup_{i=1}^{2^q} \mathcal{N}_l^{q,i}, \qquad q = 1, \ldots, p,$$

we denote the set of all points on the same grid and the same level of the process tree. $\phi_{x,l}$ is the nodal basis function on level l, associated to grid point $x \in \mathcal{N}_l$, $u_{x,l}$ is the related unknown, $f_{x,l} := f(\phi_{x,l})$ is the right hand side and $a_{x,l;y,m} := a(\phi_{x,l}, \phi_{y,m})$, $x \in \mathcal{N}_l$, $y \in \mathcal{N}_m$, $l, m = 1, \ldots, k$. Then, the equation of the semidefinite system (5) related to the grid point x on level l can be written as

$$\sum_{r=1}^{p} \sum_{m=l_r}^{k} \sum_{y \in \mathcal{N}_m^r} a_{y,m;x,l}\, u_{y,m} = f_{x,l}, \qquad x \in \mathcal{N}_l^q, \quad l = l_q, \ldots, k, \quad q = 1, \ldots, p, \tag{6}$$

where $l_r := \min\{m \mid 1 \leq m \leq k, \mathcal{N}_m^{r,j} \neq \emptyset, j = 1, \ldots, 2^r\}$ is the coarsest level of the subdomains $\Omega^{r,j}$ or separators $S^{r,j}$, respectively.

Now, let w denote the actual iterate in the iterative process. Then, in each point relaxation step of the Gauss-Seidel iteration, the new iterate $w_{x,l}$ is computed as

$$w_{x,l} := w_{x,l} + \frac{1}{a_{x,l,;x,l}}\, r_{x,l} \tag{7}$$

with the current residual

$$r_{x,l} := f_{x,l} - \sum_{r=1}^{p} \sum_{m=l_r}^{k} \sum_{y \in \mathcal{N}_m^r} a_{y,m;x,l}\, w_{y,m}. \tag{8}$$

In our symmetric Gauss-Seidel algorithm for the semidefinite system (5), we use the following ordering of the unknowns: In the first halfstep, we begin with the subdomains $\Omega^{p,i}$ and continue with the separators $S^{q,i}$, $q = p-1, \ldots, 1$, that is, we are going from the leaves to the root of the process tree of Figure 5. Within the subdomains and separators, we switch from finer levels to coarser levels. On each level, we apply a Gauss-Seidel iteration, for example a four-colour iteration for the subdomains and a two-colour iteration for the separators.
In the second halfstep we are going vice-versa. Thus, the whole algorithm reads as follows:

```
for q = p,...,1
    for i ∈ {1,...,2^q}
        if q ≠ p
            data transfer from processor P_{(i-1)2^{p-q}+2^{p-q-1}}
            to processor P_{(i-1)2^{p-q}+1}
        for l = k,...,l_q
            for x ∈ N_l^{q,i}
                compute r_{x,l}
                relax w_{x,l}
for q = 1,...,p
    for i ∈ {1,...,2^q}
        for l = l_q,...,k
            for x ∈ N_l^{q,i}
                compute r_{x,l}
                relax w_{x,l}
        if q ≠ p
            data transfer from processor P_{(i-1)2^{p-q}+1}
            to processor P_{(i-1)2^{p-q}+2^{p-q-1}}
```

The main work is now the computation of the residuals $r_{x,l}$. Of course, a naive implementation of (8) would have some severe drawbacks. First of all, the number of elements of the rows of the matrix L_k^E is not constant. Thus, we would have to store more than $O(n_k)$ elements but we only want to store the elements which contain the coupling of unknowns on the same level, that is only $a_{x,l;y,l}$, $x, y \in \mathcal{N}_l$. Altogether, we only want to store 9 elements of L_k^E for each unknown $u_{x,l}$, namely $a_{x,l;x,l}$ and $a_{x,l;y,l}$ for the eight nearest neighbours y of x on the grid Ω_l.
The second drawback of a naive implementation is that the computation of the residuals would need more than $O(n_k)$ operations in each Gauss-Seidel cycle. For conventional multigrid algorithms, we only need $c_{MG} \cdot n_k$ operations for each cycle, where c_{MG} is a constant which is independent of k. Each cycle of the domain-oriented algorithm should also need $c_{DO} \cdot n_k$ operations only, with c_{DO} independent of k. Moreover the constants c_{MG} and c_{DO} should be in the same range. For our previous implementations of domain-oriented multilevel

algorithms, c_{DO} was about 9 times greater than c_{MG}. Although the domain-oriented multilevel method needs less communication steps, it could not compete with elaborated parallel multigrid methods.
At last, a naive implementation would need more than $O(n_k)$ data to be communicated in each step whereas parallel multigrid methods only need an exchange of $O(n_k)$ data.
But all those drawbacks can be overcome by a splitting the residual (8) in partial sums which will be updated successively, using standard prolongation and restriction operators. This will be described in the following subsections. The implementation is mainly based on the fact that a basis function on level $l-1$ is a linear combination of basis functions on level l. To be more precise:

$$\phi_{x,l-1} = \phi_{x,l} + \frac{1}{2}\left(\phi_{N_l,l} + \phi_{E_l,l} + \phi_{S_l,l} + \phi_{W_l,l}\right) + \frac{1}{4}\left(\phi_{NE_l,l} + \phi_{SE_l,l} + \phi_{SW_l,l} + \phi_{NW_l,l}\right), \tag{9}$$

where, for example N_l denotes the neighbouring grid point of x on grid Ω_l in positive x_2-direction (in the north), see also Figure 6.[2]

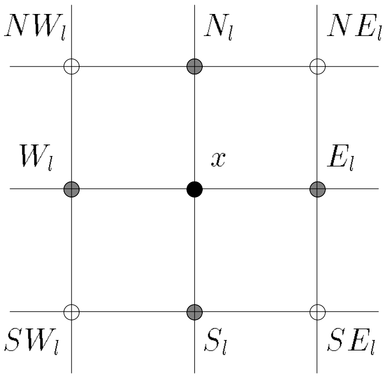

Fig. 6. *Neighbouring grid points on grid Ω_l.*

Another simplification is, that we begin with a startiterate $w^0 = (0_1^T, \ldots, 0_{k-1}^T, (w_k^0)^T)^T$ which is an approximation of u on the finest level, only. After each cycle, we also end up with an iterate where $w_{x,l} = 0$ for $l \neq k$ such that we always obtain an approximation of the unique solution on the finest grid u_k^B. For this, we use a prolongation of the coarse grid values in the second halfstep.

5.2 Decomposition of Residuals

For an efficient implementation of the first and second halfstep of the Gauss-Seidel algorithm described above, we split the residuals into three parts. The first one contains the right hand side $f_{x,l}$ and all values which have been relaxed *before* treating the unknowns related to points

[2]Of course, the position of the neighbouring grid points depends on x, but we do not want to confuse the reader (and ourselves) by using too many indices. This is why we drop the grid point index x and we only write $N_l, W_l, \ldots$ instead of $N_l(x), W_l(x), \ldots$.

in $\mathcal{N}_l^{q,i}$ in the first halfstep. Thus, this part needs not to be changed when point x is revisited on level l in the second halfstep. For $x \in \mathcal{N}_l^{q,i}$, we define

$$d_{x,l} := f_{x,l} - \underbrace{\sum_{r=q+1}^{p} \sum_{m=l_r}^{k} \sum_{y \in \mathcal{N}_l^r} a_{y,m;x,l}\, w_{y,m}}_{\substack{\text{values of lower levels}\\ \text{in the process tree}}} - \underbrace{\sum_{m=l+1}^{k} \sum_{y \in \mathcal{N}_m^{q,i}} a_{y,m;x,l}\, w_{y,m}}_{\substack{\text{values in the same}\\ \text{subdomain/separator}\\ \text{on finer levels}}} - \underbrace{\sum_{r=1}^{q-1} \sum_{y \in \mathcal{N}_k^r} a_{y,k;x,l}\, w_{y,k}^{old}}_{\substack{\text{old values on the finest level}\\ \text{of separators on higher}\\ \text{levels of the process tree}}}. \tag{10}$$

Although, the unknowns related to points $y \in \mathcal{N}_k^r, r < q$ are relaxed after $x \in \mathcal{N}_l^q$ in the first halfstep, we add the values of $w_{y,k}^{old}$ to $d_{x.l}$, that is the values on the finest level of the separators computed in the Gauss-Seidel iteration step before. This is done, because we want to obtain $c_{x,l} = 0$ ($c_{x,l}$ as defined in (11)) in the first halfstep. Thus, we only compute correction terms $\delta w_{y,k}$ for $y \in \mathcal{N}_k^r$, $r < p$, which must be added to $w_{y,k}^{old}$ at the end of each iteration. For simplicity, we write $w_{y,k}$ instead of $\delta w_{y,k}$ in the following, but we must keep in mind, that we have to add the newly computed values of $w_{y,k}$ to $w_{y,k}^{old}$ at the end of each symmetric Gauss-Seidel step.

The second part contains all values which have to be computed *after* the treatment of the unknowns corresponding to $\mathcal{N}_l^{q,i}$ in the first halfstep, that is for $x \in \mathcal{N}_l^{q,i}$:

$$c_{x,l} := \underbrace{\sum_{r=1}^{q-1} \sum_{m=l_r}^{k} \sum_{y \in \mathcal{N}_m^r} a_{y,m;x,l}\, w_{y,m}}_{\substack{\text{values of higher levels}\\ \text{in the process tree}}} + \underbrace{\sum_{m=l_q}^{l-1} \sum_{y \in \mathcal{N}_m^{q,i}} a_{y,m;x,l}\, w_{y,m}}_{\substack{\text{values in the same}\\ \text{subdomain/separator}\\ \text{on coarser levels}}} . \tag{11}$$

Note that $w_{y,k} = \delta w_{y,k} = 0$ for $y \in \mathcal{N}_k^r$, $r < q$, in the first halfstep, because the values on the finest level of the separators are stored in $w_{y,k}^{old}$. Thus, $c_{x,l}$ equals zero in the first halfstep due to the prolongation of the coarse grid values in the second halfstep, and it must only be dealt with in the second halfstep.

The third part contains all values of the *same* level and the *same* subdomain/separator:

$$e_{x,l} := \sum_{y \in \mathcal{N}_l^q} a_{y,l;x,l}\, w_{y,l}. \tag{12}$$

This summation can be reduced to the neighbouring points on level l in the subdomain.
Altogether, we obtain

$$r_{x,l} = d_{x,l} - c_{x,l} - e_{x,l} \tag{13}$$

for each $x \in \mathcal{N}_q^l$ in the relaxation step.

The values of $d_{x,l}$ that are computed in the first halfstep must be stored for each $l = 1, \ldots, k$ and $x \in \mathcal{N}_l$ because they are needed again in the second halfstep, whereas the values of $c_{x,l}$ must only be stored for certain grid points as we shall see later on. In the first halfstep, $c_{x,l}$ is zero for all x and l and, in the second halfstep, most of the coarse grid values involved in $c_{x,l}$ can be prolongated and added to the values of the iterate w on finer levels. The values of $e_{x,l}$ need not to be stored either, because they are only needed for the computation of the actual residuals $r_{x,l}$.

5.3 Efficient Computation of Residuals

5.3.1 The first Halfstep

To go more into details, we now consider the computation of the values of d in the first halfstep.
Because of (9), we can compute the value of d for any $x \in \mathcal{N}_{l-1}^{s,i}$, $i = 1, \dots, 2^s$, $s = 1, \dots, p$, on level $l-1$ by

$$d_{x,l-1} = \bar{d}_{x,l}^{[s]} + \frac{1}{2}\left(\bar{d}_{N_l,l}^{[s]} + \bar{d}_{E_l,l}^{[s]} + \bar{d}_{S_l,l}^{[s]} + \bar{d}_{W_l,l}^{[s]}\right) + \frac{1}{4}\left(\bar{d}_{NE_l,l}^{[s]} + \bar{d}_{SE_l,l}^{[s]} + \bar{d}_{SW_l,l}^{[s]} + \bar{d}_{NW_l,l}^{[s]}\right). \quad (14)$$

Here $\bar{d}_{y,l}^{[s]}$ denotes the coupling of the unknown $w_{y,l}$ with all actual values of w which are involved in $d_{x,l-1}$ as defined in (10), that is for $y \in \mathcal{N}_l^q$, $q \geq s$

$$\bar{d}_{y,l}^{[s]} := f_{y,l} - \sum_{r=s+1}^{p} \sum_{m=l_r}^{k} \sum_{z \in \mathcal{N}_l^r} a_{z,m;y,l}\, w_{z,m} - \sum_{m=l}^{k} \sum_{z \in \mathcal{N}_m^{s,i}} a_{z,m;y,l}\, w_{z,m} - \sum_{r=1}^{s-1} \sum_{z \in \mathcal{N}_k^r} a_{z,k;y,l}\, w_{z,k}^{old}. \quad (15)$$

In contrast to (10) for $d_{y,l}$, the lower limit of the second partial sum is l and not $l+1$ and the inner sum is over $y \in \mathcal{N}_m^{s,i}$ and not over $y \in \mathcal{N}_m^{q,i}$. In the first and the third partial sum, we have the limits $s+1$ and $s-1$ instead of $q+1$ and $q-1$, respectively.
It is now the main task to compute those $\bar{d}_{y,l}^{[s]}$. Of course, the sum $d_{y,l}$ is part of the sum $\bar{d}_{y,l}^{[s]}$ for $y \in \mathcal{N}_l^q$. Therefore, the computation of $\bar{d}_{y,l}^{[s]}$ is based on $d_{y,l}$, but we must add some further terms.
First, we consider the subdomains which are treated in the leaves of the process tree ($q = p = s$ in (14), (15)). Here, (15) can be written as

$$\bar{d}_{y,l}^{[p]} = d_{y,l} - a_{y,l;y,l}\, w_{y,l} - \sum_{z \in \mathcal{NB}_l} a_{z,l;y,l}\, w_{z,l}, \quad (16)$$

where $\mathcal{NB}_l$ is the set of the 8 neighbouring points of y on level l. To compute $d_{x,l-1}$ for $x \in \mathcal{N}_{l-1}^p$, we need the values of $\bar{d}_{y,l}^{[p]}$ for all $y \in \mathcal{N}_p^l$. A short analysis shows that this algorithm is equivalent to the standard multigrid method in the subdomains.

The computation of $\bar{d}_{y,l}^{[\cdot]}$ on the separators, involves not only values of levels $\geq l$ but also of coarser levels. For reason of simplicity, we only consider horizontal lines. For vertical lines we would have to replace W and E by N and S and vice-versa.
First, we need $d_{x,k}$, $x \in \mathcal{N}_k^{s,i}$, $s < p$. Here, we must take into account the old values w^{old} on the finest level at the left and right neighbour and at the point itself. Moreover, for the two points at the vertices of the separator, we also have a coupling with the old values at the points marked with a cross in Figure 7, placed on separators on higher levels of the process tree. Furthermore, we must consider all values $w_{y,l}$ at points $y \in \mathcal{N}_l$, $l = l_{s+1}, \dots, k$ with a vertical distance $h_l = 2^{k-l} h_k$ from the separator $S^{s,i}$. We denote this set by $\mathcal{NL}_l^{s,i}$ (see also Figure 7).

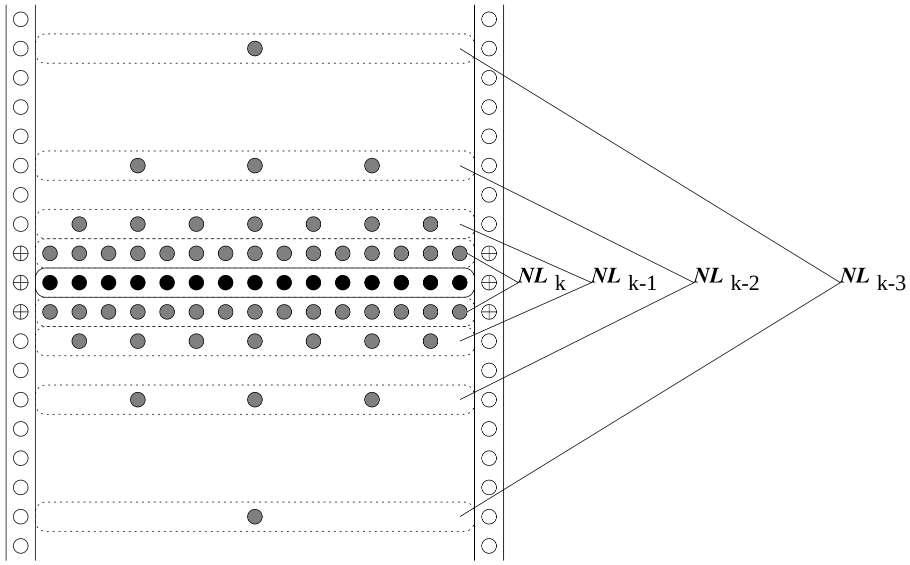

Fig. 7. *Separator and points involved in the computation.*

Using an interpolation algorithm[3], we can compute the values of

$$\hat{w}_{y,l} := \hat{w}(y) := \sum_{l=l_{s+1}}^{k} \sum_{z\in\mathcal{NL}_l^{s,i}} w_{z,l}\,\phi_{z,l}(y) \quad \text{for} \quad y\in\mathcal{NL}_l^{s,i}, \quad l=l_{s+1},\dots,k. \tag{17}$$

Then, the value of $d_{x,k}$ can be computed by

$$d_{x,k} = f_{x,k} - a_{x,k;x,k}\,w^{old}_{x,k} - a_{E_k,k;x,k}\,w^{old}_{E_k,k} - a_{W_k,k;x,k}\,w^{old}_{W_k,k} - \sum_{y\in\mathcal{NL}_k^{s,i}} a_{y,k;x,k}\,\hat{w}_{y,k}. \tag{18}$$

At the right and left vertices of the separator, we still have to add $a_{y,k;x,k}\,w^{old}_{y,k}$ for $y \in \{NW_k, SW_k\}$ or $\{NE_k, SE_k\}$, respectively.
For the computation of $d_{x,l-1}$, $l\le k$, on the separator according to (14), we need the values of $\bar{d}^{[s]}$ on level l at the 8 neighbouring points of x on level l and at x itself. Three of them are situated on the same separator but the other six are points of $\mathcal{NL}_l^{s,i}$. Thus, we need the values of $\bar{d}^{[s]}$ at all points $y\in\mathcal{N}_l^{s,i}$ on the same separator and $y\in\mathcal{NL}_l^{s,i}$ which are associated to the adjacent subdomains. The first ones can be computed analogously to (16) by

$$\bar{d}^{[s]}_{x,l} = d_{x,l} - a_{x,l;x,l}\,w_{x,l} - a_{W_l,l;x,l}\,w_{W_l,l} - a_{E_l,l;x,l}\,w_{E_l,l}. \tag{19}$$

For the computation of the latter, we split $\bar{d}^{[s]}$ for $y\in\mathcal{N}_l^q$, $s<q\le p$, into three parts:

$$\bar{d}^{[s]}_{y,l} = \bar{d}^{[q]}_{y,l} - \hat{d}^{[s]}_{y,l} - \tilde{d}^{[s]}_{y,l} \tag{20}$$

[3] We can compute $\hat{w}_{y,l}$ for $y\in\mathcal{NL}_k^{s,i}$, y in the upper half of $\Omega^{s,i}$ by the following algorithm:

> **for** $l = l_{s+1}+1,\dots,k$
>   **for** $y\in\mathcal{NL}_l^{s,i}$, **upper line, odd points**
> $\hat{w}_{y,l} = w_{y,l} + \frac{1}{4}(\hat{w}_{NW_l,l-1} + \hat{w}_{NE_l,l-1})$,
>   **for** $y\in\mathcal{NL}_l^{s,i}$, **upper line, even points**
> $\hat{w}_{y,l} = w_{y,l} + \frac{1}{2}\hat{w}_{N_l,l-1}$,

where the odd points are the 1^{st}, 3^{rd}, 5^{th},... points of the line and the even points are the 2^{nd}, 4^{th},... points of the line. An analogous algorithm can be used in the lower half of $\Omega^{s,i}$.

where

$$\hat{d}^{[s]}_{y,l} := \sum_{r=s+1}^{q} \sum_{m=l_r}^{l-1} \sum_{z\in\mathcal{N}^r_m} a_{z,m;y,l}\, w_{z,m} \tag{21}$$

contains all values of w on *coarser* levels and

$$\tilde{d}^{[s]}_{y,l} := \sum_{r=s}^{q-1} \sum_{m=l}^{k} \sum_{z\in\mathcal{N}^r_m} a_{z,m;y,l}\, w_{z,m} \tag{22}$$

contains all values of w on the *same* level or finer levels, which have been computed *between* the treatment of level l in the subdomain $\Omega^{p,j}$ or separator $S^{q,j}$ and level l on the separator $S^{s,i}$. If we consider the coupling between different unknowns, the coarse grid part (21) can be written as

$$\hat{d}^{[s]}_{y,l} = \sum_{m=l_{s+1}}^{l-1} \sum_{z\in\mathcal{NL}^{s,i}_m} a_{z,m;y,l}\, w_{z,m}\,. \tag{23}$$

Thus, the value of $\hat{d}^{[s]}_{y,l}$ can be computed during the interpolation process of $\hat{w}$ as described in Footnote 3.
The fine grid part $\tilde{d}^{[s]}_{y,l}$ is computed in several steps. Whenever we treat a separator $S^{r,i}$ on level l, we must add the values of $a_{x,l;y,m}\, w_{x,l}$, $x \in \mathcal{N}^{r,i}_l$ to the value of $\tilde{d}^{[r]}_{y,m}$ on coarser levels $m = l_r, \ldots, l$ for points y where $\tilde{d}^{[s]}_{y,m}$, $s = r, r-1, \ldots$, is needed later on in the first halfstep. Those y are exactly the points of $\mathcal{NL}^{r,i}_m$. The contribution of the values on separator $S^{r,i}$ to the relevant values of $\tilde{d}^{[r]}$ can be evaluated recursively for $m = k, \ldots, l_r$.
Altogether, we must store the values of $d_{x,l}$ and $\tilde{d}^{[s]}_{x,l}$ for each $x \in \mathcal{N}_l, l = 1, \ldots k$. For the interpolation of $\hat{w}$ and the recursive computation of $\tilde{d}$, we need some additional storage related to the points of $\mathcal{NL}^{q,i}_l, q = 1, \ldots, p-1, l = l_q, \ldots, k, i = 1, \ldots, 2^q$. In addition to the operations needed in the subdomains where we compute residuals (15) (this can be compared with (18), (19), (22) for the separators) and apply restriction operators (14), we need the interpolation (17) and the computation of the related residuals in (21) when treating the separators. Thus, we need about twice as many operations per grid point on the separators than in the subdomains.

5.3.2 The second Halfstep

In the second halfstep, we first consider the unknowns related to the separators $S^{q,i}$, $q = 1, \ldots, p-1$. After treating level $l-1$, $l = l_q + 1, \ldots, k$, we use a prolongation of the values of $w_{x,l-1}$, $x \in \mathcal{N}^{q,i}_{l-1}$, which is based on (9). For horizontal lines, we set

$$\begin{aligned}
&w_{x,l-1} = 0, \quad w_{x,l} := w_{x,l} + w_{x,l-1}, \qquad && w_{E_l,l} := w_{E_l,l} + \frac{1}{2} w_{x,l-1}, \quad w_{W_l,l} := w_{W_l,l} + \frac{1}{2} w_{x,l-1},\\
&\tilde{w}_{N_l,l} = \frac{1}{2} w_{x,l-1}, \quad \tilde{w}_{S_l,l} = \frac{1}{2} w_{x,l-1}, && \tilde{w}_{y,l} = \frac{1}{4} w_{x,l-1}, \quad y \in \{NE_l, NW_l, SW_l, SE_l\}\,.
\end{aligned}$$

After smoothing on level k, all values which have been computed for the separator are stored in $w_{x,k}$, $x \in \mathcal{N}^{q,i}_k$ and $\tilde{w}_{y,l}$, $y \in \mathcal{NL}^{q,i}_l$, $l = l_{q+1}, \ldots, k$. So, when treating the values associated to a separator $S^{q,i}$ which divides the subdomain $\Omega^{q,i}$ into two smaller subdomains, we must take into account all $w_{y,k}$ and $\tilde{w}_{y,l}$ related to the four edges of $\Omega^{q,i}$, except the edges which are part of the boundary of the overall domain Ω. Those points are marked with grey and black points in Figure 8.

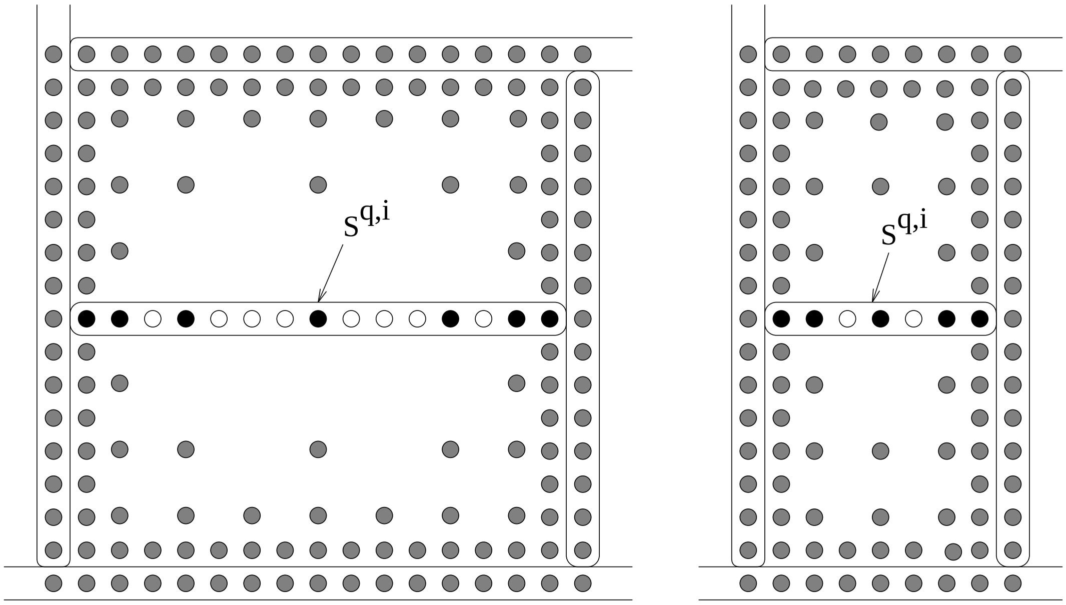

Fig. 8. *Values to be considered for the computation of a separator or a subdomain in the second halfstep.*

By $\overline{\mathcal{NE}}^{q,i}$, we denote the set of points on the four edges and, by $\mathcal{NE}_l^{q,i}$, we denote the set of grid points on level l in subdomain $\Omega^{q,i}$ which have a distance $h_l = 2^{k-l}h_k$ from one of the edges. Then, (11) becomes

$$c_{x,l} = \sum_{y \in \overline{\mathcal{NE}}^{q,i}} a_{y,k;x,l}\, w_{y,k} + \sum_{m=l_q}^{k} \sum_{y \in \mathcal{NE}_m^{q,i}} a_{y,m;x,l}\, \tilde{w}_{y,l} + \sum_{m=l_q+1}^{l} \sum_{y \in \mathcal{NL}_m^{q,i}} a_{y,m;x,l}\, \tilde{w}_{y,l} \tag{24}$$

for $x \in \mathcal{N}_l^{q,i}$, $l = l_q, \ldots, k$, $q = 1, \ldots, p-1$. Note that $\tilde{w}$ in the second term relates to the interpolated coarse grid values of the four edges of $\Omega^{q,i}$ whereas $\tilde{w}$ in the third term relates to the interpolated values of coarser levels on the separator $S^{q,i}$ itself.

Once more, $c_{x,l}$ can be split into two parts: $c_{x,l} = \tilde{c}_{x,l} + \hat{c}_{x,l}$. The first one contains all values on finer levels,

$$\tilde{c}_{x,l} := \sum_{y \in \overline{\mathcal{NE}}^{q,i}} a_{y,k;x,l}\, w_{y,k} + \sum_{m=l+1}^{k} \sum_{y \in \mathcal{NE}_m^{q,i}} a_{y,m;x,l}\, \tilde{w}_{y,l}. \tag{25}$$

If we compute $\tilde{c}_{x,l}$ for all $x \in \mathcal{NE}_l^{q,i}$, (they are all needed at last in the smallest subdomains $S^{p,j}$), we can compute them recursively for $l = k \ldots, l_q$ analogously to the values of $\tilde{d}^{[\cdot]}$.

The interpolated values from the four edges and the line itself on level l and on coarser levels are accumulated in $\hat{c}_{x,l}$,

$$\hat{c}_{x,l} := \sum_{m=l_q}^{l} \sum_{y \in \mathcal{NE}_m^{q,i}} a_{y,m;x,l}\, \tilde{w}_{y,l} + \sum_{m=l_q+1}^{l} \sum_{y \in \mathcal{NL}_m^{q,i}} a_{y,m;x,l}\, \tilde{w}_{y,l}. \tag{26}$$

The $\hat{c}$ can be computed by using an interpolation of the involved values analogously to the description in the footnote on page 13.
Adding the values of $c_{x,l}$ and $d_{x,l}$, where the latter have been computed in the first halfstep, and $e_{x,l}$, which contains only $w_{W_l,l}, w_{E_l,l}$ and $w_{x,l}$, we can compute the residual $r_{x,l}$ for the second halfstep. After the computation of the new iterates on the finest level of the separator, we still have to add the computed $w_{x,k} (= \delta w_{x,k})$ to $w_{x,k}^{old}$ to get the new iterate $w_{x,k}$, $x \in \mathcal{N}_k^{q,i}$.
In the subdomains $\Omega^{p,i}$, we use the same prolongation technique as in the separators. Before treating level l, we transform the iterates on level $l-1$ to level l by

$$\begin{aligned} w_{x,l-1} &= 0, & w_{x,l} &:= w_{x,l} + w_{x,l-1}, \\ w_{y,l} &:= w_{y,l} + \frac{1}{2} w_{x,l-1}, & y &\in \{N_l, W_l, S_l, E_l\}, \\ w_{y,l} &:= w_{y,l} + \frac{1}{4} w_{x,l-1}, & y &\in \{NE_l, NW_l, SW_l, SE_l\}, \end{aligned}$$

for all $x \in \mathcal{N}_{l-1}^{p,i}$. Then, we add the interpolated values of the four neighbouring lines

$$w_{x,l} := w_{x,l} + \tilde{w}_{x,l}, \qquad x \in \mathcal{NE}_l^{p,i}. \tag{27}$$

Thus, the residuals on level l can be computed by

$$\begin{aligned} r_{x,l} = d_{x,l} - e_{x,l} - \hat{c}_{x,l} & \quad \text{if} \quad x \in \mathcal{NE}_l^{p,i}, \\ r_{x,l} = d_{x,l} - e_{x,l} & \quad \text{if} \quad x \in \mathcal{N}_l^{p,i} \setminus \mathcal{NE}_l^{p,i}, \end{aligned}$$

where the $d_{x,l}$ have been already computed in the first halfstep and, in $e_{x,l}$, all actual values at the neighbouring points on level l are considered as in (12). Note that $e_{x,l}$ also contains the prolongated coarse grid values of the same subdomain.

5.4 Data Transport

At last, we want to consider the data that must be communicated between different processors. In the *first halfstep*, the values of $w_{y,l}$ and $\bar{d}_{y,l}^{[r]} + \tilde{d}_{y,l}^{[q+1]}$ for $y \in \mathcal{NL}_l^{q,i}$, $l = l_{q+1}, \dots, k$, $(y \in \mathcal{N}_l^r)$, are needed for the computation on the separator $S^{q,i}$. If we assign the subdomain $\Omega^{q+1,2i-1}$ to the same processor P_{i_1} as the separator $S^{q,i}$ and if we assign the subdomain $\Omega^{q+1,2i}$ to processor P_{i_2} ($i_1 = (i-1)2^{p-q} + 1$, $i_2 = (2i-1)2^{p-q-1} + 1$), we must transport one half of the above-mentioned data, namely all data belonging to points $y \in \Omega^{q+1,2i}$, from processor P_{i_2} to P_{i_1}. Moreover, for the computation of the unknowns related to the other three separators adjacent to $\Omega^{q+1,2i}$, we need the values of $w_{y,l}$ and $\bar{d}_{y,l}^{[r]} + \tilde{d}_{y,l}^{[q+1]}$ which are related to the other three edges. This data must also be transferred to P_{i_1}. Altogether, for $y \in \mathcal{NE}_l^{q+1,2i}$, $l = l_{q+1}, \dots, k$, $w_{y,l}$ and $\bar{d}_{y,l}^{[r]} + \tilde{d}_{y,l}^{[q+1]}$ must be transported from P_{i_2} to P_{i_1}. All those points are marked with grey points in Figure 9. Asymptotically, this are $4s_{q+1}$ floating point numbers, where s_{q+1} is the number of points on the finest level on the edges of $\Omega^{q+1,2i}$ ($s_q = 4\sqrt{n_k}\,(0.5)^{(q-1)/2}$ for odd q and $s_q = 4\sqrt{n_k}\,0.75\,(0.5)^{(q-2)/2}$ for even q).[4]

[4]We really do not need all this data. In fact, we can skip the data related to edges of subdomains which are part of the boundary of the overall domain Ω.

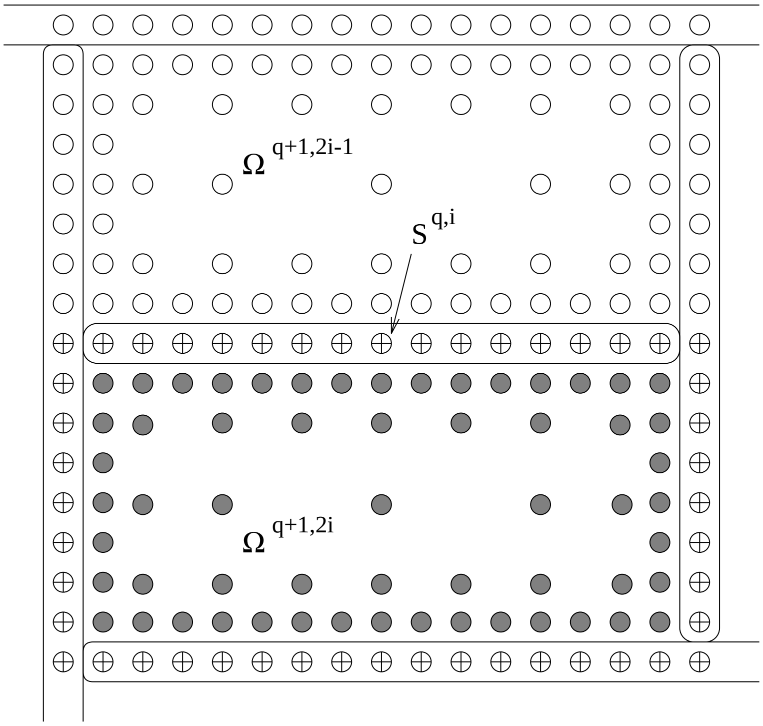

Fig. 9. *Data Transport*

For the computation on the subdomain $\Omega^{q+1,2i}$ with $(1 < q+1 < p)$ or without $(q+1 = p)$ a separator in the *second halfstep*, we need the following values for the four edges: $w_{y,k}$ for $y \in \overline{\mathcal{NE}}^{q+1,2i}$ (points with a cross in Figure 9) and $\tilde{w}_{y,l}$ and $\hat{c}_{x,l}$ for $y \in \mathcal{NE}_l^{q+1,2i}$, $l = l_{q+1}, \ldots, k$, (grey points in Figure 9). Asymptotically $5s_{q+1}$ floating point numbers must be transported from $P_{(i-1)2^{p-q}+1}$ to $P_{(2i-1)2^{p-q-1}+1}$.
Altogether, we obtained a domain oriented multilevel algorithm where the computation and communication costs are competitive to parallelized multigrid methods. The main advantage of our method is the lower number of setup steps needed in the practical case of a limited number of processors.

6 Results

We now report on the results of our numerical experiments using the algorithm described above. We consider the model problem

$$-\Delta u = f \text{ in } \Omega = (0,1)^2$$

with Dirichlet boundary conditions. Note that this simple Poisson problem is a hard test case when it comes to parallelization, since the ratio of arithmetic operations to communication costs is worse than for all other second order elliptic PDEs.
The code is written in C and uses the PVM communication system. For our experiments, we used two different networks of workstations.

6.1 Tree-like Network with 16 Workstations

First, we run our algorithm in parallel on a network of 16 HP 9000/720 workstations. They are equipped with *two* Ethernet cards (Network 1). Thus, it is possible, to build a tree-like configuration as it is shown in Figure 10. Communication can take place in parallel on the Ethernet connections of the different levels of the processor tree. The processors P_1 and P_9 have a physical memory of 80 MB whereas the other 14 processors have a physical memory of 32 MB.

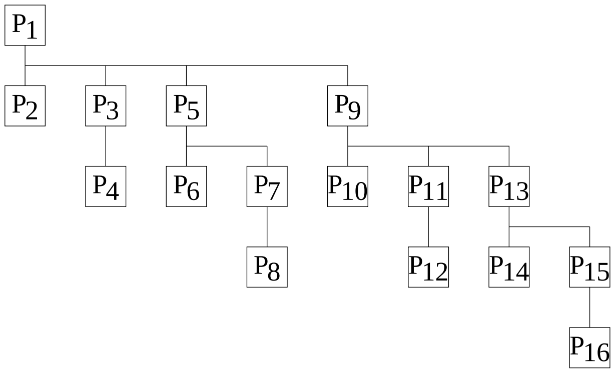

Fig. 10. *Network 1 (16 workstations, tree-like configuration).*

In the following tables, we show the time in seconds, used for computation, communication and setup in one iteration step of the parallelized symmetric Gauss-Seidel relaxation performed on the semidefinite system.
First, we measured the communication setup time S for one symmetric Gauss-Seidel iteration step. The results are shown in Table 1.[5] They only depend on the number $|P|$ of processors but they do not depend on the number k of levels. We obtained $S \approx 0.023 \cdot \log_2(|P|)$.

Table 1: Setup time S for $k = 3, \ldots, 11$, Network 1.

$\|P\|$	1	2	4	8	16
S	0.000	0.022 - 0.024	0.044 - 0.047	0.067 - 0.075	0.090 - 0.097

Secondly, we measured the communication time C, the computation time W and the total time T for one symmetric Gauss-Seidel iteration of the overall algorithm, i.e. $T = S + C + W$. The results are shown in Table 2. Note that the missing values for $k = 11, |P| = 1, 2, 4, 8$ are not given since the computation on these levels showed paging and swapping effects.
Regarding the communication time, it can be seen that C behaves like $O(\log_2 |P|)$ for fixed k. On the other hand, it depends on the size of the largest separator. So, for fixed $|P|$ we must obtain $C = O(2^k)$. A short analysis of our data shows that

$$C \approx \log_2(|P|) \cdot (0.013 + 4 \cdot 10^{-5} \cdot 2^k).$$

But this only holds for low numbers $|P|$. From theory [7], we know that the amount of data which must be communicated sequentially does not depend on $|P|$ for sufficiently large values

[5]The initialization setup time, i.e. the time necessary to distribute the code and the processes on the network and to allocate necessary data at the beginning of the computation, is not contained.

Table 2: Communication time C, computation time W and total time T, Network 1.

	$\|P\|$	1	2	4	8	16
$k=3$	C	0.000	0.013	0.026	0.042	0.059
	W	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
	T	< 0.001	0.036	0.070	0.109	0.149
$k=4$	C	0.000	0.012	0.025	0.044	0.061
	W	0.003	0.002	0.002	0.001	0.001
	T	0.003	0.036	0.072	0.112	0.152
$k=5$	C	0.000	0.012	0.026	0.046	0.064
	W	0.013	0.008	0.005	0.002	0.001
	T	0.013	0.042	0.077	0.116	0.155
$k=6$	C	0.000	0.013	0.027	0.050	0.068
	W	0.062	0.030	0.016	0.012	0.011
	T	0.062	0.066	0.091	0.131	0.170
$k=7$	C	0.000	0.016	0.034	0.057	0.073
	W	0.356	0.160	0.077	0.038	0.035
	T	0.356	0.200	0.157	0.166	0.192
$k=8$	C	0.000	0.022	0.042	0.065	0.104
	W	1.599	0.780	0.376	0.180	0.095
	T	1.599	0.825	0.466	0.321	0.293
$k=9$	C	0.000	0.027	0.067	0.102	0.127
	W	6.478	3.230	1.620	0.812	0.404
	T	6.478	3.291	1.734	0.987	0.627
$k=10$	C	0.000	0.068	0.118	0.173	0.217
	W	26.111	13.064	6.538	3.304	1.696
	T	26.111	13.156	6.703	3.551	2.001
$k=11$	C					0.392
	W					6.657
	T					7.147

of $|P|$ since some sort of geometric series develops. We then would have $C \approx 0.013 \cdot \log_2(|P|) + 4 \cdot 10^{-5} \cdot 2^k \cdot 6.75$ but this can only been seen for higher processors numbers.
For the computation time, we have $W = c_1 \cdot 2^{2k}/|P| + c_2 \cdot 2^{k+1}$ where the first term relates to the time spent for the computation of the subdomains and the second term relates to the time spent for the separators. The constants c_1 and c_2 are in the same range. For $k > \log_2(|P|)+4$ the first term dominates and we can derive from our data that

$$W \approx 2.5 \cdot 10^{-5} \cdot 2^{2k}/|P|.$$

Furthermore, we see that W nicely scales with $W^{k,|P|} = W^{k+1,4|P|}$ for $k \geq 7$.
Altogether, the part of the execution time which only depends on the number of communication steps is quite dominating. For example, even in the case $|P| = 16, k = 8$, the setup time S is still as large as the computation time W. We know that this effect must be much more crucial for parallelized multigrid methods since then communication has to take place on every level and is therefore dependent on the number k of levels. Thus, we infer that our approach is superior to most parallelized multigrid methods, especially on a cluster of workstations (with a binary-tree like configuration).

In Table 3 the efficiencies are given for $k = 7, .., 11$. Here, the efficiency E is defined by the ratio

$$E(|P|) = \frac{T(1)}{T(|P|) \cdot |P|}$$

where $T(|P|)$ denotes the time needed for the calculation on $|P|$ processors. For $k = 11$, $T(1)$ was obtained by multiplying $T(1)$ for $k = 10$ by four.

Table 3: Efficiencies E in %, Network 1.

$\|P\|$	2	4	8	16
$k = 7$	89.1	56.7	26.8	11.7
$k = 8$	97.0	85.7	62.3	34.1
$k = 9$	98.4	93.4	82.1	64.6
$k = 10$	99.2	97.4	92.0	81.2
$k = 11$				91.3

We see that we obtained good efficiencies (above 81 – 89 percent) for the case $k-6 \geq \log_2(|P|)$ even if we use a network of workstations with a relatively slow Ethernet-connection as a parallel computing system.

6.2 Clustered network with 64 Workstations

Additionally, we used a network of 64 HP 9000/720 workstations with a physical memory of 32 MB per processor (Network 2). It is configurated in 8 clusters of 8 workstations. Each cluster consists of one server and 7 clients which are connected by a single Ethernet. Moreover, the eight servers are connected by another Ethernet (see Figure 11).

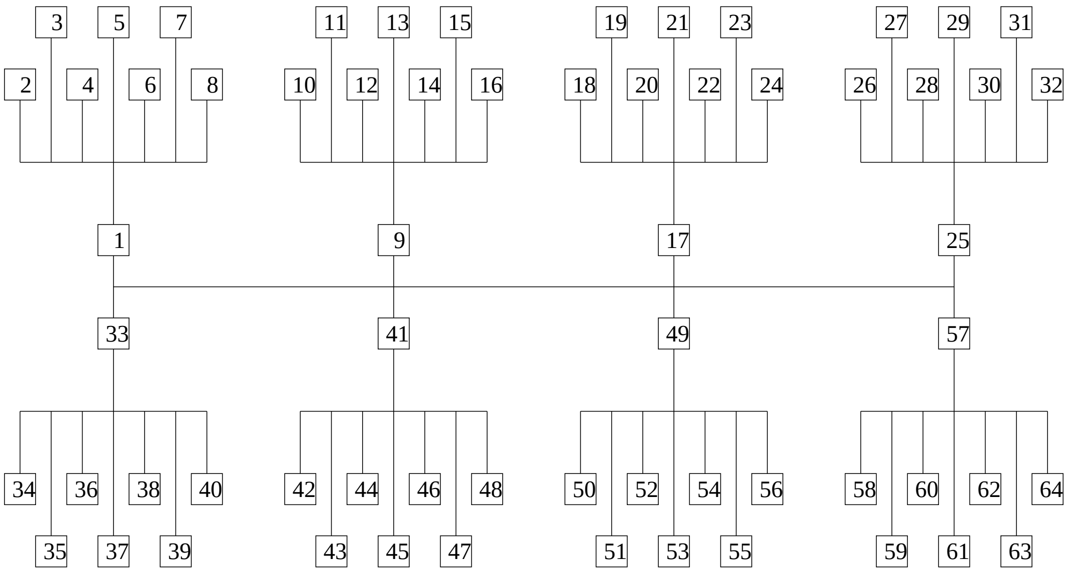

Fig. 11. *Network 2 (64 workstations, clustered).*

Of course, Network 2 is not as well adapted to our problem as Network 1. Here, data transport between adjacent levels of the process tree must be carried out on the same Ethernet (at least partially). Thus, we expect greater communication times C than for Network 1 with its specific tree-like configuration, whereas setup time S and computation time W are expected to be equal. For $k = 5, \ldots, 11$ and $|P| = 2^p$ with $p = 0, \ldots, 6$, we obtained the following results.

Table 4: Setup time S for $k = 5, \ldots, 11$, Network 2.

$\|P\|$	1	2	4	8	16	32	64
S	0.000	0.010 - 0.011	0.021 - 0.024	0.031 - 0.033	0.044 - 0.048	0.058 - 0.061	0.071 - 0.073

Table 5: Communication time C, computation time W and total time T, Network 2.

	$\|P\|$	1	2	4	8	16	32	64
$k = 5$	C	0.000						
	W	0.013						
	T	0.013						
$k = 6$	C	0.000	0.011					
	W	0.064	0.066					
	T	0.064	0.087					
$k = 7$	C	0.000	0.015	0.036				
	W	0.372	0.197	0.121				
	T	0.372	0.222	0.181				
$k = 8$	C	0.000	0.034	0.071	0.156			
	W	1.676	0.801	0.438	0.234			
	T	1.676	0.844	0.531	0.423			
$k = 9$	C	0.000	0.067	0.133	0.282	0.367	0.640	
	W	6.795	3.368	1.689	0.824	0.549	0.196	
	T	6.795	3.446	1.843	1.138	0.966	0.895	
$k = 10$	C			0.279	0.572	0.722	1.260	1.361
	W			6.784	3.289	1.680	0.685	0.347
	T			7.084	3.892	2.446	2.002	1.788
$k = 11$	C					1.508	2.515	2.778
	W					6.675	3.210	1.401
	T					8.227	5.785	4.250

Table 6: Efficiencies E in %, Network 2.

$\|P\|$	2	4	8	16	32	64
$k = 7$	83.7	51.5				
$k = 8$	99.2	79.0	49.4			
$k = 9$	98.6	92.2	74.7	44.0	23.7	
$k = 10$		95.9	87.3	69.5	42.4	23.7
$k = 11$				82.6	58.7	40.0

An analysis of our results shows that the setup time S behaves like $S \approx 0.11 \cdot \log_2(|P|)$ (see Table 4). Here, the constant is only half of the constant for Network 1. The computation time W shows nearly the same behaviour as for Network 1. Because of the configuration of Network 2, an analysis of the communication time is quite complicated and will not be given here. But we can see from our data that the communication time C and therefore the overall time T (Table 5) and the efficiencies E (Table 6) are slightly worse than for Network 1.

7 Conclusions

In this paper, we presented a new multilevel algorithm based on the generating system approach. Instead of level-oriented Gauss-Seidel iterations on the semidefinite system, which turn out to result just in standard multigrid algorithms, we used a domain-oriented ordering of the unknowns. As for conventional multigrid methods, the reduction rate is independent of the grid size. But now, in contrast to parallel multigrid methods, the number of communication steps does not depend on the number of levels but only on the number of processors. This property makes our algorithm well suited for parallel systems with high setup times like for example workstation networks. However, an efficient implementation of the domain-oriented algorithm is very tricky, in comparison to levelwise methods, because the recursive computation of the residuals involves not only the values of two neighbouring levels but also values of all levels under consideration, if we switch from one separator or subdomain to another.
We have also seen, that the communication times of our algorithm depend on the topology of the workstation network. On tree-like configurations as shown in Figure 5 and 10, data transport between different levels of the process tree can take place in parallel, whereas data transport is partly sequentially for clustered networks. Because of the low number of communication steps, we expect our algorithm to be superior to other parallel multilevel methods, especially if we use a Wide Area Network (WAN) where the setup time between different parts of the network would be extremely bad.

References

[1] P. Bastian. Parallel adaptive multigrid methods. Preprint 93-60, IWR Heidelberg, 1993.

[2] P. Bastian. Parallele adaptive Mehrgitterverfahren. PhD thesis, Universität Heidelberg, 1994.

[3] J. Bramble, J. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comp.*, 31:333–390, 1990.

[4] A. George. Nested dissection of a regular finite element mesh. SIAM, J. Numer. Anal. 10, 1973.

[5] M. Griebel. Grid- and point-oriented multilevel algorithms. In W. Hackbusch and G. Wittum, editors, *Incomplete Decomposition (ILU): Theory, Technique and Application, Proceedings of the Eighth GAMM-Seminar, Kiel, 1992, Notes on Numerical Fluid Mechanics*, volume 41, pages 32–46. Vieweg Verlag, Braunschweig, 1992.

[6] M. Griebel. Multilevel algorithms considered as iterative methods on semidefinite systems. *SIAM J. Sci. Comput.*, 15:547–565, 1994.

[7] M. Griebel. Multilevel Methoden als Iterationsverfahren über Erzeugendensystemen. Teubner-Skripten zur Numerik, H.G. Bock, W. Hackbusch, R. Rannacher, eds., 1994.

[8] M. Griebel and T. Neunhoeffer. Parallel point- and domain-oriented multilevel methods for elliptic PDE's on workstation networks. *submitted to J. Comp. Appl. Math.*

[9] R. Hempel and A. Schüller. Experiments with parallel multigrid algorithms using the SUPRENUM communications subroutine library. Arbeitspapiere der GMD 141, GMD, 1988.

[10] O.A. McBryan, P.O. Fredericson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C.A. Thole, and U. Trottenberg. Multigrid methods on parallel computers - a survey of recent developments. *Impact of Computing in Science and Engineering*, 3:1–75, 1991.

[11] P. Oswald. On discrete norm estimates related to multilevel preconditioners in the finite element method. Proc. int. Conf. Constr. Theory of Functions, Varna, 1991.

[12] P. Oswald. Norm equivalencies and multilevel Schwarz preconditioning for variational problems. Bericht Math/92/1, FSU Jena, Mathematische Fakultät, 1992.

[13] J. Xu. Iterative methods by space decomposition and subspace correction: A unifying approach. *SIAM Review*, 34(4):581–613, 1992.

[14] X. Zhang. Multilevel Schwarz methods. *Numerische Mathematik*, 63:521–539, 1992.