Parallel Point-oriented multilevel methods

Michael Griebel

September 22, 1993

Institut für Informatik, Technische Universität München
Arcisstraße 21, D-80290 München
email: griebel@informatik.tu-muenchen.de

Abstract

Instead of the usual basis, we use a generating system for the discretization of PDEs that contains not only the basis functions of the finest level of discretization but additionally the basis functions of all coarser levels of discretization. The Galerkin-approach now results in a semidefinite system of linear equations to be solved. Standard iterative GS-methods for this system turn out to be equivalent to elaborated multigrid methods for the fine grid system.

Beside Gauss-Seidel methods for the level-wise ordered semidefinite system, we study block Gauss-Seidel methods for the point-wise ordered semidefinite system. These new algorithms show basically the same properties as conventional multigrid methods with respect to their convergence behavior and efficiency. Additionally, they possess interesting properties with respect to parallelization. Regarding communication, the number of setup steps is only dependent on the number of processors and not on the number of levels like for parallelized multigrid methods. The amount of data to be communicated, however, increases slightly. This makes our new method perfectly suited to clusters of workstations as well as to LANs and WANs with relatively dominant communication setup.

1 Introduction

Recently, see [4], [6], a new concept for the development of multigrid and BPX-like multilevel algorithms has been presented. There, instead of a basis approach on the finest grid and the acceleration of the basic iteration by MG-coarse grid correction or a BPX-type preconditioner, a generating system is used to allow a non-unique level-wise decomposed representation of the solution. The degrees of freedom are associated to the nodal basis functions of all levels under consideration. With

this non-unique multilevel decomposed representation of a function, the Galerkin-approach leads to a semidefinite linear system with unknowns on all levels. Its solution is non-unique but in some sense equivalent to the unique solution of the standard problem on the finest grid.

Furthermore, it has been shown that traditional iterative methods for the semidefinite system are equivalent to modern elaborated multilevel methods applied to the standard system which exhibit optimal convergence properties. The conjugate gradient method (with appropriate diagonal scaling) for the semidefinite system is equivalent to the BPX-conjugate gradient method for the fine grid system. Gauss-Seidel-type iterations for the semidefinite system are equivalent to certain multigrid methods. For details, see [4], [6]. These methods are grid- or level-oriented and can be considered as level block techniques. An outer iteration switches from level to level, and an inner iteration operates on the specific grid.

Now, we consider the semidefinite system from a different point of view. We group together all unknowns which are associated to the same grid point. This results in a point-oriented method and can be considered as a point block technique. Now, an outer iteration switches from grid point to grid point. The local system that belongs to all basis functions of different levels centered in the same grid point can be solved either directly or by an inner iteration that runs over all levels that are associated to the grid point under consideration. Furthermore, grid points can be grouped together to form subdomains. In this sense, we get some sort of simple domain decomposition method which exhibits MG-like convergence properties. Compare also [5], [6], [7].

In contrast to the parallelization of a multilevel method where communication has to take place on all levels, our point block approach needs substantially less setup steps for the communication due to its domain decomposition qualities. In this sense, our new method is superior to other parallel multigrid and multilevel methods and very well suited to clusters of workstations or LANs and WANs with relatively dominant communication setup.

2 The semidefinite system

In this section, we introduce a generating system that replaces the usual finite element basis in the discretization of a boundary value problem of a partial differential equation. We then derive the associated semidefinite linear system and discuss its properties.

Consider a partial differential equation in d dimensions with a linear, second-order operator L on the domain $\Omega = (0,1)^d$, d=1,2,...,

$$Lu = f \text{ on } \Omega, \tag{1}$$

with appropriate boundary conditions and corresponding solution u. For reasons of simplicity, we restrict ourselves to homogeneous Dirichlet boundary conditions.

Given an appropriate function space V, the problem can be expressed equivalently in its variational form: Find a function $u \in V$ with

$$a(u,v) = (f,v) \quad \forall\ v \in V. \tag{2}$$

(In the case of homogeneous Dirichlet boundary conditions, V would be the Sobolev space $H_0^1(\Omega)$.) Here, a: $V \times V \rightarrow \mathbb{R}$ is a bounded, positive-definite, symmetric bilinear form and $(.,.)$ is the linear form for the right-hand side. Let $\|.\|_a := \sqrt{a(.,.)}$ denote the induced energy norm. We assume that V is complete with respect to $\|.\|_a$, which is true if $a(.,.)$ is H_0^1-elliptic. The Lax-Milgram lemma then guarantees the existence and uniqueness of the solution of (2). If we consider directly the functional $J(u) := 1/2 \cdot a(u,u) - (f,u)$, the problem can be stated alternatively as minimization of $J(u)$ in V.

2.1 Spaces, bases, and the generating system

Assume we are given a sequence of uniform, equidistant, and nested grids

$$\Omega_1 \subset \Omega_2 \subset ... \subset \Omega_{k-1} \subset \Omega_k \tag{3}$$

on $\bar{\Omega}$ with respective mesh sizes $h_l = 2^{-l}$, l=1,..,k, and an associated sequence of spaces V_l of piecewise d-linear functions,

$$V_1 \subset V_2 \subset .... \subset V_{k-1} \subset V_k, \tag{4}$$

with dimensions

$$n_l := dim(V_l) = (2^l - 1)^d, l = 1, .., k. \tag{5}$$

Here, 1 denotes the coarsest and k the finest level of discretization. Consider also the sequence

$$N_1 \subset N_2 \subset ... \subset N_{k-1} \subset N_k \tag{6}$$

of sets of inner grid points $N_l = \{x_1, ..., x_{n_l}\}$ of the grid Ω_l, l=1,..,k.

The standard finite element basis, that spans V_l on the equidistant grid Ω_l, is denoted by B_l. It contains the nodal basis functions $\phi_i^{(l)}$, i=1,..,n_l, which are defined by

$$\phi_i^{(l)}(x_j) = \delta_{i,j}, \quad x_j \in N_l. \tag{7}$$

Now, any function $u \in V_k$ can be expressed uniquely by

$$u = \sum_{\phi \in B_k} u_\phi \cdot \phi \tag{8}$$

with the vector $u_k := (u_\phi)_{\phi \in B_k}$ of nodal values associated to some given ordering of the functions of B_k.

In contrast to this conventional basis approach we now consider the set of functions E_k defined as the union of all the different nodal bases B_l for the levels l=1,..,k,

$$E_k = B_1 \cup B_2 \cup .... \cup B_{k-1} \cup B_k. \tag{9}$$

Obviously, being a linearly dependent set of functions, E_k is no longer a basis for V_k, but merely a generating system. See Figure 1 for a simple 1D example.

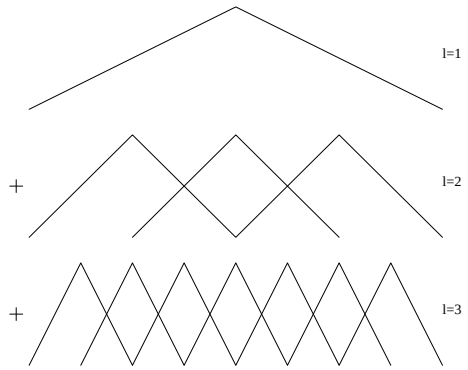

Figure 1: Functions of the generating system E_3 in the 1D case.

In any case, an arbitrary function $u \in V_k$ can be expressed in terms of the generating system by

$$u = \sum_{\phi \in E_k} w_\phi \cdot \phi \tag{10}$$

with the vector $w_k^E := (w_\phi)_{\phi \in E_k}$ associated to some given ordering of the functions of E_k.

Here and in the following, we denote representations in terms of the generating system E_k by the superscript E. The length of w_k^E is

$$n_k^E = \sum_{l=1}^{k} n_l, \tag{11}$$

which is in the 1D case about twice, in the 2D case about 4/3 times, and in the 3D case about 8/7 times as large as the length of the vectors for the basis representation above. This is due to the geometric rate of decrease of the number of grid points from fine to coarse levels with factors of 1/2, 1/4, and 1/8 for 1D, 2D, and 3D, respectively. The generalization of this concept to higher dimensions is straightforward.

Note that the representation of u in terms of E_k is not unique. In general, there exists a variety of level-wise decompositions of $u \in V_k$. However, for a given representation w_k^E of u in E_k, we can easily compute its unique representation u_k with respect to B_k. This involves in 2D the bilinear interpolation which can be expressed and implemented by MG-prolongation operators. For details, see [6], [8].

2.2 Galerkin-approach and linear systems

Using the nodal basis B_k, the Galerkin-approach results in the discrete variational problem for $u \in V_k$

$$\forall \phi_i \in B_k : a(u, \phi_i) = f(\phi_i) \tag{12}$$

and the equivalent linear system of equations for the vector of nodal values

$$L_k u_k = f_k, \tag{13}$$

where

$$\begin{array}{rcll} (L_k)_{i,j} & := & a(\phi_j^{(k)}, \phi_i^{(k)}), & 1 \le i,j \le n_k \\ (f_k)_i & := & (f, \phi_i^{(k)}), & 1 \le i \le n_k \end{array} \tag{14}$$

for some appropriate ordering of the functions of B_k.

For the generating system E_k, the Galerkin-approach leads to the variational problem

$$\forall \phi_i \in E_k : a(u, \phi_i) = f(\phi_i) \tag{15}$$

and, with representation (10), to the linear system

$$L_k^E w_k^E = f_k^E, \tag{16}$$

where

$$\begin{array}{rcll} (L_k^E)_{i,j} & := & a(\phi_j, \phi_i), & 1 \le i,j \le n_k^E \\ (f_k^E)_i & := & (f, \phi_i), & 1 \le i \le n_k^E \end{array} \tag{17}$$

for some appropriate ordering of the functions of E_k.

The system $L_k^E w_k^E = f_k^E$ has the following properties. The matrix L_k^E is semidefinite and has the same rank as L_k. Thus $n_{k-1}^E = n_k^E - rank(L_k)$ eigenvalues of L_k^E are zero. The system is solvable because the right-hand side is constructed in a consistent manner. It has not just one unique solution, but a variety of different solutions. However, the evaluation of two different solutions $w_k^{E,1}$ and $w_k^{E,2}$ with respect to their representation in B_k by means of MG-prolongation operators results in the unique solution u_k of the system $L_k u_k = f_k$. Therefore, it is sufficient to compute just one solution of the enlarged semidefinite system to obtain, via interpolation, the unique solution of the system $L_k u_k = f_k$. Note that the enlarged matrix L_k^E contains the submatrices L_l that arise from the use of the standard basis B_l, $l = 1, .., k$. A similar property holds for the right-hand sides.

3 Gauss-Seidel-type iterative methods

Now, multilevel-type algorithms are easy to construct. In [4], [6], it is shown that (after appropriate scaling) the conjugate gradient method for the semidefinite system $L_k^E w_k^E = f_k^E$ is equivalent to CG with the preconditioner of Bramble, Pasciak, and Xu [1] for the system $L_k u_k = f_k$. Furthermore, the Gauss-Seidel iteration for the semidefinite system is equivalent to the multigrid-method with Gauss-Seidel

smoother applied to $L_k u_k = f_k$. Here, depending on the ordering of the unknowns of the semidefinite system, different MG-cycle strategies can be modeled easily. See the discussion in [4] and [6] for further guidance.

In the following, we focus on Gauss-Seidel-type iterations for the semidefinite system only. First, we consider level-oriented Gauss-Seidel iterations and state that they are equivalent to certain multigrid methods. An outer iteration switches from level to level, and an inner iteration operates on the respective grids. Then, we study point-oriented Gauss-Seidel iterations. There, the unknowns of the semidefinite system that belong to basis functions centered at the same grid point are grouped together. Now, an outer iteration switches from grid point to grid point, and an inner iteration or a direct solver works on the local subsystem of the semidefinite problem belonging to the respective grid point. In the same way, domain-oriented Gauss-Seidel methods can be constructed for the semidefinite system.

Note that, in both cases, it is not necessary to assemble the matrix L_k^E and the right-hand side f_k^E explicitly. It is possible to use MG-prolongation and MG-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. For an example for the level-oriented approach, see (35) in [4] or the more detailled explanations in [6]. Furthermore, by storing and updating certain parts of the current residual in (at least two) additional vectors, it is possible to realize, for appropriate traversal orderings, the implementation of level-, point- and domain-oriented Gauss-Seidel methods to need $O(n_k^E) = O(n_k)$ operations per iteration step, only. Especially for the point- and domain-oriented block Gauss-Seidel methods, this is quite technical. A detailed description of implementation details will be given in [8]. Altogether, the required storage and the number of operations to perform a Gauss-Seidel- or block Gauss-Seidel-step is proportional to the number of grid points employed.

3.1 Grid-oriented Gauss-Seidel algorithms

We consider Gauss-Seidel iterations for the semidefinite system $L_k^E w_k^E = f_k^E$. As usual, we decompose the semidefinite matrix L_k^E by $L_k^E = D_k^E + F_k^E + (F_k^E)^T$, where D_k^E and F_k^E denote the diagonal and strictly lower triangular parts of L_k^E, respectively. Then, the Gauss-Seidel iteration (GS) is expressed by using $C_k^{E,GS} := (D_k^E + F_k^E)^{-1}$, and its symmetric counterpart SGS is expressed by using $C_k^{E,SGS} := (D_k^E + F_k^E)^{-T} D_k^E (D_k^E + F_k^E)^{-1}$ in the iteration

$$w_k^{E,(\kappa+1)} := w_k^{E,(\kappa)} + C_k^E (f_k^E - L_k^E w_k^{E,(\kappa)}). \tag{18}$$

Note that $D_k^E + F_k^E$ is generally not symmetric, but it is positive definite and invertible.

Of course, the lower triangular part of L_k^E depends on the ordering of the unknowns. For practical reasons, i.e. to maintain $O(n_k)$ operations per iteration

step, not all orderings are advisable. Here, we order the unknowns grid-wise (starting for example with the finest grid). This leads to a grid-oriented block partition of L_k^E. The unknowns of each level can be ordered lexicographically or, in the 2D case, in a four color manner.

Note that this level-wise ordering corresponds to a level-wise decomposition

$$V_k = \sum_{l=1}^{k} V_l = \sum_{l=1}^{k} \sum_{i=1}^{n_l} V_{l,x_i} \tag{19}$$

of V_k, where $V_{l,x_i} = span\{\phi_i^{(l)}\}$.

In [4], [6], it was shown that, for a level-wise ordering of the unknowns, the Gauss-Seidel iteration for the semidefinite system corresponds to the multigrid method for the standard system. The switching from grid to grid in the multigrid method corresponds to an outer (block) Gauss-Seidel iteration for the semidefinite system, whereas the MG-smoothing steps resemble an inexact solver for each block by inner Gauss-Seidel iterations. Especially the multigrid V-cycle with one pre- and post-smoothing step by Gauss-Seidel iterations becomes the SGS-method, which is also known as Aitken's double sweep. But other MG-cycle types can be modelled by different orderings of the block GS-traversal. The case of multiple smoothing steps corresponds to multiple inner iterations. Furthermore, other smoothers can be incorporated in the block GS-method as inner iterations.

These algorithms on the semidefinite system (16) can be interpreted alternatively as subspace correction methods [13]. The relaxation of an iterate $u^{(\kappa)} \in V_k$ then takes place with respect to a $\phi \in E_k$ by

$$u^{(\kappa+1)} := u^{(\kappa)} + \lambda \cdot \phi \quad \text{with} \quad a(u^{(\kappa+1)}, \phi) = f(\phi). \tag{20}$$

The different MG-type GS-algorithms for the generating system involve the cyclic application of (20) for a corresponding sequence of functions of E_k.

3.2 Point-oriented block Gauss-Seidel algorithms

Now, we partition the unknowns of the semidefinite system into groups and perform a block Gauss-Seidel iteration on the associated block-partitioned system. Thus, we decompose the semidefinite matrix L_k^E by $L_k^E = \mathcal{D}_k^E + \mathcal{F}_k^E + (\mathcal{F}_k^E)^T$, where $\mathcal{D}_k^E$ and $\mathcal{F}_k^E$ now denote the block diagonal and strictly lower block triangular parts of L_k^E, respectively. Then, the block GS-iteration is expressed by using $\mathcal{C}_k^{E,GS} := (\mathcal{D}_k^E + \mathcal{F}_k^E)^{-1}$, and its symmetric counterpart SGS is expressed by using $\mathcal{C}_k^{E,SGS} := (\mathcal{D}_k^E + \mathcal{F}_k^E)^{-T} \mathcal{D}_k^E (\mathcal{D}_k^E + \mathcal{F}_k^E)^{-1}$ in the iteration

$$w_k^{E,(\kappa+1)} := w_k^{E,(\kappa)} + \mathcal{C}_k^E (f_k^E - L_k^E w_k^{E,(\kappa)}). \tag{21}$$

Note that $\mathcal{D}_k^E + \mathcal{F}_k^E$ is invertible only for certain choices of partitions, where each group contains unknowns associated to linear independent functions. Nevertheless,

if this is not fulfilled, an iterative method still produces a non-unique solution for the corresponding semidefinite subsystem.

Alternatively, one block Gauss-Seidel step can be interpreted as a subspace correction method. The relaxation of an iterate $u^{(\kappa)} \in V_k$ now takes place simultaneously with respect to a set $\Phi \subset E_k$ by

$$u^{(\kappa+1)} := u^{(\kappa)} + \sum_{\phi \in \Phi} \lambda_\phi \cdot \phi \quad \text{with} \quad \forall \phi \in \Phi : a(u^{(\kappa+1)}, \phi) = f(\phi). \tag{22}$$

For the computation of the values of $\lambda_\Phi = \{\lambda_\phi : \phi \in \Phi\}$, the system

$$L_\Phi \lambda_\Phi = r_\Phi \tag{23}$$

with

$$\begin{array}{rcll} (L_\Phi)_{i,j} & := & a(\phi_j, \phi_i), & \phi_i, \phi_j \in \Phi \\ (r_\Phi)_i & := & (f_\Phi)_i - a(u^{(\kappa)}, \phi_i) & \\ (f_\Phi)_i & := & (f, \phi_i), & \phi_i \in \Phi, \end{array} \tag{24}$$

has to be solved. The block Gauss-Seidel method for the semidefinite system is the cyclic application of (22) for a given sequence of (disjoint) subsets of E_k. This type of iteration is only convergent if the union of all involved sets of functions span V_k. Note that the case of non-disjoint subsets can still be denoted in terms of (22), but not more by (21).

Of course, this approach is heavily dependent on the chosen sequence of subsets of E_k. In the following, we suggest a point-oriented approach. We group all unknowns together that are associated to the same grid point. This results in point-oriented methods and can be considered as a point-block technique. Now, an outer iteration switches from grid point to grid point. The local system that belongs to all basis functions of different levels centered in the same grid point can be solved either directly or by an inner iteration running over all unknowns associated to the grid point under consideration.

To be more specific, we consider as blocks the unknowns associated to all functions that are centered in the same grid point $x \in N_k$:

$$P_x := \{\phi \in E_k : \phi(x) = 1\} \tag{25}$$

This corresponds to a point-oriented decomposition

$$V_k = \sum_{x \in N_k} \sum_{l : x \in N_l} V_{l,x} \tag{26}$$

of the space V_k. Note that in comparison with the level-wise decomposition (19) just the summations are exchanged.

Now, we can step through the set N_k of grid points and relax simultaneously the unknowns that belong to the same grid point. This results in systems of linear equations (23) with $\Phi = P_x$, $x \in N_k$, that form the block diagonal matrix $\mathcal{D}_k^E$

involved in (21). Note that the size of the systems belonging to $x \in N_l \setminus N_{l-1}, l = k, .., 2$, is $k - l + 1$. The size of the system belonging to the center point $x = (0.5, 0.5)$ is k. For the case of an operator with constant coefficients, the point-block matrices L_{P_x} are full, definite and symmetric Toeplitz matrices. It turns out that their condition number is $O(1)$. See also the results in [6]. Thus, the solution of (23) for $x \in N_l \setminus N_{l-1}$ can be obtained by some appropriate iterative method in $O((k - l + 1)^2)$ operations. The number of operations necessary for solving all arising point-block subsystems is in 2D

$$C\frac{3}{4}n_k \left(1^\gamma + \frac{1}{4}2^\gamma + \frac{1}{16}3^\gamma + \frac{1}{64}4^\gamma + ... \frac{1}{4^{k-l}}(k+1-l)^\gamma + ... + \frac{1}{4^{k-1}}k^\gamma, \right) \quad (27)$$

where $\gamma = 2$ for SGS and $\gamma = 3$ for direct solution by Gaussian elimination. Altogether, this results in $O(n_k)$ operations. The coupling between two point-blocks is described by the respective submatrix entries of $\mathcal{F}_k^E$, or, in (24), by $a(u^{(\kappa)}, \phi_i), i = 1, .., |P_x|$, with a non-unique representation (10) of $u^{(\kappa)}$. It is easy to see that by this coupling information is exchanged on all respective levels of discretization simultaneously.

In practice, however, not all traversal orderings through the set of grid points N_k are advisable. We restrict ourselves to the sequence of point-blocks where the grid points of $N_k \setminus N_{k-1}$ are ordered first, for example in a three-color fashion. Second, the remaining grid points of N_{k-1} are ordered recursively in the same way. Then, neither the point-block matrix nor the off-diagonal blocks have to be assembled explicitly, but can be expressed by means of prolongation and restriction operators. However, the efficient solution of the diagonal-block problems and the block Gauss-Seidel iteration is still tricky to implement. It can be shown that one point-block Gauss-Seidel step can be implemented to need $O(n_k)$ operations. Details are reported in [8].

Note that it is in general not necessary to compute the exact solution for each point-block problem. Often, a few GS- or SGS-iterations are sufficient. In the extreme case of one GS step only, we obtain the GS iteration for the semidefinite system (16) with just a special point-oriented ordering.

3.3 Domain-oriented block Gauss-Seidel algorithms

The point-oriented approach can be easily generalized. We may allow an arbitrary domain decomposition of Ω into K non-overlapping subdomains with associated decomposition

$$N_k = \bigcup_{i=1}^{K} N_k^i, \quad N_k^i \cap N_k^j = \{\} \text{ for } i \neq j \quad (28)$$

of the grid points N_k. Now, we group together the unknowns of the semidefinite system that are associated to functions of E_k whose center points are situated in

the same N_k^i by

$$P_{N_k^i} := \bigcup_{x \in N_k^i} P_x. \tag{29}$$

The resulting block Gauss-Seidel algorithm now switches from subdomain to subdomain in some prescribed order. Thus, we obtain some sort of simple domain decomposition method which exhibits MG-type convergence properties.

Note that in contrast to the point block approach the arising subproblem matrices are now in general not longer invertible, since they can be semidefinite. However, an iterative method still is able to produce a non-unique solution for the corresponding subsystem. Alternatively, a direct solver for any definite subsystem with full rank can be used.

For practical purposes, a nested dissection-like decomposition [2] of the grid points into subdomains is advisable. Compare also Figure 7. Then, by using multigrid prolongation and restriction operators, the submatrices for the subdomains have not to be assembled explicitly, and one overall block Gauss-Seidel iteration can be performed with $O(n_k)$ operations.

3.4 Convergence properties

Based on Xu [6], Zhang [15] and [13], it can be shown that the above mentioned plain Gauss-Seidel methods, the point-oriented Gauss-Seidel methods and to some extend also the domain-oriented Gauss-Seidel methods working on the semidefinite system converge *independent* of k or n_k^E without any regularity assumption.

In contrast to Xu [13], the use of the generation system gives us the possibility to express the estimates for the convergence rate directly in terms of the semidefinite matrix itself, which is similar to conventional convergence estimates for classical Gauss-Seidel and SOR methods, c.f. [6].

Interestingly, the arising esimate for the plain Gauss-Seidel method (as well as for the point-orineted algorithm) is *independent* of the ordering of the generating system and gives an upper bound for *all* possible orderings. Thus, it also holds for plain Gauss-Seidel method with level-oriented traversal ordering, that corresponds to a multigrid V-cycle with either one pre- or one post-smoothing step with any Gauss-Seidel smoother (lexicographic, red-black, four-color) on each level. Furthermore, since the symmetric Gauss-Seidel method consists just of two Gauss-Seidel iterations with one in reversed order, we immediately obtain the estimate $\rho^{SGS} \leq (\rho^{GS})^2 \leq c < 1$. for the convergence rate of the symmetric Gauss-Seidel method (i.e. V-cycle with one pre- and one post-smoothing step), which is also independent of k and n_k^E.

In addition, not only level-wise orderings have to be considered. We can give up the level-fixed approach that is inherent in usual multigrid methods and gain the freedom to perform Gauss-Seidel relaxations in a point- or domain-oriented ordering as well without loosing the property that the convergence rate is independent of k and n_k^E. For example, the point- and domain-oriented Gauss-Seidel

methods where the point- and domain-subsystems are not solved exactly but only iterated by one Gauss-Seidel step reduce to a simple Gauss-Seidel method for the overall semidefinite system with just a special point- or domain-oriented traversal ordering. Therefore, their convergence rate is also independent of k and n_k^E.

Similar observations can also be made for the point- and for the domain-oriented methods. For further details, we refer to [6].

4 Parallelization properties

So far, we have mentioned that point-oriented Gauss-Seidel methods or domain-oriented Gauss-Seidel methods (with one Gauss-Seidel step in each subdomain) possess a convergence rate that is independent of k just like level-oriented Gauss-Seidel methods (multigrid). However, the convergence rates measured in numerical experiments for point- and domain-oriented methods, c.f. [5], are slightly worse than that for sophisticated level-oriented methods. We obtain convergence factors of 0.1-0.2. In addition, the amount of operations to perform one iteration step is somewhat larger, roughly a factor 5-14 depending on the respective implementations to be compared. (Up to now, we merely have a not yet well optimized prototype version of the point-oriented method and must compare it to a perfectly tuned implementation of the multigrid method, the HTMG-method [3]).

Thus, we can pose the question whether we need the point- and domain-oriented method at all. The answer gets clear if we consider the parallelization properties of the different methods on most presently available MIMD computers and especially on networks of workstations or LANs and WANs. There, the time necessary to setup communication is often relatively large. Thus, it is 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 dominated exclusively by the setup time. Then, the time necessary for data exchange and parallel computation is not important any more.

This is where the level- and the point-/domain-oriented methods are different. For the level-oriented method, the number of communication steps is dependent on the number k of levels whereas for the point-/domain-oriented method it only depends on the number of processors P, i.e. it is of the order $\mathcal{O}(\log P)$. However, in the point- and domain-oriented case, the overall amount of data to be exchanged is slightly larger. For practical situations with $n_k \gg P$, this property of the point-/domain-oriented method is a crucial advantage that pays off on certain MIMD computers and especially on workstation networks or LANs and WANs with relatively dominant communication setup. In the following we will explain this difference between the level- and point- or domain-oriented approach in more detail.

An efficient implementation of the level-oriented Gauss-Seidel method for the semidefinite system can follow the multigrid strategy. In general, it consists of the smoothing step, the computation of the residual, the restriction operator and the

prolongation operator. For the parallelization of a multigrid method, the domain Ω is usually subdivided into P more or less equally sized subdomains. Then, each subdomain with his grid points on all levels (together with some overlapping border grid points) is associated to its specific processor. Usually array-, tree- or pyramid-type processor topologies are used [10].

Following this domain decomposition principle, the different components of a multigrid method (smoothing, residual computation, restriction, prolongation) can be executed, at least to some extent, in every subdomain independent of each other. This holds especially for grid points, that are situated for each respective level sufficiently in the interior of each subdomain. However, for grid points that are contained in the local boundary of the subdomains or that are within one mesh width distance from the local boundary, the situation looks different. In general, for the computation of the multigrid components in these points, data of adjacent grid points is necessary, that can be situated in some adjacent subdomain and that thus is stored on another processor. Therefore, the exchange of data between adjacent processors is necessary. This is usually performed collectively for all border points that belong to two adjacent processors in one communication step. Since the multigrid methods works level by level, this makes clear, that in a parallel multigrid method data has to be exchanged on *each* level of the multigrid hierarchy. For further details, see [9], [10].

Another bottleneck for the parallelization of multigrid methods occurs on the coarser grids. These grids contain relatively few grid points. Thus, only few processors can work here in parallel and the remaining processors have to stay idle. Additionally, in an array-like processor topology, certain adjacent grid points on coarse grids are not longer associated to adjacent processors. Then, communication has to take place sequentially between distant processors and is therefore more costly. Furthermore, processor load on these coarser grids can in general not be distributed equally. Therefore, so-called agglomeration techniques [9] have to be applied.

Altogether, for the parallelization of level-oriented methods, communication is necessary on *every* level. Furthermore, the computations on coarser levels can not employ all available processors, they are usually not well balanced and, additionally, it can be necessary to exchange data between distant processors. Theoretically, it is possible to obtain a parallel complexity of $\mathcal{O}(\log n_k)$ if a sufficiently large number of processors is available. Practically, however, this is often not the case. There, $P \ll n_k$ usually holds, and the overall computation time of a parallel multigrid method is strongly dependent on the number of communication steps and the communication properties of the parallel computing system. It can even happen that the communication setup time dominates both the data exchange time and the computation time and spoils the speed up and the efficiency.

For the simple one-dimensional case with $k = 5$, we see in Figure 2 the hierarchy of grids employed in the multigrid method and its distribution to $P = 3$ processors. Arrows indicate schematically the necessary communication steps.

It is clear that due to the level-wise computation the number of necessary

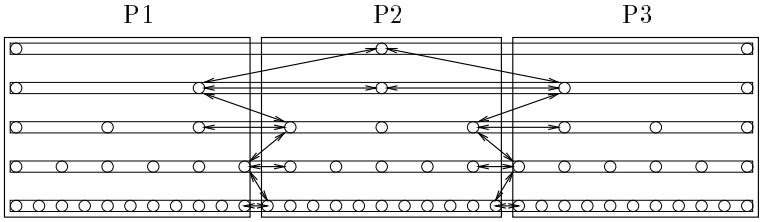

Figure 2: Distribution of the level-oriented method to 3 processors.

communication steps is of the order $\mathcal{O}(k)$, whereas, at least in our 1D-example, the amount of data to be exchanged in each step is of the order $\mathcal{O}(1)$. Thus, the number of communication steps is dependent on k ond not on P.

Now, we turn to the parallelization properties of the point- and domain-oriented methods. To explain the basic parallelization strategy and the requirements for communication and distributed storage and to show the main difference to the level-oriented methods, we restrict ourselves for reasons of simplicity to the one-dimensional case. In the point-oriented Gauss-Seidel method for the semidefinite system, all unknowns that belong to different levels but to the same grid point are grouped together and treated collectively as block in the iteration procedure.

For the parallelization of this method we apply the "divide and conquer"-principle. In a first step, we assign the center point of the domain together with all its unknowns of w_k^E and further data (like right hand side, certain parts of the residual) that belong to the center point to a processor $P1$. Then, the remaining grid points and all their degrees of freedom with respect to the generating system are split into two mutually independent subsets, that are assigned to two further processors ($P2$, $P3$). Altogether, we obtain the situation as shown schematically in Figure 3. Once again, arrows denote necessary communication steps.

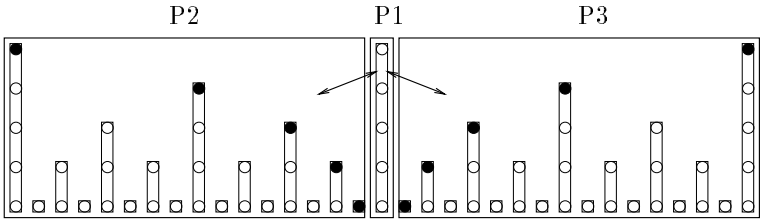

Figure 3: Schematical distribution of the point-oriented method to 3 processors.

First, the unknowns that belong to the left and right subsystem of the semidefinite system are relaxed simultaneously on the processors $P2$ and $P3$. Then,

necessary data (the associated grid points are marked in Figure 3 by $\bullet$) is sent to processor $P1$ and the unknowns that belong to the center point can be relaxed or computed exactly. The resulting new values have to be sent to the processors $P1$ and $P2$ and the iteration can continue.

Now, a first difference to the level-oriented method of Figure 2 gets clear: Only the two processors $P2$ and $P3$ can work simultaneously and they have to stay idle if computations are performed on processor $P1$. However, there are only two (simultaneous) communication steps necessary which is independent of k. It can be seen directly that, at least for our simple 1D example with only three processors, the number of communication steps is only of the order $\mathcal{O}(1)$. However, the amount of the data to be exchanged is now of the order $\mathcal{O}(k)$. In contrast to the level-oriented method of Figure 2, the point-oriented approach allows to collect the larger amount of data belonging to different level and to exchange them *collectively* in one communication step.

In the case of many processors, the left and right subproblem of the semidefinite system can be further subdivided in a *recursive* manner. In a natural way, we then obtain a tree-like parallelization structure. Each node of the tree contains the parts of the semidefinite system that are associated to the point block of the respective grid point. The leaves contain the subproblems that belong to the respective subdomains. The point block algorithm sweeps through this tree level by level. Now, the processors of each level work in parallel. This is indicated in Figure 4.

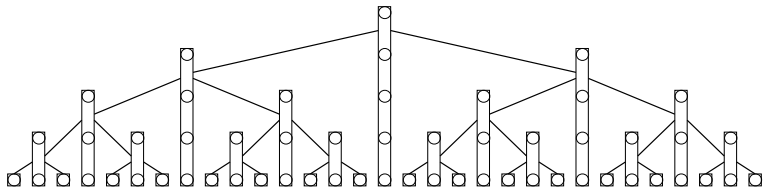

Figure 4: Tree-like parallelization structure of the point-oriented method.

To maintain the data supply of the processor nodes over many levels in the case of recursive subdivision, it is not more sufficient to exchange only the data between father- and son-processors, that belong to the points of Figure 3 that are marked by $\bullet$. For example, for the computation on level l (here $l = 2$), data is necessary that belongs to the points of Figure 5 (below) that are marked by $\bullet$. However, for the computation on level $l - 1$, data is necessary that belongs to the points of Figure 5 (above) that are marked by $\bullet$.

Therefore, certain additional values have to be stored and updated in every node of the processor tree to maintain data supply over many levels. The corresponding points on level l are shown in Figure 6. This additional data only doubles the amount of data to be stored in each processor and results in a factor 2 for the

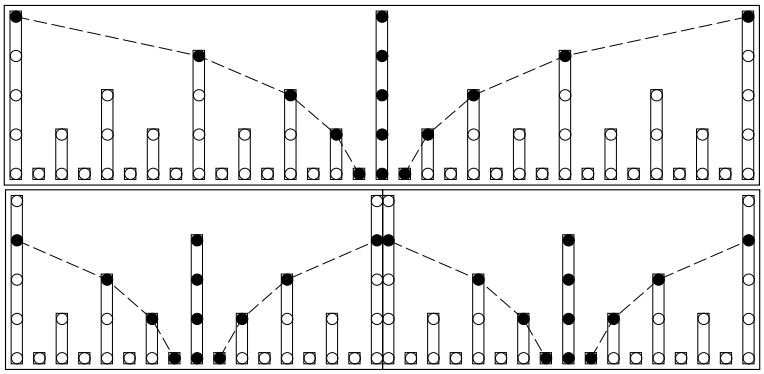

Figure 5: Points per processor that correspond to data necessary for the computation.

size of the data packages to be exchanged between father- and son-processors. The amount of storage as well as the amount of data to be exchanged remains proportional to $k - l + 1$ for each processor on level l.

A short analysis (sum up over the processor tree) then gives a *parallel* complexity of $\mathcal{O}(n_k/P)+\mathcal{O}((\log P-1)(\log n_k-(\log P)/2+1))$ for the number of operations, $\mathcal{O}((\log P-1)(\log n_k-(\log P)/2+1))$ for the amount of data to be exchanged and $\mathcal{O}(\log P)$ for the number of communication steps of the overall algorithm. Theoretically, with $P \approx n_k$, we obtain now a parallel complexity of $\mathcal{O}((\log n_k)^2)$ only, whereas the parallelization of the multigrid method (on a multilevel-type processor topology but not on a binary tree processor topology !) even results in $\mathcal{O}(\log n_k)$. However, remember that the setup time is now not more dependent on $\log n_k$ but only on $\log P$. Thus, mainly on networks of workstations as well as on LANs and WANs where the set up time contributes dominantly to the overall execution time, run time advantages can be expected for practical values of P and n_k.

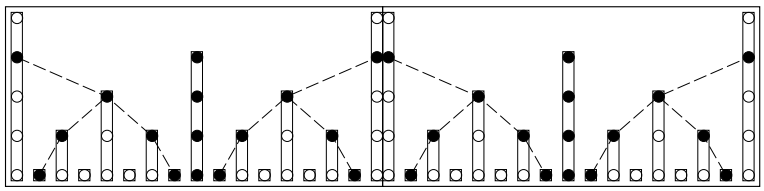

Figure 6: Points per processor that correspond to data necessary for the computation and communication.

This "divide and conquer"-approach for the parallelization can be generalized to the two-dimensional case. Then, we decompose the domain Ω along a middle line and assign the corresponding part of the semidefinite system to a first processor. The remaining part of the semidefinite system is split into two independent parts that can be treated in parallel. These two subsystem can be further subdivided in a recursive manner. If this subdivision always takes place in the same direction we obtain a stripe-wise decomposition of the domain, see figure 7 (left). Furthermore, the direction of the subdivision can be altered with each step. In both cases we obtain a binary tree-type parallelization structure where the subproblems of the semidefinite system that correspond to a node of the tree can be treated on each level in parallel. The alternating subdivision is closely related to the decomposition of the domain into four parts by means of a separating "cross". Then, we obtain a tree-like parallelization structure, where each node possesses four sons, compare 7 (right).

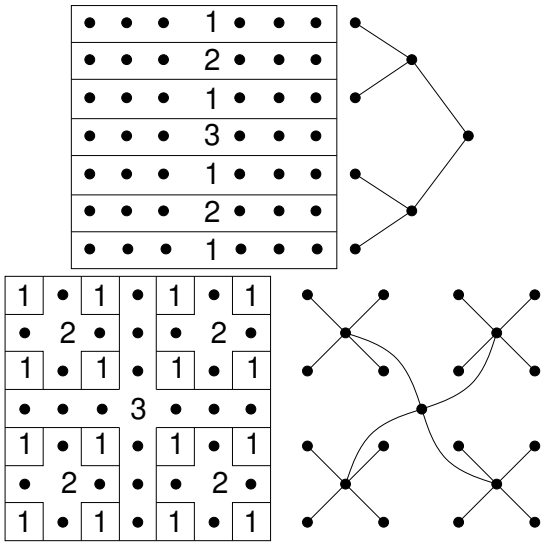

Figure 7: Stripe- and box-wise nested dissection decompositions and their parallelization structures.

Now, similar to the 1D case (compare Figure 5), data of adjacent points of each level is necessary for the computation of the separator-subproblems and has to be stored on the respective processors. Furthermore, to maintain data supply over many levels of the processor tree, certain additional values have to be stored, updated and exchanged in every node of the processor tree similar to the 1D case (compare Figure 6). A more detailled discussion on that problem is found in [6].

Note at last that the iteration of the one-dimensional multilevel subsystems of

the semidefinite system that belong to a separator-line or -cross can be further decomposed in a *second* parallelization step, which is analogous to the parallelization strategy in the one-dimensional case.

5 Concluding remarks

In this paper we presented different multilevel algorithms based on the generating system approach. We studied level-oriented techniques where Gauss-Seidel methods for the arising semidefinite system turn out to result just in standard multigrid algorithms. Additionally, we presented new point- and domain-oriented methods. There, block Gauss-Seidel iterations for the semidefinite system are obtained that exhibit a reduction rate independent of the grid size and the number of levels like conventional multigrid methods. Furthermore, these algorithms possess interesting properties with respect to parallelization. Especially the property that the number of communication steps and thus the setup is only dependent on P and not on k like for multigrid methods makes these algorithms well suited to networks of workstations. Meanwhile, first parallelization results have been obtained on different MIMD computers (IPSC/860, nCube2, Transputers), virtual shared memory machines (KSR-1), and networks of workstations (HP9000/720 with Ethernet) but this will be reported elsewhere.

In [7], our level- and point-oriented approach is adopted to an extended generating system that additionally contains the basis functions of all grids that result from semi-refinement steps with respect to both coordinate directions. Furthermore, the full and the sparse grid case [14] is considered. For the level-oriented methods, we obtain multigrid algorithms similar to that of Naik and van Rosendale [12] and Mulder [11]. For the point- and domain-oriented approach, analogous point block Gauss-Seidel methods as well as BPX-like preconditioners are derived.

References

[1] J. Bramble, J. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 31 (1990), pp. 333–390.

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

[3] M. Griebel, *Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen-Transformations-Mehrgitter-Methode*, TU München, Institut f. Informatik, TUM-I9007; SFB-Report 342/4/90 A, 1990.

[4] ——, *Multilevel algorithms considered as iterative methods on indefinite systems*, TU München, Institut f. Informatik, TUM-I9143; SFB-Report 342/29/91 A, 1991.

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

[6] ——, *Punktblock-Multilevelmethoden zur Lösung elliptischer Differentialgleichungen*, Habilitationsschrift, Institut für Informatik, TU München, 1993.

[7] M. Griebel, C. Zenger, and S. Zimmer, *Multilevel Gauss-Seidel-algorithms for full and sparse grid problems*, Computing, 50 (1993), pp. 127–148.

[8] M. Griebel and S. Zimmer, *Multilevel Gauss-Seidel-algorithms for full and sparse grid problems - Part II: Implementational aspects*, SFB-Report, TU München, Institut f. Informatik, 1993.

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

[11] W. Mulder, *A new multigrid approach to convection problems*, CAM Report 88-04, (1988).

[12] N. H. Naik and J. van Rosendale, *The improved robustness of multigrid elliptic solvers based on multiple semicoarsened grids*, ICASE report 51-70, (1991).

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

[14] C. Zenger, *Sparse Grids*, Notes on Numerical Fluid Mechanics 31, W. Hackbusch, ed., Vieweg-Verlag, 1991.

[15] X. Zhang, *Multilevel Schwarz methods*, Numerische Mathematik, 63 (1992), pp. 521–539.