ELSEVIER

Applied Numerical Mathematics 18 (1995) 441–459

APPLIED NUMERICAL MATHEMATICS

Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers

E. de Sturler [a,1], H.A. van der Vorst [b,2]

[a] *Interdisciplinary Project Center for Supercomputing (IPS-ETHZ), Swiss Federal Institute of Technology, Clausiusstrasse 59, Zurich, Switzerland*

[b] *Mathematical Institute, Utrecht University, Budapestlaan 6, Utrecht, Netherlands*

Abstract

In this paper we study possibilities for the reduction of communication overhead introduced by inner products in the iterative solution methods CG and GMRES(m). The performance of these methods on massively parallel distributed memory machines is often limited because of the global communication required for the inner products. We investigate two ways of improvement. One is to assemble the results of a number of inner products collectively. The other is to create situations where communication can be overlapped with computation. The matrix–vector products may also introduce some communication overhead, but for many relevant problems this involves only communication with a few nearby processors that is easily overlapped as well. So this may, but does not necessarily, further degrade the performance of the algorithm.

Keywords: Parallel computing; Distributed memory computers; Conjugate gradient methods; Performance; GMRES, Modified Gram–Schmidt

1. Introduction

The Conjugate Gradients (CG) method [17] and the GMRES(m) method [23] are popular methods for the iterative solution of specific classes of linear systems. The time-consuming kernels in these methods are: inner products, vector updates, and matrix–vector products (including preconditioning). In many situations, especially when the matrix operations are well-structured, these operations are suited for implementation on vector computers and shared memory parallel computers [15].

For parallel distributed memory machines the picture may be entirely different. In general the matrix and the vectors are distributed over the processors, so that even when the matrix

[1] This author wishes to acknowledge Shell Research B.V. and STIPT for the financial support of his research.

[2] Corresponding author.

SSDI 0168-9274(95)00079-8

operations can be implemented efficiently by parallel operations, we cannot avoid the global communication required for inner product computations. These global communication costs become relatively more and more important when the number of parallel processors is increased and thus they have the potential to affect the scalability of the algorithms in a negative way [4,7,11]. This aspect has received much attention and several approaches have been suggested to improve the performance of these algorithms [1,4–8,11,20].

For CG the approaches come down to reformulating the orthogonalization part of the algorithm, so that the required inner products can be computed in the same phase of the iteration step (see, e.g., [8,20]), or to combine the orthogonalization for several successive iteration steps, as in the s-step methods [5]. The numerical stability of these approaches is a major point of concern.

For GMRES(m) the approach comes down to some variant of the s-step methods [1,6]. After having generated basis vectors for part of the Krylov subspaces by some suitable recurrence relation, they have to be orthogonalized. One often resorts to inexpensive but potentially unstable methods like Gram–Schmidt orthogonalization.

In the present study we investigate other ways to reduce the global communication overhead caused by inner products. Our approach is either to identify operations that may be executed while communication takes place, or to combine the messages for several inner products. For CG the first goal is achieved by rescheduling the operations, without changing the numerical stability of the method [10]. For GMRES(m) both goals are achieved by reformulating the modified Gram–Schmidt orthogonalization step [11,12].

We believe that our findings are relevant for other Krylov subspace methods as well, since methods like BiCG, and its variants CGS, BiCGSTAB, and QMR have much in common with CG from the implementation point of view. Likewise, the communication problems with GMRES(m) are representative for the problems in methods like ORTHODIR, GENCG, FOM, and ORTHOMIN.

We analyse the effectiveness of our algorithmic approaches by a simple communication and computation model (Section 3), but we have also done experiments on a 400-processor Parsytec Supercluster at the Koninklijke/Shell-Laboratorium in Amsterdam, in order to verify the validity of our theoretical expectations.

The processors of the Parsytec are connected in a fixed 20×20 mesh, of which arbitrary submeshes can be used. Each processor is a T800-20 transputer. The transputer supports only nearest neighbor synchronous communication; more complicated communication has to be programmed explicitly. The communication rate is fast compared to the flop rate, but to current standards the T800 is a slow processor. Another feature of the transputer is the support of time-shared execution of multiple "parallel" CPU-processes on a single processor, which facilitates the implementation of programs that switch between tasks when necessary (on interrupt basis), e.g., between communication and computation. Finally, transputers have the possibility of concurrent communication and computation. As a result it is possible to overlap computation and communication on a single processor.

The program that runs on each processor consists of two processes that run time-shared: a computation process and a communication process. The computation process functions as the master. If at some point communication is necessary, the computation process sends the data to the communication process (on the same processor) or requests the data from the communica-

tion process, which then handles the actual communication. This organization permits the computation processes on different processors to work asynchronously even though the actual communication is synchronous. The communication process is given the higher priority so that if there is something to communicate this is started as soon as possible. For a more elaborate description see [14].

2. Assumptions for the communication modeling

We will study the relative delaying effects of global communication for two algorithms, CG and GMRES(m), see Figs. 1 and 2, caused by the global communication for the inner products. Since the global communication is the same for each iteration step or iteration cycle, we are not interested in the number of iteration steps, but only in the relative effect *per step*. This means that we avoid discussions on the accelerating effects of preconditioning.

The most simple way to include preconditioning is to apply the given schemes with $K^{-1}A$ instead of A. In general effective preconditioners require additional global communication, and therefore our communication model should be viewed as an optimistic scenario. If it predicts that the effectiveness of more processors is limited, then this will even be more the case when preconditioning is included.

We will make the following assumptions:

- The processors are configured as a rectangular two-dimensional grid. Our model can be easily adapted to other configurations. For other processor configurations, e.g., hyper-

start:

x_0= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \beta_{-1} = 0$;
$\rho_0 = (r_0, r_0)$;

iterate:

for $i = 0, 1, 2, \ldots$ do
$p_i = r_i + \beta_{i-1} p_{i-1}$;
$q_i = Ap_i$;
$\alpha_i = \rho_i / (p_i, q_i)$;
$x_{i+1} = x_i + \alpha_i p_i$;
$r_{i+1} = r_i - \alpha_i q_i$;
compute $\|r\|$;
if accurate enough then quit;
$\rho_{i+1} = (r_{i+1}, r_{i+1})$;
$\beta_i = \rho_{i+1} / \rho_i$;
end

Fig. 1. The CG algorithm.

cubes, global communication may increase at a much lower rate when the number of processors increases, but also then the inner products are a source for concern [4].

- Because the results of the inner products are needed on all processors, the Hessenberg matrix $\bar{H}_m$ (see Fig. 2) is available on each processor. Hence, the computation of y_m can be done on each processor. This is often efficient, because it would be a synchronization point if implemented on a single processor, and then the other processors would have to wait for the result. However, if the size of the reduced system is large compared with the local number of unknowns, the computation might be expensive enough to make the distribution and parallel solution worthwhile. We have not pursued this idea.
- The communication overhead in the matrix–vector product can be kept low by a good domain decomposition, which minimizes the amount of data to be communicated. In recent years, several algorithms have become available that compute good partitionings in a reasonable time [2,16,18,22,24]; some of these algorithms have also been parallelized [19], and research on minimizing the number of messages and the distance they have to travel has also been done [4,9]. Further reduction of the communication cost can be achieved by overlapping the communication with computation. So even in the general case (for three-dimensional, irregular finite element or finite volume discretization) the communication overhead of the matrix–vector product is relatively small. Therefore, we neglect the communication effect for the matrix–vector product in our model. Furthermore, we will

start:
x_0= initial guess; $r_0 = b - Ax_0$;
$v_1 = r_0/\|r_0\|_2$;

iterate:
for $j = 1, 2, \ldots, m$ do
$\hat{v}_{j+1} = Av_j$;
for $i = 1, 2, \ldots, j$ do
$h_{i,j} = (\hat{v}_{j+1}, v_i)$;
$\hat{v}_{j+1} = \hat{v}_{j+1} - h_{i,j}v_i$;
end;
$h_{j+1,j} = \|\hat{v}_{j+1}\|_2$;
$v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$;
end
form the approximate solution:
$x_m = x_0 + V_m y_m$; where
y_m minimizes $\| \|r_0\|_2 e_1 - \bar{H}_m y\|_2$, $y \in \mathbb{R}^m$

restart:
compute $r_m = b - Ax_m$; if satisfied then
stop, else $x_0 = x_m$; $v_1 = r_m/\|r_m\|_2$;
goto *iterate*

Fig. 2. The GMRES(m) algorithm.

use model problems derived from two-dimensional regular grids for which a good decomposition is easy to find.

- The iteration schemes may be applied with preconditioning, that is A may be replaced by $K^{-1}A$. Of course, the choice of the preconditioner greatly influences the number of iteration steps, but we are only interested in the relative delaying effect of global communication *per step*. We assume that a preconditioner is available which does not need global communication. This may be unrealistic, but if the preconditioner requires significant global communication then this only adds to our arguments of the importance of global communication, and communication cost reducing techniques should be used there as well. Moreover, recently for many relevant problems preconditioners or (Schwarz) relaxation schemes that can be used as preconditioners have been developed that need only local communication over subdomain boundaries and have very good convergence properties nevertheless [13,14,21,26–28].
- Since the vectors are distributed over the processor grid, the inner product (ddot) is computed in two steps. All processors start to compute in parallel the local inner product. After that, the local inner products are accumulated on one "central" processor and broadcasted. The communication time of an accumulation or a broadcast is inherently of the order of the diameter ($\equiv 2p_d$) [3] of the processor grid, because the transputer does not support global communication. The accumulation and broadcast algorithm is optimal in this respect (see [14]). This means that for an increasing number of processors the communication time for the inner products increases as well, and hence this is a potential threat to the scalability of the method. Each processor is active at only a few steps in the accumulation and broadcasting process, and hence there is a potential to do other useful work meanwhile. Indeed, if the global communication for the inner products is not overlapped, it often becomes a bottleneck on large processor grids, as will be shown later.

We will describe a simple performance model based on these considerations which clearly shows quantitatively the dramatic influence of the global communication for inner products over large processor grids on the performance of Krylov subspace methods. We selected the problem size for our experiments such as to illustrate the scaling behaviour of Krylov subspace methods for the standard implementations and for our versions. The slow computational speed of the T800 transputer and its fast communication lead to a small problem size. However, results analogous to those presented in Sections 5 and 7 will be seen for much larger problems on new processor configuration(s) with much faster computational speed and often slower global communication. This is the current trend in parallel computers. [4] Moreover, if the

[3] We accumulate on and broadcast from the "most central" processor.

[4] For example, on the 96-processor Intel Paragon at ETH Zurich the global sum operation (using the native **gdsum** routine) is approximately four times slower than our global sum implementation on the 400-processor Parsytec. On the other hand, we obtain about 120 times higher flop rates per processor on the Paragon. This means that we see about the same behaviour on the Paragon with GMRES(50) for a problem size of about three to five million unknowns as for the Parsytec with GMRES(50) for a problem size of ten thousand unknowns. Five million unknowns constitutes a large problem in the sense that GMRES(50) cannot be used due to memory limitations even on 96 processors (the Intel Paragon at ETH Zurich provides about 26MB user space per node).

problem sizes increase proportionally to the number of processors, the local computation time remains the same but the global communication cost increases. This is nicely illustrated by experiments and observations reported in [7], where it has been concluded that for an increasing number of processors the communication time will dominate eventually, even if the problem size grows linearly with the number of processors.

3. Parallel performance of GMRES(m) and CG

We will describe a simple model for the computation time and the communication cost for the main kernels in Krylov subspace methods; for a more elaborate description and analysis of the performance of Krylov subspace methods we refer to [14].

We use the term *communication cost* to indicate all the wall clock time spent in communication that is not overlapped with useful computation (so that it really contributes to wall-clock time). The term *communication time* refers to the wall-clock time of the whole communication. In the case of a nonoverlapped communication, the communication time and the communication cost are the same.

Remark. Our quantitative formulas are not meant to give very accurate predictions of the exact execution times, but they will be used to identify the bottlenecks and to evaluate improvements. Indeed, our experiments show that our models are relatively close to reality, and thus may be used as convenient tools for understanding the negative effects of global communication.

3.1. Computation time

The computation time for the solution of the Hessenberg system is neglected in our model. For a vector update (daxpy) or an inner product (ddot) the computation time is given by $2t_{\text{fl}}N/P$, where N/P is the local number of unknowns of a processor and t_{fl} is the average time for a double precision floating point operation. The computation time for the (sparse) matrix–vector product is given by $(2n_z - 1)t_{\text{fl}}\, N/P$, where n_z is the average number of nonzero elements per row of the matrix. We assume that the preconditioner takes about the same computational work, and for convenience we take the total time for these two matrix–vector operations as $4n_z t_{\text{fl}}N/P$.

A full GMRES(m) cycle requires approximately $\frac{1}{2}(m^2 + 3m)$ inner products, the same number of vector updates, and $(m + 1)$ multiplications with the matrix and the preconditioner (if one computes the exact residual at the end of each cycle). The complete (local) computation time for the GMRES(m) algorithm is given by the equation:

$$T_{\text{comp}}^{\text{G}} = \left(2(m^2 + 3m) + 4n_z(m + 1)\right)\frac{N}{P}t_{\text{fl}}. \tag{1}$$

A single iteration of CG requires three inner products (if one computes also the norm of the residual for the unpreconditioned system), the same number of vector updates and one

multiplication with the matrix and the preconditioner. The complete (local) computation time is given by the equation:

$$T_{\text{comp}}^{\text{C}} = (12 + 4n_z)\frac{N}{P}t_{\text{fl}}. \tag{2}$$

3.2. Communication cost

For a processor grid with $P = p^2$ processors, we have that $p_{\text{d}} = \sqrt{P}$. For nearest neighbor communication, let t_{s} denote the communication start-up time and let the transmission time associated with a single inner product computation be t_{w}. Then the global accumulation and broadcast time for one inner product is taken as $2p_{\text{d}}(t_{\text{s}} + t_{\text{w}})$, while the global accumulation and broadcast for k simultaneous inner products takes $2p_{\text{d}}(t_{\text{s}} + kt_{\text{w}})$.

For GMRES(m) the communication time for the modified Gram–Schmidt algorithm (with $\frac{1}{2}(m^2 + 3m)$ accumulations and broadcasts) is

$$T_{\text{comm}}^{\text{G}} = (m^2 + 3m)p_{\text{d}}(t_{\text{s}} + t_{\text{w}}). \tag{3}$$

For CG the communication time for the three inner products per iteration, is

$$T_{\text{comm}}^{\text{C}} = 6p_{\text{d}}(t_{\text{s}} + t_{\text{w}}). \tag{4}$$

Note that for other processor configurations one only needs to bring in an appropriate value for p_{d} in (3) and (4).

4. Communication overhead reduction: parGMRES(m)

4.1. Combination of messages

From (3) we conclude that the communication cost for GMRES(m) is of the order $O(m^2\sqrt{P})$ and for large processor grids this may become a bottleneck. Moreover, in the standard implementation we cannot reduce these costs by accumulating multiple inner products together or overlap this expensive communication with computation, since the modified Gram–Schmidt orthogonalization of one single vector against a set of vectors and its subsequent normalization is an inherently sequential process.

However, if the modified Gram–Schmidt orthogonalization of a set of vectors can be done, then we have more possibilities for reduction of communication overhead. We discuss how to generate such a set of vectors in Section 4.3.

```
for i = 1, 2, ..., m do
    orthogonalize v̂_{i+1}, ..., v̂_{m+1} on v_i;
    v_{i+1} = v̂_{i+1}/‖v̂_{i+1}‖_2;
end
```

Fig. 3. A block-wise modified Gram–Schmidt orthogonalization.

Suppose the set of vectors $v_1, \hat{v}_2, \hat{v}_3, \ldots, \hat{v}_{m+1}$ has to be orthogonalized, where $\| v_1 \|_2 = 1$. The modified Gram–Schmidt process can be implemented as sketched in Fig. 3. This reduces the number of messages to only m instead of $\frac{1}{2}(m^2 + 3m)$ for the usual implementation of GMRES(m), but the length of the messages has increased. In this way, start-up time is saved by packing small messages, corresponding to one block of orthogonalizations, into one larger message.

4.2. Possibilities for overlap

If we wish to create opportunities for overlap of communication time with computation time, we may proceed as follows. Instead of computing all local inner products in one block and accumulating these partial results only once for the whole block, we split each step into two blocks of orthogonalizations. The overlap is achieved by performing the accumulation and broadcast of the local inner products of the first block concurrently with the computation of the local inner products of the second block and performing the accumulation and broadcast of the local inner products of the second block concurrently with the vector updates of the first block, see Fig. 4.

Note that the computation time for this approach is equal to that for the standard modified Gram–Schmidt algorithm. We will call this variant of GMRES: *parGMRES*(m).

If we assume that sufficient computational work can be done to completely fill the communication time, then the communication cost $T^{\mathrm{G}}_{\mathrm{comm}}$, see (3), can be almost neglected. In fact we will see mainly communication cost for neighbor–neighbor connections, like local start-up times, that cannot be overlapped. Hence, in this case the communication cost is $O(m)$, instead of $O(m^2\sqrt{P})$ as for the nonoverlapped situation.

```
for i = 1,2,...,m do
    split v̂_{i+1},...,v̂_{m+1} into two blocks
    compute local inner products (LIPs) block_1

    ||  { accumulate LIPs block_1
        { compute LIPs block_2

    update v̂_{i+1}, compute LIP for ||v̂_{i+1}||_2,
    place this LIP into block_2

    ||  { accumulate LIPs block_2
        { update vectors block_1

    update vectors block_2
    normalize v̂_{i+1}
end
```

Fig. 4. The implementation of the modified Gram–Schmidt process.

In many situations we will not have sufficient work in the inner products for overlapping all communication time. In that case the wall-clock time for the inner products is determined by the global communication time

$$T_{\text{comm}}^{\text{pG}} = 4mp_{\text{d}}t_{\text{s}} + (m^2 + 3m)p_{\text{d}}t_{\text{w}}. \tag{5}$$

Some remarks can be made.

First, if we assume that sufficient computational work is available, then the contribution of start-up times to the communication cost is reduced from $O(m^2\sqrt{P})$ with standard GMRES(m) to $O(m)$ when using the parallel modified Gram–Schmidt algorithm. Especially for machines with relatively high start-up times this is important. In fact, if the start-ups dominate the communication cost, then we can reduce this contribution again by a factor of two with the algorithm given in Fig. 3 (even if we neglect the overlap).

Second, still assuming sufficient computational work, the communication cost no longer depends on the size of the processor grid; instead of being of the order of the diameter of the processor grid, it is then more or less constant.

Finally, note that in order to achieve a noticeable reduction in the communication costs for GMRES(m), the value of m should not be small. For small values of m, say $m = 2$, we will see pretty much the same situation as for CG. For any practical value of m we will see some gain, but the gain will be larger for larger values of m.

4.3. Creation of the basis before orthogonalization

In order to be able to use this parallel modified Gram–Schmidt algorithm in GMRES(m), a basis for the Krylov subspace has to be generated first. The idea to generate some basis for the Krylov subspace first, and then to orthogonalize this basis afterwards, was already suggested for the CG algorithm, referred to as s-step CG, in [5] for shared (hierarchical) memory parallel vector processors. In [5] it is also reported that the s-step CG algorithm may converge slowly due to numerical instability for $s > 5$.

In the parGMRES(m) algorithm stability seems to be much less of a problem since each vector is explicitly orthogonalized against all the other vectors, and we first generate a polynomial basis for the Krylov subspace, see [1,12]. Furthermore, because of the restart, rounding errors made before restart can not have an accumulated effect on iterations carried out after the restart. The basis vectors for the Krylov subspace $\hat{v}_i$ are generated as indicated in Fig. 5, where the parameters d_i are chosen to get the condition number of the matrix $[v_1, \hat{v}_2, \ldots, \hat{v}_{m+1}]$ sufficiently small. In order to achieve this one might start in the first cycle

$\hat{v}_1 = v_1 = r/\|r\|_2$
for $i = 1, 2, \ldots, m$ do
  $\hat{v}_{i+1} = \hat{v}_i - d_i A\hat{v}_i$
end

Fig. 5. Generation of a polynomial basis for the Krylov subspace.

with a suitable Chebychev recursion, for later cycles one may then take into account the GMRES information of the previous cycle. For more details on this, see [1] and [25, Section 6].

In the next section we restrict our considerations to one single GMRES(m) cycle.

The parallel computation of the Krylov subspace basis requires m extra daxpys, but we will neglect these additional costs in our discussions.

4.4. Reconstruction of the Hessenberg matrix

Because we generate the Krylov subspace basis before orthogonalization, the Hessenberg matrix that we obtain from the inner products is not $V_{m+1}^{\mathrm{T}} A V_m$, as in the standard GMRES(m) algorithm, and therefore we need to solve the least squares problem in a slightly different way. Define $\hat{v}_1 = v_1 = \| r \|_2^{-1} r$, and generate the other basis vectors as $\hat{v}_{i+1} = (I - d_i A)\hat{v}_i$, for $i = 1, \ldots, m$. This gives the following relation:

$$\left[\hat{v}_2 \ \hat{v}_3 \ \ldots \ \hat{v}_{m+1}\right] = \hat{V}_m - A\hat{V}_m D_m, \tag{6}$$

where $D_m = \mathrm{diag}(d_i)$ and $\hat{V}_m$ is the matrix with the vectors $\hat{v}_i$ as its columns. This relation between vectors and matrices composed from these vectors will be used throughout this discussion.

The parallel modified Gram–Schmidt orthogonalization gives the orthogonal set of vectors $\{v_1, \ldots, v_{m+1}\}$, for which we have

$$v_{j+1} = h_{j+1,j+1}^{-1}\left(\hat{v}_{j+1} - \sum_{i=1}^{j} h_{i,j+1} v_i\right), \quad \text{for } i = 1, \ldots, m, \tag{7}$$

where $h_{i,j}$ is defined by (but computed differently)

$$h_{i,j} = \begin{cases} (v_i, \hat{v}_j), & i \leqslant j, \\ 0, & i > j. \end{cases} \tag{8}$$

Notice the subtle difference with the definition of $\overline{H}_m$ in the standard implementation of GMRES(m). Here the matrix H_{m+1} is upper triangular. Furthermore, as long as $h_{i,i} \neq 0$ the matrix H_i is nonsingular, whereas $h_{i,i} = 0$ indicates a lucky breakdown.

We will further assume, without loss of generality, that $h_{i,i} \neq 0$, for $i = 1, \ldots, m+1$. Let h_i denote the ith column of H_{m+1}. From Eqs. (7) and (8) it follows that

$$\hat{V}_i = V_i H_i, \quad \text{for } i = 1, \ldots, m+1. \tag{9}$$

Eq. (6) can be rewritten as

$$\hat{V}_m = \left[\hat{v}_2 \ \ldots \ \hat{v}_{m+1}\right] = A\hat{V}_m D_m = A V_m H_m D_m. \tag{10}$$

Define $\hat{H}_m = [h_1 \ h_2 \ldots h_m] - [h_2 \ h_3 \ldots h_{m+1}]$, so that $\hat{H}_m$ is an upper Hessenberg matrix of rank m, since $h_{i,i} \neq 0$, for $i = 1, \ldots, m+1$. Substituting this in (10) finally leads to

$$V_{m+1}\hat{H}_m = A V_m H_m D_m. \tag{11}$$

Using this expression the least squares problem can be solved in the same way as for standard GMRES(m):

$$\min_y \| r - AV_m y \|_2 = \min_{\hat{y}} \| r - AV_m H_m D_m \hat{y} \|_2, \quad \text{where } H_m D_m \hat{y} = y. \tag{12}$$

Because H_m and D_m are nonsingular, the latter by definition, $H_m D_m \hat{y} = y$ is always well-defined. Combining (11) and (12) yields

$$\hat{y} : \min_{\tilde{y}} \left\| r - V_{m+1} \hat{H}_m \tilde{y} \right\|_2 = \min_{\tilde{y}} \left\| \| r \|_2 e_1 - \hat{H}_m \tilde{y} \right\|_2. \tag{13}$$

The additional computational work in this approach is only $O(m^2)$ and therefore negligible.

5. Performance of GMRES(m) and parGMRES(m)

5.1. Theoretical discussion

Before we discuss our experiments, we present a short theoretical analysis. For a more extensive discussion we refer to [11,14].

The runtime for a GMRES(m) cycle on $P \geqslant 4$ processors is given by $T_P = T^{\mathrm{G}}_{\mathrm{comp}} + T^{\mathrm{G}}_{\mathrm{comm}}$, see (1) and (3):

$$T_P = \big(2(m^2 + 3m) + 4n_z(m+1)\big) t_{\mathrm{fl}} \frac{N}{P} + \big((m^2 + 3m)(t_s + t_w)\big)\sqrt{P}\,. \tag{14}$$

This equation shows that for each m and for sufficiently large P the communication will dominate.

We introduce the value $P_{\max}$ as the number of processors that minimizes the runtime for GMRES(m). Minimization of (14) gives

$$P_{\max} = \left(\frac{\big[4(m^2 + 3m) + 8n_z(m+1)\big] t_{\mathrm{fl}}}{(m^2 + 3m)(t_s + t_w)} N \right)^{2/3}. \tag{15}$$

The efficiency $E_P = T_1/(PT_P)$ for $P_{\max}$ processors is $E_{P_{\max}} = \frac{1}{3}$, where $T_1 = T^{\mathrm{G}}_{\mathrm{comp}}$. This means that $\frac{2}{3} T_{P_{\max}}$ is spent in communication, because in this model efficiency is lost only through communication.

Note that for parGMRES(m) we can improve the performance due to created overlap possibilities and reduced start-up times. For parGMRES(m) we define P_{ovl} as the number of processors for which all (overlappable) communication can just be overlapped with computation. For P_{ovl} we have that the (total) communication time $T^{\mathrm{pG}}_{\mathrm{comm}}$ minus some nonoverlappable part is equal to the overlapping computation time $(m^2 + 3m) t_{\mathrm{fl}} N/P$ (see [14]):

$$\big(4mt_s + (m^2 + 3m)t_w\big)\big(\sqrt{P_{\mathrm{ovl}}} - 4\big) = (m^2 + 3m) t_{\mathrm{fl}} \frac{N}{P_{\mathrm{ovl}}}. \tag{16}$$

If $P \leqslant P_{\text{ovl}}$, then there is no significant communication visible.
If $P > P_{\text{ovl}}$, then the overlap is no longer complete and the communication cost is given by

$$T_{\text{comm}}^{\text{pG}} - (m^2 + 3m)t_{\text{fl}}\frac{N}{P}.$$

The runtime is then determined by $T_{\text{comp}}^{\text{G}}$ plus the communication cost, which gives

$$\tilde{T}_P = \big((m^2 + 3m) + 4n_z(m+1)\big)t_{\text{fl}}\frac{N}{P} + \big(4mt_{\text{s}} + (m^2 + 3m)t_{\text{w}}\big)\sqrt{P}\,. \tag{17}$$

For $P > P_{\text{ovl}}$ we see that the efficiency decreases again, because the communication time increases and the computation time in the overlap decreases.

From (15) and (16), we see that if t_{s} dominates the communication, that is $t_{\text{s}} \gg t_{\text{w}}$, then $P_{\text{ovl}} > P_{\text{max}}$ and for $P \leqslant P_{\text{max}}$ we can overlap all communication after the reduction of start-ups. Since with P_{max} we lost two-third of the time in communication, this means that we can reduce the runtime by almost a factor of three.

When $P_{\text{ovl}} < P_{\text{max}}$, then for $P = P_{\text{max}}$ the runtime is given by (17). When $t_{\text{s}} \approx t_{\text{w}}$ and m is large enough we get $\tilde{T}_P \approx \frac{1}{2}T_P$, so that the time is reduced by a factor of almost two.

5.2. Experimental observations

We now discuss our experimental observations on the parallel performance of GMRES(m) and parGMRES(m) on the Parsytec. We will only consider the performance of one (par-) GMRES(m) cycle.

We have solved a convection–diffusion problem discretized by finite volumes over a 100×100 grid, resulting in the standard five-diagonal matrix with a tridiagonal block-structure, corresponding to the 5-point star. This relatively small problem size was chosen, because for the processor grids of increasing size it very well shows the expected degradation of performance for GMRES(m) and the large improvements of parGMRES(m) over GMRES(m). Furthermore, the parallel behavior for this problem size on this slow machine corresponds quite well to much larger problems on more modern machines (see footnote 4). The discretization gives $n_z = 5$; for the other parameters we have approximately $t_{\text{s}} = 5$ μs, $t_{\text{w}} = 15$ μs, and $t_{\text{fl}} = 3$ μs.

The measured runtimes for a single (par) GMRES(m) cycle are listed in Table 1 for $m = 30$ and $m = 50$. For $m = 30$ we have that $P_{\text{max}} \approx 400$ and $P_{\text{ovl}} \approx 200$. For $m = 50$ we have $P_{\text{max}} \approx 375$ and $P_{\text{ovl}} \approx 200$. We give speed-ups and efficiencies in Table 2. Since the problem did

Table 1
measured runtimes for GMRES(m) and parGMRES(m)

Processor grid	$m = 30$		$m = 50$	
	GMRES(m) (s)	parGMRES(m) (s)	GMRES(m) (s)	parGMRES(m) (s)
10×10	1.01	0.813	2.47	1.93
14×14	0.738	0.448	1.90	1.05
17×17	0.667	0.389	1.66	0.891
20×20	0.682	0.365	1.75	0.851

Table 2
Efficiencies and speed-ups for GMRES(m) and parGMRES(m)

Processor grid	$m = 30$				$m = 50$			
	GMRES(m)		parGMRES(m)		GMRES(m)		parGMRES(m)	
	E (%)	S	E (%)	S	E (%)	S	E (%)	S
10×10	77.2	77.2	95.9	95.9	76.8	76.8	98.2	98.2
14×14	53.9	106.	88.8	174.	50.9	99.8	92.1	181.
17×17	40.5	117.	69.4	201.	39.5	114.	73.6	213.
20×20	28.6	114.	53.4	214.	27.1	108.	55.7	223.

not fit on a single processor, the speed-up and efficiency values were computed against a (fairly accurate) estimated sequential runtime. For our purposes this is not really important since we are only interested in relative speed-ups and efficiencies. We are interested in the speed-up of parGMRES(m) compared with GMRES(m), and in the speed-up and efficiency of parGMRES(m) on 100 processors compared with the speed-up and efficiency of parGMRES(m) on $196, \ldots, 400$ processors and the same for GMRES(m). For these fractions, of course, the actual runtime of the sequential program plays no role.

The runtime for GMRES(m) is reduced by approximately 25%, when increasing the number of processors from 100 to 196. When increasing this from 100 to 289, the runtime reduces only by some 35%. When we further increase the number of processors to 400 then the runtime is already (slightly) more than for 289 processors, which is in agreement with the previous discussion because $P \approx P_{\max}$ for $m = 30$ and $P > P_{\max}$ for $m = 50$. We clearly see that the communication spoils the performance of GMRES(m) completely for large P.

On the other hand, for parGMRES(m) the runtime reduction when increasing from 100 to 196 processors is approximately 45%, where the upper bound is $100/196 \times 100\% = 49\%$, so this is almost optimal (even for this small system of equations). Such a speed-up shows that the efficiency remains almost constant for this increase in the number of processors, see also Table 2. This is to be expected because we have $P < P_{\text{ovl}}$, so that any increase in the communication time of the inner products is completely annihilated by the overlapping computation.

On 289 processors the runtime is about 53% of the runtime on 100 processors, which is still quite good. There is some degradation of the efficiency, which is explained by the fact that $P > P_{\text{ovl}}$. If we continue to increase the number of processors, we see that for 400 processors the runtime is not much better than for 289 processors, although it is still decreasing. At this point the speed-up for parGMRES(m) levels off, because there is insufficient computational work for overlapping ($P > P_{\text{ovl}}$).

A direct comparison between the runtimes of GMRES(m) and parGMRES(m) shows that, for 100 processors, GMRES(m) is about 25% slower than parGMRES(m). However, for 196 processors, this has increased already to 65% and 81% for $m = 30$ and $m = 50$, respectively. From then on the relative difference increases more gradually to a maximum of about a factor of two for $P_{\max}$ processors. These results are very much in agreement with our theoretical expectations, so we may say that at least qualitatively the model is good. Note that although the maximum is reached for $P_{\max}$ the improvement is already substantial for 196 processors, which is close to P_{ovl}. As a final comment on the performance we mention that in [3] a maximum

$x_{-1} = x_0$= initial guess; $r_0 = b - Ax_0$;
$p_{-1} = 0; \alpha_{-1} = 0$;
$s = L^{-1}r_0$;
$\rho_{-1} = 1$;
for $i = 0, 1, 2, \ldots$ do
(1) $\rho_i = (s, s)$;
$w_i = L^{-T}s$;
$\beta_{i-1} = \rho_i/\rho_{i-1}$;
$p_i = w_i + \beta_{i-1}p_{i-1}$;
$q_i = Ap_i$;
(2) $\gamma = (p_i, q_i)$;
$x_i = x_{i-1} + \alpha_{i-1}p_{i-1}$;
$\alpha_i = \rho_i/\gamma$;
$r_{i+1} = r_i - \alpha_i q_i$;
(3) compute $\|r\|$;
$s = L^{-1}r_{i+1}$;
if accurate enough then
$x_{i+1} = x_i + \alpha_i p_i$;
quit
end;

Fig. 6. The parCG algorithm.

speed-up is reported of 93 on the same 400 processor Parsytec for GMRES(100) for a problem with 9881 unknowns and an average of approximately eight nonzero coefficients per row of the matrix. [5]

6. Communication overhead reduction in CG

For a reduction in the communication overhead for preconditioned CG we follow the approach suggested in [10]. In that approach the operations are rescheduled to create more opportunities for overlap. This leads to an algorithm (parCG) as the one given in Fig. 6, where we have assumed that the preconditioner K can be written as $K = LL^T$. Note that this variant involves only a rescheduling of the operations and is numerically just as stable as the original preconditioned CG method (in fact, the suggested way for the computation of ρ in Fig. 6 guarantees the positiveness of ρ, in contrast to the standard procedure; this advantage may be of limited practical value however).

For our purposes it is relevant to point at the inner products at lines (1), (2) and (3). The communication for these inner products can be overlapped by the computational work in the subsequent line. We split the preconditioner to create overlap for the inner products (1) and

[5] In [3] the speed-up was computed relative to the runtime of the parallel program on one processor.

(3), and we have also overlap possibilities since the inner product (2) is followed by the update for x corresponding to the previous iteration step.

Under the assumption of a complete overlap for the communication time, similar as for GMRES(m), the communication cost for the three inner products in a parCG iteration vanishes almost completely. Therefore, the communication cost is reduced from $O(\sqrt{P})$ to O(1) (for some neighbor–neighbor communication).

7. Performance of CG variants

7.1. Theoretical discussion

We will follow the same way of reasoning as in Section 5; for a more elaborate discussion see [11,14].

The runtime for a CG iteration with $P \geqslant 4$ processors is given by $T_P = T_{\text{comp}}^{\text{C}} + T_{\text{comm}}^{\text{C}}$, (cf. (2) and (4)):

$$T_P = (12 + 4n_z)t_{\text{fl}}\frac{N}{P} + 6(t_s + t_w)\sqrt{P}\,. \tag{18}$$

This expression shows that for sufficiently large P the communication time will dominate. Also for CG we define $P_{\max}$ as the number of processors that gives the minimal runtime, and P_{ovl} as the number of processors for which the (overlappable) communication time in the inner products ($T_{\text{comm}}^{\text{C}}$) is just equal to the sum of the computation time of the preconditioner and one vector update ($(2 + 2n_z)t_{\text{fl}}N/P$). This leads to

$$P_{\max} = \left(\frac{(24 + 8n_z)Nt_{\text{fl}}}{6(t_s + t_w)}\right)^{2/3}. \tag{19}$$

For $P = P_{\max}$ processors the efficiency $E_P = T_1/(PT_P)$ is again $E_{P_{\max}} = \frac{1}{3}$, where $T_1 = T_{\text{comp}}^{\text{C}}$; therefore, the communication time is then $\frac{2}{3}T_{P_{\max}}$.

The value for P_{ovl} follows from (see [14])

$$6(t_s + t_w)\left(\sqrt{P_{\text{ovl}}} - 4\right) = (2 + 2n_z)t_{\text{fl}}\frac{N}{P_{\text{ovl}}}. \tag{20}$$

For $P \leqslant P_{\text{ovl}}$ the communication cost is almost annihilated, which gives a reduction by a factor of $O(\sqrt{P})$. For $P > P_{\text{ovl}}$ the communication cost is given by $T_{\text{comm}}^{\text{C}} - (2 + 2n_z)t_{\text{fl}}N/P$.

A comparison of (19) and (20) shows that $P_{\text{ovl}} < P_{\max}$. Even though the preconditioner is strongly problem- and implementation-dependent, this holds in general, because for $P = P_{\text{ovl}}$ the communication time is equal to a part of the computation time, whereas for $P = P_{\max}$ the communication time is already twice the computation time. This leads to three phases in the performance of parCG. Let a be the computation time, and for $\beta \in [0, 1]$ let βa be the computation time for the "potential" overlap, and let c be the communication time. Then the runtime of CG is given by $a + c$, whereas for parCG it is given by $(1 - \beta)a + \max(\beta a, c)$. For increasing P, a decreases and c increases as described above. For small P, $c \ll \beta a \Leftrightarrow P \ll P_{\text{ovl}}$,

all communication can be overlapped but the communication time is relatively unimportant. For medium P, $c \approx \beta a \Leftrightarrow P \approx P_{ovl}$, the communication time is more or less in balance with the computation time for the overlap and the improvement is maximal, see below. For large P, $c \gg \beta a \Leftrightarrow P \gg P_{ovl}$, the communication time will be dominant, and then we will not have enough computational work to overlap it sufficiently. It is easy to prove that the fraction

$$(a+c)/((1-\beta)a + \max(\beta a, c))$$

is maximal if $\beta a = c$, that is for $P = P_{ovl}$, and then the improvement is

$$\frac{a+c}{(1-\beta)a + \max(\beta a, c)} = \frac{a+\beta a}{a} = 1+\beta. \tag{21}$$

Hence, the maximum improvement of parCG over CG is determined by this fraction β. The larger this fraction is, the larger is the maximum improvement by parCG. If the computation time of the preconditioner is dominant, e.g. when n_z is large and when we use preconditioners from a factorization with fill-in, then $\beta \approx 1$, and we can expect an improvement by a factor of two. In our model we have $\beta = 2n_z/(9+4n_z)$, so that for n_z large enough we can expect a reduction by a factor of 1.5. For our model problem we have $n_z = 5$, so that the improvement is limited to factor of 1.33.

7.2. Experimental observations

We will now discuss the results for the parallel implementation of the standard CG algorithm and parCG on the Parsytec for a model problem. Again, since we are only interested in the delaying effects relative to the computational time, we will only consider the runtime for one single iteration.

We have solved a diffusion problem discretized by finite volumes over a 100×100 grid, resulting in a symmetric positive-definite five-diagonal matrix (corresponding to the 5-point star). This problem size displays, for processor grids of increasing size, nicely the three different phases mentioned before. The values of the parameters t_s, etc. are the same as in Section 5.2.

Table 3 gives the measured runtimes for one iteration step, the speed-ups, and the efficiencies for both CG and parCG for several processor grids. The speed-ups and efficiencies are computed relative to the runtime of one (really) sequential CG iteration. Although CG has much fewer inner products than GMRES(m) per iteration (or rather, per matrix–vector

Table 3
Measured runtimes for CG and parCG, speed-up and efficiency

Processor grid	CG			parCG			Diff (%)
	T_P (ms)	S_P	E_P (%)	T_P (ms)	S_P	E_P (%)	
10×10	10.7	73.6	73.6	10.2	77.3	77.3	4.90
14×14	6.90	114.	58.3	5.84	135.	68.8	18.2
17×17	6.09	129.	44.8	5.29	149.	51.5	15.1
20×20	5.59	141.	35.2	5.04	156.	39.1	10.9

product), we observe that the performance levels off fairly quickly. This is in agreement with observations made in [7,11].

For our test problem we have that $P_{max} \approx 625$ and $P_{ovl} \approx 250$.

For the processor grids that we used we have $P < P_{max}$, so that the runtime decreases for increasing numbers of processors as predicted by our analysis. Note also the large relative difference between P_{ovl} and P_{max} compared to the relative small difference for GMRES(m). This indicates that for this test problem, with a small n_z and with a relatively cheap preconditioner, we have a small β. Hence, the improvement in the runtime will be limited, as is illustrated in Table 3.

We see that the parCG algorithm leads to better speed-ups than the standard CG algorithm, especially on the 14×14 and 17×17 processor grids, where the number of processors is closest to P_{ovl}. Moreover, for parCG we observe that if the number of processors is increased from 100 to 196, the efficiency remains almost constant, and the runtime is reduced by a factor of about 1.75 (against a maximum of 1.96). Just as for GMRES(m) this is predicted by our analysis, because $P < P_{ovl}$, so that the increase in the communication time is masked by the overlapping computation.

The initial decrease of efficiency when going from 1 to 100 processors is due to a substantial initial overhead, which is mainly caused by the explicit implementation of the (global) communication. The three phases in the performance of parCG are illustrated by the difference in runtime between CG and parCG. For small processor grids the communication time is not very important and we see only small differences. For processor grids with P near P_{ovl} the communication and the overlapping computation are in balance and we see an increase in the runtime difference. For larger processor grids we can no longer overlap the communication, which dominates the runtime, to a sufficient degree, and we see the differences decrease again.

We cannot quite match the improvements for parGMRES(m), but on the other hand it is important to note that the improvement for parCG comes virtually for free. Besides, for GMRES(m) we have the possibility to combine messages as well as to overlap communication, whereas for CG we can only exploit overlap of communication unless we combine multiple iterations. Expression (21) indicates that for our problem we cannot expect much more: $\beta \approx \frac{1}{3}$ so that the maximum improvement is approximately 33%. This estimate is rather optimistic in view of the large initial parallel overhead. When the computation time for the preconditioner is large or even dominant ($\beta \approx 1$) then the improvement may also be large. This would be the case if n_z is large or when (M)ILU preconditioners with fill-in are used. For many problems this may be a realistic assumption.

Another observation is that as long as $P > P_{ovl}$, we can increase the computation time of the preconditioner without increasing the runtime of the iteration, because the preconditioner is overlapped with the accumulation and distribution. That means that we can potentially decrease the number of iterations without increasing the runtime of an iteration.

8. Conclusions

We have studied the effects of global communication for the inner products on the parallel performance of GMRES(m) and CG on distributed memory parallel computers. These algo-

rithms represent two different classes of Krylov subspace methods, and their parallel properties are quite representative. The experiments show how the global communication in the inner products degrades the performance on large processor grids, as is indicated by our model in Section 3 and the discussions in Sections 5 and 7. We have considered alternative algorithms for GMRES(m) and CG in which the actual cost for global communication is decreased by reducing synchronization, reducing start-up times, and overlapping communication with computation.

Our experiments clearly indicate this to be a successful approach. For GMRES(m) we have reduced the communication cost by reducing the contribution of start-ups from $O(m^2\sqrt{P})$ to $O(m)$. This results in a total reduction of the runtime by about a factor of two. For CG we can only overlap the communication. In theory this may reduce the communication cost by a factor of $O(\sqrt{P})$, but for our model problem we cannot achieve this performance improvement. The (maximum) improvement for parCG over CG depends mainly on the computation time for the preconditioner relative to the total computation time. For problems in which the cost of the preconditioner is more dominant, that is for a large average number of nonzero coefficients per row in the matrix and fill-in in the preconditioner, better results may be expected and such a situation is not unrealistic. Moreover, in the case that we cannot overlap all communication in parCG, we can use a more expensive preconditioner that may reduce the number of iterations without increasing the runtime for a single iteration. If we want to further improve the performance of CG, then polynomial preconditioners or methods which optimize over a larger subspace per step might be considered.

Although illustrated and supported only by a model problem of moderate size, our analysis (which seems to give relatively accurate predictions) indicates that also for other problem sizes and processor grids our conclusions remain valid.

References

[1] Z. Bai, D. Hu and L. Reichel, A Newton basis GMRES implementation, Technical Report 91-03, University of Kentucky (1991).

[2] S.T. Barnard and H.D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, Technical Report RNR-92-033, NASA Ames Research Center, Moffet Field, CA (1992).

[3] R.H. Bisseling, Parallel iterative solution of sparse linear systems on a transputer network, in: A.E. Fincham and B. Ford, eds., *Parallel Computation, Proceedings of the IMA Conference on Parallel Computation, Oxford, UK, September 18–20, 1991* (Oxford University Press, Oxford, 1993) 253–271.

[4] A.T. Chronopoulos, Towards efficient parallel implementation of the CG method applied to a class of block tridiagonal linear systems, in: *Proceedings Supercomputer '91* (IEEE Computer Society Press, Los Alamitos, CA, 1991) 578–587.

[5] A.T. Chronopoulos and C.W. Gear, s-Step iterative methods for symmetric linear systems, *J. Comput. Appl. Math.* 25 (1989) 153–168.

[6] A.T. Chronopoulos and S.K. Kim, s-Step Orthomin and GMRES implemented on parallel computers, Technical Report 90/43R, UMSI, Minneapolis, MN (1990).

[7] L.G.C. Crone and H.A. van der Vorst, Communication aspects of the Conjugate Gradient method on distributed memory machines, *Supercomputer* X (6) (1993) 4–9.

[8] E.F. D'Azevedo and C.H. Romine, Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessors, Technical Report ORNL/TM-12192, Oak Ridge National Lab. (1992).

[9] J. De Keyser and D. Roose, Distributed mapping of SPMD programs with a generalized Kernighan–Lin heuristic, in: W. Gentzsch and U. Harms, eds., *High-Performance Computing and Networking*, Lecture Notes in Computer Science 797 (Springer-Verlag, Berlin, 1994) 227–232.
[10] J.W. Demmel, M.T. Heath and H.A. van der Vorst, Parallel numerical linear algebra, *Acta Numer.* 2 (1993).
[11] E. de Sturler, A parallel restructured version of GMRES(m), Technical Report 91-85, Delft University of Technology (1991).
[12] E. de Sturler, A parallel variant of GMRES(m), in: R. Vichnevetsky and J.H.H. Miller, eds., *Proceedings 13th IMACS World Congress on Computation and Applied Mathematics* (IMACS, Criterion Press, Dublin, 1991) 682–683.
[13] E. de Sturler, Incomplete Block LU preconditioners on slightly overlapping subdomains for a massively parallel computer, *Appl. Numer. Math.* (to appear).
[14] E. de Sturler, Iterative methods on distributed memory computers, Ph.D. Thesis, Delft University of Technology, Delft, Netherlands (1994).
[15] J.J. Dongarra, I.S. Duff, D.C. Sorensen and H.A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers* (SIAM, Philadelphia, PA, 1991).
[16] C. Farhat and F.X. Roux, Implicit parallel processing in structural mechanics, Technical Report CU-CSSC-93-26, Center for Aerospace Structures, College of Engineering, University of Colorado, Boulder, CO (1993).
[17] M.R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.* 49 (1954) 409–436.
[18] Z. Johan, Data parallel finite element techniques for large-scale computational fluid dynamics, Ph.D. Thesis, Stanford University (1992).
[19] Z. Johan, K.K. Mathur, S. Lennart Johnson and T.J.R. Hughes, Mesh decomposition and communication procedures for finite element applications on the connection machine CM-5 system, in: W. Gentzsch and U. Harms, eds., *High-Performance Computing and Networking*, Lecture Notes in Computer Science 797 (Springer-Verlag, Berlin, 1994) 233–240.
[20] G. Meurant, Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2, Technical Report LBL-18023, University of California, Berkeley, CA (1984).
[21] F. Nataf, F. Rogier and E. de Sturler, Optimal interface conditions for domain decomposition methods, Technical Report CMAP-301, Centre de Mathématiques Appliqués, CNRS URA-756, Ecole Polytechnique (1994).
[22] A. Pothen, H.D. Simon and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, *SIAM J. Matrix Anal. Appl.* 11 (1990) 430–452.
[23] Y. Saad and M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 7 (1986) 856–869.
[24] H.D. Simon, Partitioning of unstructured problems for parallel processing, Technical Report RNR-91-008, NASA Ames Research Center, Moffet Field, CA (1991).
[25] G.L.G. Sleijpen, H.A. van der Vorst and D.R. Fokkema, BiCGstab(l) and other hybrid Bi-CG methods, *Numer. Algorithms* 7 (1994) 75–109.
[26] H. Sun and W.P. Tang, Overdetermined Schwarz alternating method, Technical Report CS-93-53, Department of Computer Science, University of Waterloo, Ont. (1993).
[27] K.H. Tan and M.J.A. Borsboom, Problem-dependent optimization of flexible couplings in domain decomposition methods, with an application to advection-dominated problems, Preprint 830, Mathematical Institute, University of Utrecht, Netherlands (1993).
[28] W.P. Tang, Generalized Schwarz splittings, *SIAM J. Sci. Statist. Comput.* 13 (1992) 573–595.