

# Applications of Mathematics

---

Martin Hanke; Marlis Hochbruck; Wilhelm Niethammer  
Experiments with Krylov subspace methods on a massively parallel computer

*Applications of Mathematics*, Vol. 38 (1993), No. 6, 440–451

Persistent URL: <http://dml.cz/dmlcz/104566>

## Terms of use:

© Institute of Mathematics AS CR, 1993

Institute of Mathematics of the Czech Academy of Sciences provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.



This document has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ: The Czech Digital Mathematics Library* <http://dml.cz>

EXPERIMENTS WITH KRYLOV SUBSPACE METHODS  
ON A MASSIVELY PARALLEL COMPUTER

MARTIN HANKE, Karlsruhe, MARLIS HOCHBRUCK, Zürich,  
and WILHELM NIETHAMMER, Karlsruhe

*Summary.* In this note, we compare some Krylov subspace iterative methods on the MASPAR, a massively parallel computer with 16K processors. In particular, we apply these methods to solve large sparse nonsymmetric linear systems arising from elliptic partial differential equations. The methods under consideration include conjugate gradient type methods, semiiterative methods, and a hybrid variant. Our numerical results show that, on the MASPAR, one should compare iterative methods rather on the basis of total computing time than on the basis of number of iterations required to achieve a given accuracy. Our limited numerical experiments here suggest that, in terms of total computing time, semiiterative and hybrid methods are very attractive for such MASPAR implementations.

*Keywords:* massively parallel computers, iterative methods, nonsymmetric linear systems, Krylov subspace methods, preconditionings

*AMS classification:* 65F10, 65W05

1. INTRODUCTION

The University of Karlsruhe recently installed a massively parallel computer, a MASPAR MP-1 with 16,384 processors. Like the Connection Machine CM-2, the MASPAR is extremely well-suited (cf. Section 3) for the particular data structures arising from finite-difference discretizations of elliptic partial differential equations, such as

$$(1) \quad \begin{aligned} Lu \equiv -a\Delta u + bu_x + cu_y + du &= f && \text{in } \Omega = (0, 1) \times (0, 1), \\ u &= g && \text{on } \Gamma = \partial\Omega, \end{aligned}$$

where  $a$ ,  $b$ ,  $c$ ,  $d$ ,  $f$  and  $g$  are appropriate real-valued coefficient functions. These discretizations give rise to large sparse real nonsymmetric linear systems

$$(2) \quad A\mathbf{x} = \mathbf{b}.$$

The development of iterative methods for (2), and their implementation on modern supercomputers is currently an important research area in numerical linear algebra

(cf., e.g., Freund, Golub, and Nachtigal [5]). It is well-known that on supercomputers, and in particular on single-instruction-multiple-data (SIMD) machines, it is not only the rate of convergence but also the parallel potential of the particular iterative schemes which matters. We refer to Niethammer [12] for a comparison of Krylov subspace methods on vector computers, and to Tong [18] for a similar comparison on the Connection Machine CM-2.

In this note, we want to summarize some of our numerical experiments on the MASPAR with Krylov subspace methods which compute an approximate solution of (2). In Section 2, we give a short description of Krylov subspace methods. These can be divided into two main classes: conjugate gradient type methods, and semiiterative methods. While the most important basic operation—as far as the number of arithmetical operations is concerned—is the same for both classes, namely the matrix-vector product with  $A$ , there is nevertheless a major difference between these two types of methods: semiiterative methods require certain parameters which depend on a priori required spectral information on  $A$ ; conjugate gradient type methods are parameter-free, but accumulate similar information by means of inner products which are computed during the iterative process. Note that no inner products arise in semiiterative methods. As we will show below, on a massively parallel computer like the MASPAR, it is no longer the matrix-vector multiplication which dominates the cost per iteration, because an inner product is more expensive in terms of computational time. Hybrid variants combine methods from both classes in order to take advantage of their particular properties: typically, a few steps of a conjugate gradient type method are performed which acquire enough spectral information on  $A$ , to switch to a promising semiiterative methods.

For our numerical experiments we selected representative methods from each class, and one hybrid method; we also included preconditioning. In Section 3 we describe some aspects concerning the implementation of these methods on the MASPAR. Our codes are written in MPL, a parallel extension of  $C$ ; we emphasize the simplicity of the MPL routines.

In the final section we present our numerical results. We consider two model problems taken from the literature. Our results are significantly different, when compared on the basis of number of iterations required or on the basis of total computing time; the semiiterative methods converged fastest in all our experiments, the timings of the hybrid method are also quite promising.

## 2. KRYLOV SUBSPACE METHODS

Krylov subspace methods are known as powerful methods for the iterative solution of linear systems. In each step, they produce approximations  $\mathbf{x}_n$  to the exact solution  $A^{-1}\mathbf{b}$  of the form

$$(3) \quad \mathbf{x}_n = \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A), \quad n = 1, 2, \dots$$

Here  $\mathbf{x}_0 \in \mathbb{R}^N$  is any initial guess for the solution of (2),  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$  is the initial residual, and

$$(4) \quad \mathcal{K}_n(\mathbf{r}_0, A) = \text{span} \{ \mathbf{r}_0, A\mathbf{r}_0, \dots, A^{n-1}\mathbf{r}_0 \}$$

is the  $n$ th Krylov subspace with respect to  $\mathbf{r}_0$  and  $A$ .

In the following, we denote by  $\Pi_n$  the set of all complex polynomials of degree at most  $n$ , and by  $\Pi$  the set of all complex polynomials.

From (3) and (4) it is straightforward to deduce that the  $n$ th residual vector  $\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n$  can be written as

$$(5) \quad \mathbf{r}_n = p_n(A)\mathbf{r}_0, \quad n = 0, 1, 2, \dots,$$

where the so-called residual polynomials  $p_n \in \Pi_n$  fulfill

$$p_n(0) = 1.$$

It is obvious from (5) that the convergence of a Krylov subspace method will be fastest if its associated residual polynomials are as small as possible in a region  $\mathcal{G}$  enclosing the spectrum of  $A$ . This leads to the polynomial minimization problem

$$\max_{z \in \mathcal{G}} |p_n(z)| = \min_{p \in \Pi_n : p(0)=1} \max_{z \in \mathcal{G}} |p(z)|,$$

for which an exact solution is known only in special cases.

The various Krylov subspace methods differ in the way they approach the computation of the residual polynomials. The classical Hestenes-Stiefel conjugate gradient (CG) method [8], for instance, chooses the residual polynomials to be orthogonal with respect to the inner product

$$\langle p, q \rangle := \mathbf{r}_0^T p(A)q(A)\mathbf{r}_0, \quad p, q \in \Pi.$$



As a result, the  $n$ th iterate can be computed by means of three-term or coupled two-term recurrences, and the residual vectors satisfy a minimization property in the  $A^{-1}$  norm:

$$\|\mathbf{r}_n\|_{A^{-1}} = \sqrt{\mathbf{r}_n^T A^{-1} \mathbf{r}_n} = \min_{\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A)} \|\mathbf{b} - A\mathbf{x}\|_{A^{-1}}.$$

Unfortunately, classical CG only works for symmetric positive definite systems of equations.

In the past few years many new generalizations of the conjugate gradient method to nonsymmetric linear systems have been proposed. It is well-known that for general nonsymmetric linear systems no method exists which minimizes a given norm of the residual in each step using only short recurrences (cf. Faber and Manteuffel [4]). For example, the residual vectors of the GMRES method, developed by Saad and Schultz [15], satisfy a minimization property in the Euclidean norm:

$$\|\mathbf{r}_n\|_2 = \min_{\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A)} \|\mathbf{b} - A\mathbf{x}\|_2,$$

but long recurrences are required to update the iterates. Therefore it is usually impossible to run the full algorithm and it is necessary to use restarts. In the following, we refer to GMRES( $m$ ) as the GMRES algorithm restarted after each cycle of  $m$  iterations.

Alternatively, generalized CG-type methods have been based on other concepts such as biorthogonality (like BCG, cf. Lanczos [9], as its most popular representative), that still allow short recurrences. Unfortunately, most of these algorithms lack numerical stability. Therefore, several modifications have been suggested recently. One of them, the quasi-minimal residual (QMR) algorithm by Freund and Nachtigal [7], uses a look-ahead version of the Lanczos algorithm [6] and in addition imposes a quasi-minimization principle. This combination leads to a stable algorithm. Another modification is Van der Vorst's BiCGStab algorithm [19], which appends a smoothing step on top of the BCG algorithm, but does not address the breakdown problem. For an excellent overview on Krylov subspace methods for nonsymmetric linear systems we refer the reader to the survey paper [5].

All methods mentioned so far are parameter-free, in the sense that no a priori information on the spectrum of the matrix  $A$  is used. Another important class of Krylov subspace methods consists of parameter dependent or semiiterative methods as introduced by Varga [20]. These require some spectral information on the matrix  $A$  before the algorithm itself can be started. With this information at hand, tools from complex polynomial approximation theory can be applied to determine suitable residual polynomials  $p_n$  for (5). We refer to the survey paper by Eiermann, Niethammer and Varga [3] for an extensive exposition of semiiterative methods. The most

important representatives out of this class are the Chebyshev algorithm (cf. Mantueffel [10]), stationary  $k$ -step methods (Niethammer and Varga [13]), and cyclic Richardson methods (cf., e.g., Smolarski and Saylor [16]).

From an implementation point of view, the main difference between parameter-free and semiiterative methods is that, for the latter, no computation of inner products is necessary. On massively parallel machines the computation of inner products represents a bottleneck of the implementation, and therefore semiiterative methods have a certain advantage in computational time per iteration over parameter-free methods.

Since spectral information is usually not available and expensive to obtain, new hybrid methods (cf., e.g., Smolarski and Saylor [16], Nachtigal, Reichel, and Trefethen [11], and Starke and Varga [17]), which try to combine the advantages of parameter-free and semiiterative methods, recently have gained importance. The basic idea of hybrid algorithms is to start with a parameter-free method such as GMRES, to determine suitable parameters for a semiiterative method. In the second phase, one switches to a much cheaper semiiterative method, using the above parameters. If the semiiterative method does not converge, another GMRES phase is run to improve the set of parameters, and so forth.

For our experiments on the MASPAR we concentrated on four particular methods: BiCGStab, as a representative of parameter-free methods based on the Lanczos process, restarted GMRES(16) which is based on the Arnoldi process, stationary two- and four-step semiiterative methods, and the hybrid GMRES-Richardson method by Nachtigal, Reichel, and Trefethen. For the latter, Hybrid(16), we used cycles of length 16 and based the implementation of the Richardson cycle on Horner's scheme. However, we noticed that this variant is susceptible to numerical instabilities. We did not consider the QMR algorithm in this paper because preliminary experiments with QMR showed that look-ahead is substantial for our examples. An actual implementation of QMR with look-ahead on the MASPAR is currently under work.

Note that none of the methods under consideration involves multiplications with  $A^T$ , i.e., they are transpose-free. We would like to stress that the implementation of the multiplication with  $A^T$  does not involve any difficulties in our particular application. It can be implemented similar to the multiplication with  $A$  at exactly the same expenses.

Today, Krylov subspace methods are rarely applied to the basic linear system without preconditioning. Preconditioning means that one transforms the underlying equation into an equivalent one by multiplying  $A$  with adequate nonsingular matrices  $Q_1$  and  $Q_2$  from the left and/or from the right. This leads to the preconditioned problem

$$Q_1 A Q_2 \mathbf{z} = Q_1 \mathbf{b}, \quad \mathbf{x} = Q_2 \mathbf{z}.$$

The idea is to determine  $Q_1$  and  $Q_2$  in such a way that the new system has better spectral properties, and thus, iterative methods can converge faster. As a rule of thumb, a good preconditioner somehow satisfies  $Q_2 Q_1 \approx A^{-1}$ .

Usually, the matrices  $Q_1$  and  $Q_2$  are only given implicitly. To gain good speedup, it is important that their implementation has an inherent parallel structure that can be employed on the MASPAR. In our numerical experiments we used (i) the original matrix without preconditioning, and (ii) symmetric SSOR preconditioning with red/black ordering and relaxation parameter  $\omega = 1$ . Note that  $\omega = 1$  is the optimal relaxation parameter for SSOR in our applications (cf., e.g., Ortega [14, p. 175]). For implementation details concerning SSOR we refer to [14, Section 3.4].

### 3. IMPLEMENTATION

The SIMD machine MASPAR MP-1 has 16,384 (4-Bit) processors, arranged in a regular two-dimensional toroidal grid of  $128 \times 128$  processors. Another processor takes the exceptional role of the array control unit (ACU) which controls the execution of the grid processor's operations, and carries out the serial parts of the algorithm. The ACU also contains the shared memory. The MP-1 uses a UNIX workstation as front-end computer. All programs are written and compiled on the front-end; in addition, there is a feature to switch to front-end routines to execute bigger sequential parts of the program. Our codes were written in MPL, a parallel extension of C.

Each of the MASPAR processors has forty 32-bit registers, and can use up to 16 Kbytes local memory for so-called `plural` variables; 128 Kbytes shared memory of the ACU contain the `single` variables. All our computations have been performed in double precision.

Employing the natural data mapping of the unit square domain  $\Omega = (0, 1) \times (0, 1)$  onto the MASPAR processor array (cf. [14]), each processor maintains one of the grid unknowns of a regular mesh with mesh size  $h = 1/129$ ; note that we did not reserve processors for grid points on the boundary  $\Gamma = \partial\Omega$ . The  $N$ -dimensional vector  $\mathbf{x}$  and the right hand side vector  $\mathbf{b}$  thus reduce to simple `plural double` variables in MPL.

The block tridiagonal matrix  $A$  from a central difference discretization of the partial differential equation (1) can be represented as a five-point stencil in the usual way, cf., e.g., [14]. The stencil elements per grid point are stored in the respective processors in a parallel initialization step.

A multiplication with  $A$  then simply requires the extremely fast `xnet`-communication with the nearest neighbors in the North, West, East, and South. Extra considerations are necessary concerning the grid points on the boundary. As in conventional serial discretizations we eliminate boundary elements from the system by

incorporating their contribution into the right hand side. Due to the “wrap around” concept of the MASPAR, stencils of close-to-boundary grid points refer to grid points on the opposite side of the square. This is easily fixed by initializing the respective stencil elements to zero. In this fashion, the final matrix-vector product reduces to a one-line statement in MPL, cf. the first part of Figure 1. Again, we emphasize the simplicity of the MPL routines.

```

plural double x,y,z;
plural double c,n,s,w,e;
        /* c,n,s,w,e contain the five-point stencil */

        /* matrix-vector product: */
y = c*x + n*xnetN[1].x + s*xnetS[1].x
        + w*xnetW[1].x + e*xnetE[1].x

        /* SSOR preconditioned matrix-vector product: */
z = 0;
if (red)
{   z = n*xnetN[1].x + s*xnetS[1].x
        + w*xnetW[1].x + e*xnetE[1].x;   }
y = x - n*xnetN[1].z - s*xnetS[1].z
        - w*xnetW[1].z - e*xnetE[1].z;

```

Figure 1: MPL-listings of the basic routines

For the red-black-SSOR preconditioned matrix-vector product we obtain a slightly more complicated subroutine (cf. Figure 1): in a first substep, the stencil is evaluated for all red grid points; the second substep then computes the final update in all grid points. In this way, the SSOR implementation takes about twice the time as a standard matrix-vector product (cf. Table 1).

Compared with these local communication routines, the implementation of the inner product involves global communication. We compute inner products with the built-in function `reduceAddd`, which performs a global summation over one `plural double` variable of all processors.

For illustration, we provide the execution times for our basic linear algebra routines in Table 1. The consideration of semiiterative methods is justified by the fact that one inner product is more time-consuming than a matrix-vector product.

$\mathbf{x} + \alpha\mathbf{y}$	95 $\mu\text{secs}$
$\mathbf{x}^T\mathbf{y}$	480 $\mu\text{secs}$
$A\mathbf{x}$	445 $\mu\text{secs}$
$Q_1AQ_2\mathbf{x}$	805 $\mu\text{secs}$

Table 1: Timings of the basic routines

#### 4. EXPERIMENTAL RESULTS

For our numerical experiments we considered two particular partial differential equations (1) with operators

(a) 
$$Lu \equiv -\Delta u + \frac{8}{h} u_s,$$

(b) 
$$Lu \equiv -e^{st} \Delta u + (50(s+t) - te^{st})u_s - se^{st}u_t + (5 + \frac{1}{1+s+t})u.$$

In both examples, the functions  $f$  and  $g$  were chosen such that  $u \equiv 1$  is an exact solution of  $Lu = f$ . The first example, which is taken from [17], is chosen such that the associated cell-Reynolds number is four, and thus gives a highly nonsymmetric but well-conditioned linear system  $Ax = b$ ; the coefficient matrix of the second example—taken from [1]—is more ill conditioned. Thus, the two examples emphasize different properties of the iterative solvers. We would like to stress, that it is not merely the condition number which affects the rate of convergence of iterative methods, but that the location of the eigenvalues and the condition number of the associated eigenvalue problem play an important role.

In the first example, the spectrum of  $A$  can be computed explicitly (cf. [17]): it fills up a rectangle with vertices  $2 < \text{Re } \lambda < 6$  and  $|\text{Im } \lambda| < 2\sqrt{15}$ . Following Eiermann [2, Sect. 4] we used this information to determine parameters for an almost optimal stationary four-step method (SIM) with a convergence factor  $\kappa \approx 0.84$ . Using Young's functional equation between the eigenvalues of the Jacobian and those of the SOR iteration matrix, we computed the eigenvalues of the SSOR matrix, too. Its spectrum is better suited for a two-step semiiterative method than for a four-step method. Manteuffel's algorithm [10] was used to find the parameters for the optimal stationary two-step method; the corresponding convergence factor is  $\kappa \approx 0.74$ .

In Table 2 we present the time that is required in the average to perform one iteration by SIM, BiCGStab, GMRES(16), and Hybrid(16), respectively; the timing of Hybrid(16) is based on the (quite pessimistic) assumption that one out of seven cycles of Hybrid(16) is a GMRES(16) cycle while the remaining six cycles are Richardson cycles of length 16, each. Our timings concern the basic iteration and the preconditioned iteration, respectively.

	without precond.	with precond.
BiCGStab	3.98	4.69
GMRES(16)	11.30	11.68
SIM	1.02	1.37
Hybrid(16)	3.10	3.49

Table 2: Average time per iteration (in msecs)

What is important here is that the semiiterative method is by far the most inexpensive per iteration; the most expensive method is GMRES(16). On the other

hand, we can see that SSOR preconditioning leads to an overhead of 10 to 40% in time, the precise number depending on the particular iterative scheme. Clearly, it is the semiiterative method where the time per iteration increases the most with preconditioning, because here the matrix-vector multiplication dominates the overall costs.

We now present the numerical results. In all our figures, the solid line represents BiCGStab, the dashed line GMRES(16), the dotted line the semiiterative method, and the dash-dotted line Hybrid(16). Our plots show the relative residual norms  $\|\mathbf{r}_n\|_2/\|\mathbf{r}_0\|_2$  versus the iteration index  $n$  on the left, and the relative residual norms versus time (in seconds) on the right.

Figure 2 contains the results for equation (a) without preconditioning. Since  $A$  is well-conditioned in this case, the semiiterative method performs extremely well, both with respect to number of iterations and with respect to time. We point out the five peaks of the plot of Hybrid(16) where the Richardson cycle changed the iteration to the worse, forcing the program to switch back to GMRES. In total, seven GMRES(16) cycles were performed in this case. In this example, BiCGStab needs more than twice as many iterations as GMRES(16) but nevertheless significantly less time.

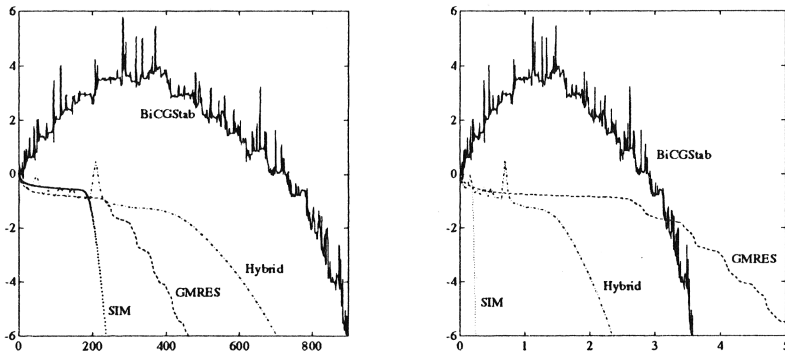


Figure 2: Example (a), no preconditioner

The same example with preconditioning gave the results in Figure 3. In particular, we emphasize the tremendous speedup of BiCGStab. It required fewest iterations, and only the optimized semiiterative method was faster in time. Hybrid(16) needed more iterations than GMRES(16) but was almost as fast as BiCGStab. The reason is that Hybrid(16) required only one GMRES cycle in this test.

Next, we consider the more ill-conditioned second problem. For this example, we only present the results with preconditioning, cf. Figure 4. Our motivating remarks from the introduction are clearly underlined: CG-type methods required (signifi-

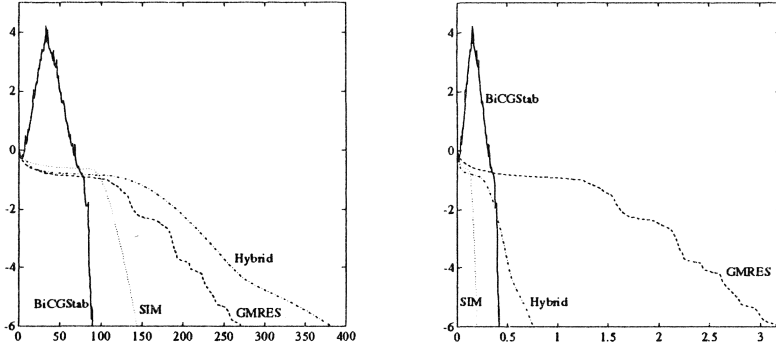


Figure 3: Example (a), SSOR preconditioner

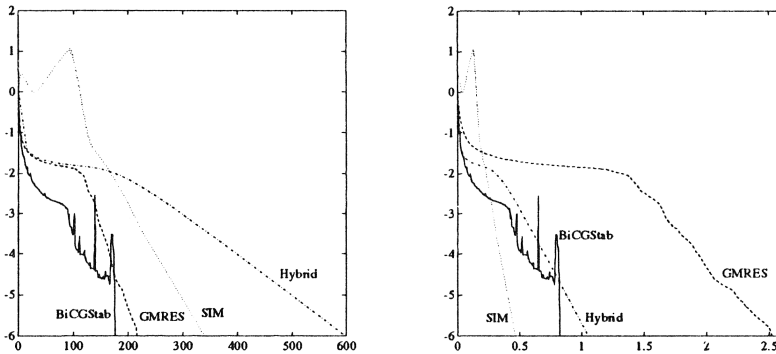


Figure 4: Example (b), SSOR preconditioner

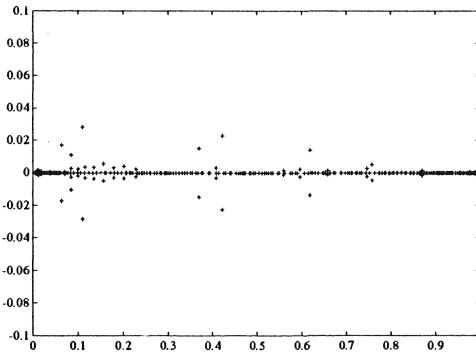


Figure 5: Eigenvalue estimates for Example (b)

cantly) less iterations in this example than the semiiterative and the hybrid method; nevertheless, as far as time is concerned, there is no significant difference between BiCGStab and Hybrid(16); GMRES(16) is not competitive here.

To include a good semiiterative method for ease of comparison, we acquired eigenvalue estimations from the look-ahead Lanczos algorithm [6] (cf. Figure 5). From these estimates we determined a stationary two-step method as before with convergence factor  $\kappa \approx 0.92$ . Clearly, this is no practical approach, but our emphasis here was merely to demonstrate the potential of semiiterative and hybrid methods.

## 5. CONCLUSION

We conclude that within our limited set of examples, the semiiterative method would always have been our favorite choice because it achieved the required accuracy fastest. However, the a priori determination of the parameters that are required to start the semiiterative method is not always practical. In principle, it should be possible to design hybrid methods that are close to the semiiterative ones as far as computational time is concerned. Nevertheless, we observe that there is still a significant gap between the timings of Hybrid(16) and the optimized semiiterative methods. It is a topic of current research to narrow this gap.

**Acknowledgement.** This project has been supported by the Deutsche Forschungsgemeinschaft (DFG). We are grateful to Michael Eiermann, Gerhard Starke, and Richard Varga for their helpful advice and to Noël Nachtigal for his careful reading of this manuscript.

## References

- [1] *D. Baxter, J. Saltz, M. Schultz, S. Eisenstat, and K. Crowley*: An experimental study of methods for parallel preconditioned Krylov methods, Tech. Rep. RR-629, Department of Computer Science, Yale University, 1988.
- [2] *M. Eiermann*: On semiiterative methods generated by Faber polynomials, *Numer. Math.* *56* (1989), 139–156.
- [3] *M. Eiermann, W. Niethammer, and R. S. Varga*: A study of semiiterative methods for nonsymmetric systems of linear equations, *Numer. Math.* *47* (1985), 505–533.
- [4] *V. Faber and T. Manteuffel*: Necessary and sufficient conditions for the existence of a conjugate gradient method, *SIAM J. Numer. Anal.* *21* (1984), 352–362.
- [5] *R. W. Freund, G. H. Golub, and N. M. Nachtigal*: Iterative solution of linear systems, *Acta Numerica* *1* (1992), 57–100.
- [6] *R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal*: An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, *SIAM J. Sci. Statist. Comput.* *14* (1993), 137–158.
- [7] *R. W. Freund and N. M. Nachtigal*: QMR: a quasi-minimal residual method for non-Hermitian linear systems, *Numer. Math.* *60* (1991), 315–339.
- [8] *M. R. Hestenes and E. Stiefel*: Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standards* *49* (1952), 409–436.
- [9] *C. Lanczos*: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, *J. Res. Nat. Bur. Standards* *45* (1950), 255–282.



- [10] *T. A. Manteuffel*: The Tchebychev iteration for nonsymmetric linear systems, *Numer. Math.* *28* (1977), 307–327.
- [11] *N. M. Nachtigal, L. Reichel and L. N. Trefethen*: A hybrid GMRES algorithm for nonsymmetric linear systems, *SIAM J. Matrix Anal. Appl.* *13* (1992), 796–825.
- [12] *W. Niethammer*: Iterative solution of non-symmetric systems of linear equations, In: *Numerical Mathematics, Singapore 1988* (R. P. Agarwal, Y. M. Chow and S. J. Wilson, eds.), Birkhäuser, Basel, 1988, pp. 381–390.
- [13] *W. Niethammer and R. S. Varga*: The analysis of  $k$ -step iterative methods for linear systems from summability theory, *Numer. Math.* *41* (1983), 177–206.
- [14] *J. M. Ortega*: *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, London, 1988.
- [15] *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.
- [16] *D. C. Smolarski and P. E. Saylor*: An optimum iterative method for solving any linear system with a square matrix, *BIT* *28* (1988), 163–178.
- [17] *G. Starke and R. S. Varga*: A hybrid Arnoldi-Faber iterative method for nonsymmetric systems of linear equations, *Numer. Math.* *64* (1993), 213–240.
- [18] *C. Tong*: The preconditioned conjugate gradient method on the Connection Machine, In: *Proceedings of the Conference on Scientific Applications of the Connection Machine* (H. Simon, ed.), World Scientific, Singapore, New Jersey, London, Hong Kong, 1989, pp. 188–213.
- [19] *H. A. Van der Vorst*: Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* *13* (1992), 631–644.
- [20] *R. S. Varga*: *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, New Jersey, 1962.

*Authors' addresses: Martin Hanke, Wilhelm Niethammer, Institut für Praktische Mathematik, Universität Karlsruhe, Englerstr. 2, W-7500 Karlsruhe 1, Germany; email af01@dkauni2.bitnet; Marlis Hochbruck, Interdisciplinary Project Center for Supercomputing, ETH Zürich, ETH-Zentrum, CH-8092, Zürich, Switzerland; email na.hochbruck@na-net.ornl.gov.*