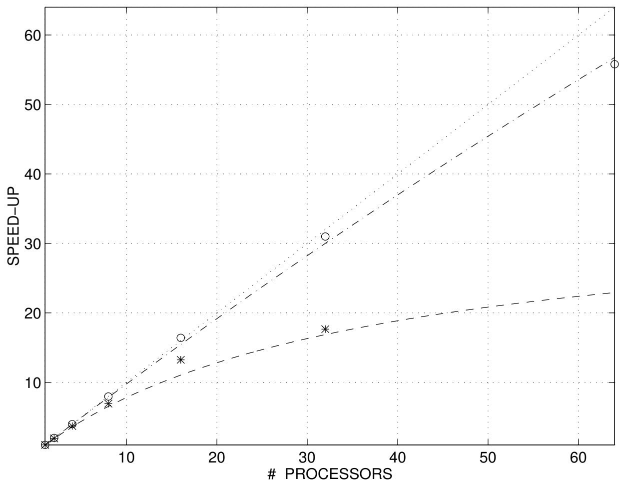
Universiteit Utrecht

Department of Mathematics



Approximated Implicit Time-stepping Schemes in a Distributed Memory Parallel Environment

by

Mikhail A. Botchev and Henk A. van der Vorst

Preprint

nr. 1054

March, 1998

0

Also available at URL `http://www.math.ruu.nl/publications/`

Approximated implicit time-stepping schemes in a distributed memory parallel environment[†]

Mikhail A. Botchev[*‡] Henk A. van der Vorst[*]

March 13, 1997

Abstract

The recently proposed Minimal Residual Approximate Implicit (MRAI) schemes [3] have been developed as cheaper and parallelizable alternatives for implicit time stepping. For an implicit scheme of interest, the approach is based on the use of a restricted number of GMRES iterations to solve the implicit (linearized) system. The main difference with the conventional use of iterative techniques is that the convergence of the iterative process is not checked, but the step size of the scheme is adjusted adaptively for stability. Since the GMRES process is relatively easy to parallelize, the MRAI schemes are also well parallelizable. On platforms as the Cray T3E and IBM SP2, the MRAI codes show similar speed-ups as for explicit schemes, while the stability properties are much better. As a model problem we consider the 3D spatially discretized heat equation. Speed-up results for the Cray T3E and IBM SP2 are reported and analyzed.

1 Introduction

Implicit time stepping leads to the necessity to solve large sparse linear systems. This is usually realized by a direct method, and direct methods for sparse matrices are often difficult to parallelize. Application of iterative schemes for the linear solves in implicit time-stepping codes may lead to a significant improvement in performance on a sequential computer (see e.g. [10, 5]), and this may be attractive from the parallelization point of view.

A simple approach is to perform only a modest number of iterations for the linear solves. The use of a few steps of a minimal residual iterative scheme, for example, GMRES [14, 2], is especially favorable in this context. This combination

[†]This research was supported by the Netherlands organization for scientific research NWO, project 95MPR04

[*]Mathematical Institute, Utrecht University, P.O.Box 80.010, 3508 TA Utrecht, the Netherlands. E-mail: [botchev, vorst]@math.ruu.nl, fax: (+31 30) 251 8394

[‡]Corresponding author. Since June 1, 1998 the address is: CWI, P.O.Box 94079, 1090 GB Amsterdam, the Netherlands, fax: (+3120) 592 4199

is referred to as Minimal Residual Approximate Implicit (MRAI) time stepping [3]. The main difference with the conventional use of iterative techniques is that the iterative process convergence in MRAI is not checked. However, the step size for the time stepping is adjusted adaptively to assure stability. A natural way to derive an MRAI scheme is to start from some given implicit scheme. The resulting approximated implicit scheme can be interpreted as explicit and, hence, is not unconditionally stable. The stability control proposed in [3] allows efficient automatic selection of the step size in the MRAI schemes. We will present numerical results for the parallelization of the MRAI time-stepping schemes.

The outline of our paper is as follows. In Section 2 we describe briefly the MRAI time stepping and its parallelization aspects. In Section 3, we estimate the expected speed-up of the MRAI schemes. Numerical experiments and actual speed-up results are presented in Section 4. Our conclusions are formulated in Section 5.

2 MRAI approach and modification of implicit codes

2.1 MRAI time stepping

Suppose that we are interested in an implicit scheme for the solution of a stiff system of ODE's

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}), \qquad \boldsymbol{y}|_{t=0} = \boldsymbol{y}^0 \in \mathbb{R}^N, \tag{1}$$

for example, the Euler Backward (EB) scheme

$$\boldsymbol{y}^{n+1} - \boldsymbol{y}^n = \tau \boldsymbol{f}(t_{n+1}, \boldsymbol{y}^{n+1}). \tag{2}$$

One needs to solve the nonlinear equation (2) in order to obtain the solution $\boldsymbol{y}^{n+1}$ on the next time level t_{n+1}. This is usually done by linearization and solving the resulting Jacobian equation

$$(I - \tau J)(\boldsymbol{y}^{n+1} - \boldsymbol{y}^n) = \tau \boldsymbol{f}(t_{n+1}, \boldsymbol{y}^n), \qquad J = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_{n+1}, \boldsymbol{y}^n). \tag{3}$$

In a Newton process this procedure is repeated.

The basic idea in the MRAI time stepping [3] is as follows: at each time step, for one or more Newton iteration, we solve (3) approximately with k steps of GMRES [14, 2]. The value for k is taken small (say 5). Since the GMRES process involves only explicit matrix-vector operations with $I - \tau J$, the resulting time stepping is explicit, and this makes MRAI schemes easy to parallelize.

Analysis in [3] shows that for a consistent scheme, an initial guess $\boldsymbol{y}^{n+1}_{(0)}$ for the iterative process has to be taken appropriately. For example, $\boldsymbol{y}^{n+1}_{(0)}$ can be taken as the solution obtained with one step of the Euler Forward scheme.

The approach can also be followed for higher order implicit schemes. In that case we obtain $\boldsymbol{y}^{n+1}_{(0)}$ from one step of an explicit scheme of the same (higher) order. Note that straightforward linearization, as in (3), would diminish the order of the scheme to 2, but this can be easily repaired (see [3] for details).

Unlike other approaches for the usage of iterative methods in implicit time stepping, in MRAI schemes one does not control the residual reduction achieved in GMRES; the number of iterations k is simply kept fixed. A problem in conventional approaches is that it is often not clear what tolerance for the residual reduction stopping criterion should be used; for a too strict tolerance an unnecessary amount of computational work has to be done, and, on the other hand, a too modest tolerance might lead to unstability.

The MRAI scheme is an approximation for an implicit scheme and therefore it is not unconditionally stable. A step size control for stability is proposed in [3]. It is based on information extracted from the GMRES process. For MRAI schemes as based on EB, the Trapezoidal rule and some others, it is possible to arrange the computations so that the step size can be changed immediately at the current time step without recomputing the GMRES part.

Numerical tests and comparisons of the MRAI schemes with other time-stepping strategies (as in [16, 4]) can be found in [3]. The MRAI time stepping has been used in the general purpose MHD solver VAC [13, 17].

In this paper it is assumed that the Jacobian J is not available explicitly; its action on a vector is approximated by the directional difference,

$$J(t_n, \boldsymbol{y}^n)\boldsymbol{v} \approx \frac{\boldsymbol{f}(t_n, \boldsymbol{y}^n + \epsilon \boldsymbol{v}) - \boldsymbol{f}(t_n, \boldsymbol{y}^n)}{\epsilon}, \qquad \epsilon = \frac{\sqrt{\delta}\, \boldsymbol{v}^T \boldsymbol{y}^n}{\|\boldsymbol{v}\|}, \tag{4}$$

where δ is the floating point relative machine accuracy.

2.2 Parallelization of MRAI schemes

Several approaches for the parallelization of Krylov subspace iterative methods, in particular, GMRES have been proposed (see e.g. [2, 19, 1, 8]). We briefly describe here our parallel implementation of the GMRES part in MRAI.
The main CPU time consuming ingredients in GMRES are
(i) the modified Gram-Schmidt process, for the computation of an orthogonal $N \times (k+1)$-matrix V_{k+1} with columns $(\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_{k+1})$, and
(ii) k vector updates for the computation of the approximated solution $\boldsymbol{x}_k$ as

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + V_k u, \qquad u \in \mathbb{R}^k, \tag{5}$$

where $\boldsymbol{x}_0$ is the initial guess.

The modified Gram-Schmidt process requires k matrix–vector multiplications with $I - \beta\tau J$, and $k(k+1)/2 + k$ inner products. The inner products have to be done sequentially, so that $k(k+1)/2 + k$ synchronized communications are required. Of course, this may lead to a decrease in performance on distributed

Table 1: Speed-up for modified and classical Gram-Schmidt on the Cray T3E

# of PEs	modified Gram-Schmidt (36 communications)	classical Gram-Schmidt (8 communications)
1	0.424 sec	0.423 sec
2	0.214 sec	0.213 sec
4	0.106 sec	0.104 sec
8	0.047 sec	0.047 sec

memory computers. If this is the case then the classical Gram-Schmidt process or, better, techniques of [8, 1] can be employed, in which communications can be combined.

On the Cray T3E and IBM SP2, for the values of k we are interested in ($k \leqslant 7$), there is no need to look for alternatives with less synchronization for modified Gram-Schmidt. This is evident from our tests on the Cray T3E (Table 1) and the IBM SP2 (the results were similar to those in Table 1). In these runs, the matrix was taken to be diagonal of order $N \times N$, $N = 80\,000$, and $k = 7$ steps of the Gram-Schmidt process were carried out. Each inner product was computed locally on the processor elements (PEs), then a global summation function, based on the SHMEM communication library [6], was used. These communications had to be done synchronously; all together, there were 36 synchronization points in the modified Gram-Schmidt process versus 8 in the classical one. As we see from Table 1, the speed-up results for the classical and the modified Gram-Schmidt versions are virtually identical, at least for this rather large problem and modest number of PEs. For non-diagonal matrices the costs for Gram-Schmidt will be relatively less and, hence, the influence of this communication will be hardly visible.

When the Gram-Schmidt process has been completed and the matrix V_k has been computed, another $\mathcal{O}(k^2)$ operations are required to compute the small k-vector u required in (5). In the Gram-Schmidt process, we accumulate the computed inner products on each PE. Therefore, the vector u can be computed by every PE in order to avoid further communication. Thus, the k vector updates (5) can be done completely in parallel. Note that this $\mathcal{O}(k^2)$ work is relatively unimportant for large N ($N \gg k$).

3 Estimating the speed-up

Let T_p denote the CPU time required to make one time step advance with the MRAI scheme in parallel on p PEs. We will derive estimates for the speed-up $S_p = T_1/T_p$. Although the speed-up estimates will be derived for a particular MRAI scheme that is based on the EB scheme, the analysis can be straightforwardly applied for other

MRAI schemes.

T_p mainly consists of the CPU times spent for the $k(k+1)/2+k$ inner products, the $k+1$ Jacobian-vector products, and 1 function evaluation (FEVAL). We separate the part of T_p which is spent for all the inner products. As we have seen in the previous section, this part of the computations is well parallelizable. Since the Jacobian actions are evaluated matrix free according to (4), each new Jacobian-vector product costs one FEVAL and one inner product. Thus, in total, there are $k(k+1)/2+2k+1$ inner products. Assume that it takes fT_1 to compute all of them by one PE. The remainder of T_1 is spent for $k+2$ FEVALs, each of which takes t_1^{feval} by 1 PE. This means that

$$T_1 = fT_1 + (k+2)t_1^{\text{feval}},$$

and, since the inner product part is almost perfectly parallelizable,

$$T_p = \frac{fT_1}{p} + (k+2)t_p^{\text{feval}}. \tag{6}$$

Assume, for simplicity, that the communications required to perform FEVAL are not overlapped with other computations, then

$$t_p^{\text{feval}} = \frac{t_1^{\text{feval}}}{p} + t_p^{\text{comm}},$$

where t_p^{comm} is the total time spent for communication in FEVAL. Hence, we have that

$$S_p = \frac{T_1}{T_p} = \frac{p}{1+(k+2)p\,t_p^{\text{comm}}/T_1},$$

$$S_p = \frac{p}{1+(1-f)p\,t_p^{\text{comm}}/t_1^{\text{feval}}}. \tag{7}$$

The meaning of the ratio $p\,t_p^{\text{comm}}/t_1^{\text{feval}} =: \alpha_p$ is illuminated by observing that $t_p^{\text{feval}} = (1+\alpha_p)t_1^{\text{feval}}/p$, in other words, the ratio simply shows the communication overhead in FEVAL.

Suppose that 1 FEVAL equals in costs approximately to F inner products. Then, the value $1-f$, which is all the FEVAL costs divided by the total costs (for FEVALs and inner products), can be estimated as

$$1-f = \frac{(k+2)F}{k(k+1)/2+2k+1+(k+2)F}. \tag{8}$$

For a typical value for the number of GMRES iterations in MRAI codes $k=5$, we have

$$1-f = \frac{7F}{26+7F}. \tag{8'}$$

Table 2: Estimates for the FEVAL communication time t_p^{comm} for 1D (three-point stencil), 2D (five-point stencil, $N = n \times n$), and 3D (seven-point stencil, $N = n \times n \times n$). p PEs are logically arranged as $p \times 1$, $\sqrt{p} \times \sqrt{p}$, or $p^{1/3} \times p^{1/3} \times p^{1/3}$ grid

Problem dimension	Grid of PEs	Communication time t_p^{comm}
1D	1D	const$(N; p)$
2D	1D	$\sim \sqrt{N}$, const(p)
2D	2D	$\sim \sqrt{N}$, $\sim \frac{1}{\sqrt{p}}$
3D	1D	$\sim N^{2/3}$, const(p)
3D	2D	$\sim N^{2/3}$, $\sim \frac{1}{\sqrt{p}}$
3D	3D	$\sim N^{2/3}$, $\sim \frac{1}{p^{2/3}}$

We now consider a particular situation, which corresponds to the model problem described in the next section. We will specify the values t_p^{comm} and F in the expressions (7),(8). It will be clear from the presentation how the speed-up analysis can be applied for other cases.

Suppose, (1) stems from the spatial discretization of a PDE, and the function $\boldsymbol{f}$ is a 3D differential operator discretized on the regular seven-point stencil. Let the 3D grid be distributed among the set of PEs logically arranged in a 2D processor grid, so that each PE possesses the whole range of the grid nodes in one direction. Assume for simplicity that $N = n \times n \times n$, $p = \sqrt{p} \times \sqrt{p}$, and, in the FEVAL operation, each PE first successfully sends and receives four messages, and then the FEVAL computations are performed. These four send / receive calls are performed in parallel, therefore

$$t_p^{\mathsf{comm}} = c_1 \frac{N^{2/3}}{\sqrt{p}} + c_2, \tag{9}$$

where c_1 and c_2 are some constants. The term $N^{2/3}/\sqrt{p}$ corresponds to the amount of data sent: if the processor grid becomes denser, for example, $\sqrt{p}$ is increased by factor two, then, evidently, the messages become two times shorter. Of course, if the start-up time term c_2 were zero, this would also reduce the t_p^{comm} by factor two. As we see, the communication time decreases as number of PEs grows. According to Table 2, this can also be the case for other discretized PDE provided that PEs are logically organized in 2D or 3D grid.

Since with the seven-point stencil one needs at least 7 multiplications and 6 additions for the FEVAL operation at each grid point, the FEVAL expenses are $F \geqslant 6.5$. Suppose $F = 9$. With (8$'$) we get $1 - f = 0.7$. Substitution of this together

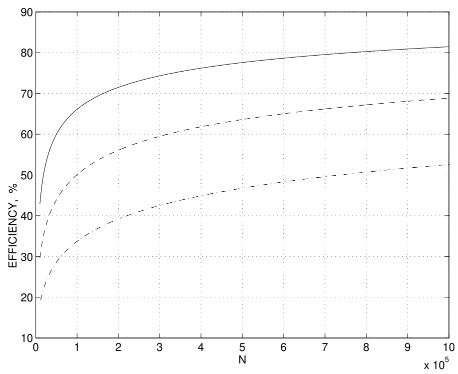


Figure 1: Ratio (MRAI speed-up) / (ideal speed-up)$\cdot 100\%$ versus the problem size N on the IBM SP2 for different number of PEs (solid line—$p = 4$, dashed line—$p = 16$, dashdotted line—$p = 64$)

with (9) into the speed-up estimate (7) leads to

$$S_p = \frac{p}{1 + 0.7\,\dfrac{c_1 N^{2/3}\sqrt{p} + c_2 p}{t_1^{\mathsf{feval}}}}. \tag{10}$$

To determine c_1 and c_2 we propose to run a simple code with a single call to the FEVAL subroutine where t_p^{comm} is measured explicitly. The code is to be run twice, on different number of PEs p, and this gives system of two equations in c_1 and c_2. The value of t_1^{feval} can also be timed explicitly. Such an explicit simple timing has an advantage that the predicted speed-up corresponds exactly to the particular computer, compiler, FEVAL implementation, etc. Since the actual performance may depend on the problem size N, for a new value of N, it is better to redo the timings.

However, it is often reasonable to assume that the performance depends on N only mildly, so that t_1^{feval} is directly proportional to the problem size: $t_1^{\mathsf{feval}} = c_3 N$, c_3 is a constant. Substitution of the last expression into (10) gives an explicit dependence of the speed-up on the size problem N and number of PEs p:

$$S_p(N;p) = \frac{p}{1 + 0.7\,\dfrac{c_1 N^{2/3}\sqrt{p} + c_2 p}{c_3 N}}. \tag{11}$$

In Figure 1, we depicted the dependence (11) for the IBM SP2 with parameters c_1, c_2, c_3 estimated for $N = 64\,000$. The plot shows how large the problem size N should be for a good efficiency $S_p(N;p)/p$. To have an efficiency of at least 50% from the ideal, for example, N should not be less than $1\cdot 10^5$ for $p = 16$, and not less than $7\cdot 10^5$ for $p = 64$.

To adapt the speed-up estimates for a different problem (i.e. for a different FEVAL), one has only estimate the FEVAL expenses according to (8), and adjust the communication time expression (9) (see Table 2).

We note that the estimate (9) and similar, as presented in Table 2, can be reformulated in terms of the hardware parameters r^c_∞ and t^c_0 (the asymptotic communication bandwidth and the latency, respectively). These parameters, together with the scalar performance r^s_∞, can be useful for further performance analysis. For such a technique, we refer the reader to [12, 18].

4 Numerical experiments with the MRAI schemes

In our test runs we have used two MRAI codes. The first one is based on the simple Euler Backward scheme (we refer to this code as EB/MRAI), the second is an experimental MRAI-modified stiff ODE solver LSODE described in [3] (the LSODE/MRAI code). Such a choice of basic implicit schemes, Euler Backward and LSODE, leads to more or less a complete assessment of the MRAI approach, since one scheme, EB, is a simplest implicit scheme, however still actively used in practice, whereas another, LSODE, is a quite advanced high-order scheme.

The LSODE code is a black-box stiff integrator [11], in which the variable-order implicit backward differentiation formulae are used with a Newton process, and the inner linear solves are made by direct methods from LINPACK. In the MRAI/LSODE code, linear solves are replaced by a fixed number of GMRES steps, and the Jacobian evaluation (i.e. the Jacobian action on a vector) is made according to (4). These techniques are similar to those employed in the VODPK code [4], the difference is that in the VODPK code convergence of GMRES is controlled. In both EB/MRAI and LSODE/MRAI codes, the number of GMRES steps was $k = 5$ (this is our default value).

Our model problem is a spatially discretized 3D heat equation (for more details see [16]). The standard seven-point stencil finite difference discretization on the spatial grid $40 \times 40 \times 40$ leads to the system of size $N = 64\,000$. The numerical integration was done for $t \in [0, 0.7]$.

We first briefly comment on how the MRAI strategy competes with other time-stepping techniques for this problem on a sequential computer. In [3], performance of the LSODE/MRAI code was compared with those of the RKC [16, 15] and VODPK [4] codes. For our test problem all three codes performed about the same for the whole range of tolerance parameters. In these runs, speed-ups of which are presented below, the tolerance parameters `atol` and `rtol` in the LSODE/MRAI code were chosen 10^{-3}. With this tolerance, the code requires 22 steps with 283 FEVALs.

The simple EB/MRAI code (based on the Euler Backward scheme) needs $2\,212$ FEVALs to finish the computation within 316 time steps. The step size τ was chosen each time step according to the step size control of [3]. To compare with, the Euler Forward scheme with the largest possible step size would require more than $7\,000$

steps (and the same number of FEVAL operations). Hence, the gain factor achieved by EB/MRAI is about 3.2. Since the MRAI schemes are as well parallelizable as explicit ones, the observed gain factors transfer to parallel distributed memory environment.

For grid-based problems, as our model problem, conventional implicit schemes may be of interest, with efficient direct sparse methods for the solution of the linear systems. An attractive feature of the direct methods is that the LU factors can typically be reused for several time steps. According to the estimates of [5], for a 3D seven-point stencil discretization problem, each sparse LU factorization costs $\mathcal{O}(N^2)$ flops, and, at each time step, forward / backward substitution solve adds to this amount $\mathcal{O}(N^{4/3})$ flops. Let us assume that for the corresponding MRAI scheme the step size is in average 20 times smaller (which is in practice a pessimistic estimate for MRAI), and that an LU factorization is made only for each 10 time steps (these two values, 20 and 10, are hardly possible to occur simultaneously since for larger step sizes the LU factorization has to be updated more often). Even for this strongly biased, in favor of direct methods, situation one still has a substantial gain with the MRAI approach where the work per step is just $\mathcal{O}(N)$. Similar conclusions, although less pronounced, can be made for the 2D case. Besides, direct sparse methods are much more difficult to parallelize [9], so that the picture will be even less favorable for them on a parallel computer.

Furthermore, we note that the above mentioned RKC and VODPK codes virtually possess high parallelism too. However, the RKC code is in general less attractive since it does not work well for Jacobians with complex spectrum. The VODPK concept is specially developed for the inexact Newton method framework, where Newton / GMRES convergence criteria are determined by the user-prescribed accuracy tolerance; our simpler MRAI approach is of interest for a wider class of schemes.

For our model problem, we have parallelized the LSODE/MRAI and EB/MRAI codes using the MPI communication library [7]. The 3D grid was distributed among the PEs in two dimensions, so that each PE owned the whole range of nodes in z-direction. The FEVAL subroutine includes four send and four receive calls to exchange information with the neighboring PEs.

For the predicted speed-up values we have used the relation (10), with the estimated parameters c_1, c_2 (Section 3), which were

$$\begin{array}{lll} \text{Cray T3E:} & c_1 = 6.1\cdot 10^{-7}, & c_2 = 2.3\cdot 10^{-4}, \\ \text{IBM SP2:} & c_1 = 3.0\cdot 10^{-6}, & c_2 = 6.6\cdot 10^{-3}. \end{array}$$

We estimated the parameter F (cf. (8)) for this problem as $F \approx 9$. We note that in the LSODE/MRAI code the number of FEVALs per time step varies, so that our speed-up predictions (which formally are valid for the EB/MRAI code) have only approximate values for LSODE/MRAI.

The speed-up results are presented in Table 3 and in Figure 2. Exactly the

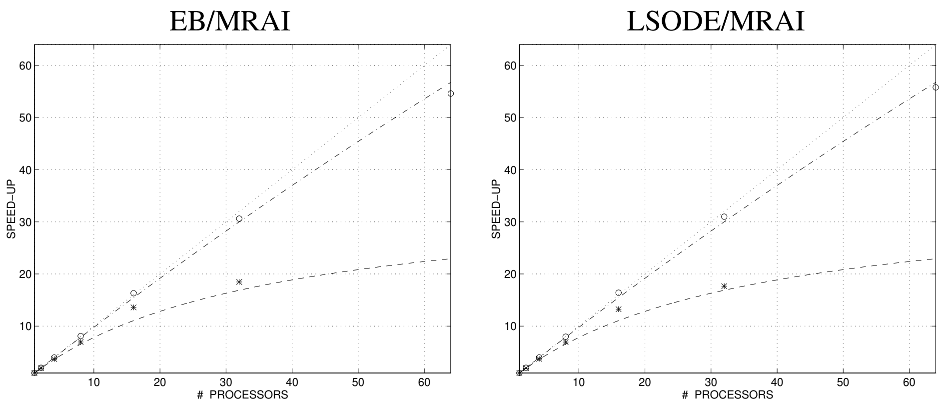


Figure 2: Speed-up results for the Cray T3E (predicted: $-\cdot-$, observed: $\circ$) and IBM SP2 (predicted: $---$, observed: $*$) versus the ideal speed-up ($\cdot\cdot\cdot$)

Table 3: CPU time (sec.) for the 3D heat equation model problem on the Cray T3E and IBM SP2

# of	EB/MRAI		LSODE/MRAI	
PEs	Cray T3E	IBM SP2	Cray T3E	IBM SP2
1	404.2	426.6	55.8	58.3
2	202.8	221.2	28.0	30.3
4	101.8	115.8	14.0	15.7
8	50.0	61.4	7.0	8.4
16	24.8	31.4	3.4	4.4
32	13.2	23.1	1.8	3.3
64	7.4	—	1.0	—

same codes have been executed on the Cray T3E and IBM SP2, but, as we see, the speed-ups for the IBM SP2 are smaller. This is by no means a surprise since the communication start-up time (the latency) is larger for this computer. Indeed, if we assume that the speed of computations on one PE of the Cray T3E and IBM SP2 is approximately the same (which turns out to be realistic), then the difference in the speed-ups is due to the different values of the t_p^{comm}. According to (9), and the estimated values of c_1, c_2, for sufficiently large p the communication is about 30 times faster on the Cray T3E. This is probably not only because of the faster communication start-ups, but also due to the well optimized MPI library on the Cray T3E (in our limited experience, on the Cray T3E, the MPI-based codes often perform only slightly worse than codes based on the Cray's native communication library SHMEM [6]).

5 Conclusions

The recently proposed MRAI time stepping approach can be viewed as an attempt to get parallelizable cheap alternative for implicit schemes, while preserving stability properties as much as possible [3].

Experiments with the MRAI technique on the Cray T3E and the IBM SP2 parallel computers show that the MRAI schemes possess the parallelism of explicit schemes, i.e. the speed-up is restricted only by the function evaluation operations in (1).

Hence the MRAI approach seems to be a promising tool for parallel time stepping.

Acknowledgments We thank Gerard Sleijpen for careful reading the paper and suggesting several improvements.

References

[1] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA J. Numer. Anal.*, 14:563–581, 1991.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. A. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994. See also `http://www.netlib.org/templates/`.

[3] M. A. Botchev, G. L. G. Sleijpen, and H. A. van der Vorst. Stability control for approximate implicit time-stepping schemes with minimal residual iterations. Technical Report 1043, Department of Mathematics, Utrecht University, Dec. 1997.

[4] G. D. Byrne, A. C. Hindmarsh, and P. N. Brown. VODPK, large non-stiff or stiff ordinary differential equation initial-value problem solver. Available at `http://www.netlib.org`, 1997.

[5] T. F. Chan and K. R. Jackson. The use of iterative linear-equation solvers in codes for large systems of stiff IVPs for ODEs. *SIAM J. Sci. Stat. Comput.*, 7(2):378–417, 1986.

[6] Cray. *CRAY T3E Fortran Optimization Guide*, 1995. Cray manual SG-2518 3.0. Available at `http://soleil.rc.tudelft.nl:8080/`.

[7] Cray. *Message Passing Toolkit: MPI Programmer's Manual*, 1995. Cray manual SR-2197. Available at `http://soleil.rc.tudelft.nl:8080/`.

[8] E. de Sturler and H. A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *J. Appl. Num. Math.*, 18:441–459, 1995.

[9] 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.

[10] C. W. Gear and Y. Saad. Iterative solution of linear equations in ODE codes. *SIAM J. Sci. Stat. Comput.*, 4(4):583–601, 1983.

[11] A. C. Hindmarsh. LSODE: Livermore solver for ordinary differential equations. Available at `http://www.netlib.org`, 1987.

[12] R. W. Hockney. *The Science of Computer Benchmarking*. Software, Environments, Tools. SIAM, Philadelphia, PA, 1996.

[13] R. Keppens, G. Tóth, M. A. Botchev, and A. van der Ploeg. Implicit and semi-implicit schemes in the Versatile Advection Code: algorithms. *Int. J. Numer. Methods in Fluids*, Submitted, 1997.

[14] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986.

[15] B. P. Sommeijer, L. F. Shampine, and J. G. Verwer. RKC, a nearly-stiff ODE solver. Available at `ftp://cwi.nl/pub/bsom/rkc` and `http://www.netlib.org`, 1997.

[16] B. P. Sommeijer, L. F. Shampine, and J. G. Verwer. RKC: An explicit solver for parabolic PDEs. Technical Report MAS-R9715, CWI Amsterdam, 1997.

[17] G. Tóth, R. Keppens, and M. A. Botchev. Implicit and semi-implicit schemes in the Versatile Advection Code: numerical tests. *Astronomy and Astrophysics*, to appear, 1998.

[18] A. J. van der Steen. *Benchmarking of High Performance Computers for Scientific and Technical Computations*. PhD thesis, Utrecht University, Utecht, the Netherlands, 1997.

[19] H. A. van der Vorst and T. C. Chan. Linear system solvers: sparse iterative solvers. In D. E. Keyes, A. Samed, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*, volume 4 of *ICASE/LaRC Interdisciplinary Series in Science and Engineering*, pages 91–118, Dordrecht, 1997. Kluwer Academic.