A parallel nearly implicit time-stepping scheme[†]

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

To appear in *J. Comput. Appl. Math.*, 2001

Abstract

Across-the-space parallelism still remains the most mature, convenient and natural way to parallelize large scale problems. One of the major problems here is that implicit time stepping is often difficult to parallelize due to the structure of the system. Approximate implicit schemes have been suggested to circumvent the problem [5]. These schemes have attractive stability properties and they are also very well parallelizable.
The purpose of this article is to give an overall assessment of the parallelism of the method.
Key words: parallelism across the problem, parallel time integration, domain decomposition, speed up, stiff ODEs, GMRES.
2000 Mathematics Subject Classification: 65Y05, 65M20, 65M12. Secondary: 65F10, 68Q25.

1 Introduction

Over the last two decades, substantial progress has been made in the development of highly parallelizable methods for time-stepping integration. In Section 2 we give a brief survey of these methods. Across-the-space based parallelism, which is also called domain-decomposition or across-the-problem parallelism, still remains the most popular way to parallelize large scale problems. Our method, which is of the "across-the-space" type, is designed for parallel computing. In fact, it has the parallelism of an explicit scheme, yet, its stability properties are much better.

Large scale 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 accurate iterative schemes for the linear solves in implicit time-stepping codes may sometimes lead to a significant improvement in performance on a sequential computer (see e.g. [16, 10]), but it may as well lead to high CPU-times if the iterative schemes converge slowly. We will consider situations where stability is important, in particular where the time-step is restricted by stability constraints rather than by accuracy requirements.

[†]This research was supported by the Netherlands organization for scientific research NWO, project 95MPR04
[‡]Corresponding author. CWI, P.O.Box 94079, 1090 GB Amsterdam, the Netherlands. E-mail: botchev@cwi.nl, fax: (+3120) 592 4199.
[*]Mathematical Institute, Utrecht University, P.O.Box 80.010, 3508 TA Utrecht, the Netherlands. E-mail: vorst@math.uu.nl.

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 [28, 3], is attractive in this context. This combination is referred to as Minimal Residual Approximate Implicit (MRAI) time stepping [5]. The main difference with the conventional use of iterative techniques is that the number of iterative solution steps per time-step does not depend on the accuracy required for the time-stepping. Of course, this may lead to loss of stability, and therefore 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 a given implicit scheme. The resulting approximated implicit scheme can be interpreted as explicit and, hence, is not unconditionally stable. However, the minimum residual solver leads to a different explicit method per time-step and it turns out that the succession of different explicit solvers leads to improved global stability [5]. The stability control proposed in [5] allows for efficient automatic selection of the step size.

In fact the only difference between MRAI and classical explicit schemes is the inclusion of a minimum residual step in the former. The advantage of this approach is an intrinsically high parallelism of MRAI, at the price of only a modest loss in stability as compared with a fully implicit scheme. The aim of this article is to give an overall assessment of the parallelism of the method, including its minimum residual solver part.

The outline of our paper is as follows. Existing parallel time-stepping methods are briefly surveyed in Section 2. In Section 3 we describe the MRAI time-stepping scheme. In Section 4, we discuss parallel aspects of these schemes. Numerical experiments are presented in Section 5.

2 Parallel time stepping: a short overview

Here we discuss briefly various approaches for parallel time stepping. For more detailed surveys we refer to, e.g., [7, 8, 30, 35].

Three major types of parallelism are usually distinguished within parallel time-stepping schemes: parallelism across the space, parallelism across the method, and parallelism across the time (terminology introduced by Gear [35]).

Parallelism across the space, or domain decomposition based parallelism, is the most simple and widely used approach. However, across-the-space parallelization of implicit schemes is in general not straightforward. The reason is that the structure of the problem often inhibits efficient parallel solution of the implicit relations by linear direct solves. A naive domain decomposition parallelization is possible where for each of the subdomains an independent time stepping process is applied and the subdomains only exchange their boundary values. Such a technique is used in some applications but it may lead to problems with stability and with load balancing for the processors (see e.g. [39]).

Iterative methods can be successfully exploited for cases where the structure of the problem does not allow to perform efficient direct linear solves in parallel. Note that iterative linear solves may be attractive even on sequential computers [16, 10, 9, 29]. Very often preconditioning is indispensable to achieve high performance with iterative methods. For a survey of various parallel preconditioning techniques, see for instance [15, 3]. One problem with iterative methods is to decide when to stop the iterations. Too few as well as too many iterations

are not desirable, the first leads to instability, the second means a waste of CPU time. The MRAI scheme circumvents this problem.

Another type of parallelism—across the method—suggests a special time-stepping scheme where work for the solution on the next time level can be (partially) done in parallel [7, 8, 30, 35]. Independent of the size of the system, be it a single equation or a system of millions of ODEs, parallelism of the method is determined only by the selected scheme. In particular, in the class of Runge-Kutta and general linear methods one can identify a family of such parallel schemes. For the best methods of this type, one can expect performances comparable with those for the best sequential schemes on one processor and significant gain in parallel mode (see for instance experiments with the PSIDE code [14]): the speed up is less than the number of stages. A large number of stages is possible but would lead to schemes of higher accuracy and that is often not efficient.

Across-the-time (or across-the-steps) parallel methods form a relatively young family of parallel schemes. The idea behind the approach is to try to solve simultaneously at different time levels. For example, if a conventional implicit scheme is solved with an iterative method, one can proceed in time with the current iterative approximation without waiting until all the iterations are done. Because usually not many iterations are needed per time step, such time parallelism is restricted [35, 40]. Nevertheless, this restriction can be eliminated for special multigrid iterations. Two popular classes of parallel across-the-time multigrid methods are *multigrid waveform relaxation methods* [27, 23] and *parabolic multigrid methods* [17, 6]. Multigrid waveform relaxation methods possess nice speed-up properties both in time and space [24]; they are more robust than the parabolic multigrid method [38]. A serious drawback for both approaches, however, is that they require a substantial amount of problem-dependent programming. Both approaches also require much computer memory.

Another recently proposed class of implicit schemes possessing inherent parallelism are Krylov-solver-based *exponential integrators* [20, 21]. For certain class of the problems, such as highly oscillatory systems, these methods can be very successful. However, if a good preconditioning is available for the Jacobian matrix, the standard implicit time integration is usually more efficient [19, 26].

3 MRAI time stepping

Suppose that we are, due to stability considerations, interested in an implicit scheme for the solution of a stiff system of ODE's

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

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

$$\begin{aligned}(I-\tau J)(\boldsymbol{y}_{(1)}^{n+1}-\boldsymbol{y}_{(0)}^{n+1}) &= \tau \boldsymbol{f}(t_{n+1},\boldsymbol{y}_{(0)}^{n+1})+\boldsymbol{y}^n-\boldsymbol{y}_{(0)}^{n+1},\\ J &= \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_{n+1},\boldsymbol{y}_{(0)}^{n+1}),\end{aligned} \tag{3}$$

with $\boldsymbol{y}_{(0)}^{n+1}$ an initial guess, that has to be chosen carefully in order to have order consistency (see later).
In a Newton process this procedure is iteratively repeated until $\boldsymbol{y}_{(j)}^{n+1}$ is close enough to the wanted solution $\boldsymbol{y}^{n+1}$.

The basic idea behind the MRAI time stepping [5] is as follows: at each time step, for one or more Newton iterations, we solve (3) approximately with k steps of GMRES [28, 3]. 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 [5] shows that for a consistent scheme, an initial guess $\boldsymbol{y}_{(0)}^{n+1}$ for the iterative process has to be taken appropriately. For example, $\boldsymbol{y}_{(0)}^{n+1}$ 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 $\boldsymbol{y}_{(0)}^{n+1}$ from one step of an explicit scheme of the same (higher) order.

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, for instance $k=5$. 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 instability.

With only k GMRES steps we lose, of course, the unconditional stability and to monitor this loss, we need a step size control. We will now describe the MRAI step size control proposed in [5].
Let the initial vector $\boldsymbol{y}_{(0)}^{n+1}$ be computed with an Euler Forward step. First, k GMRES steps are performed for system (3). As a result, $k+1$ Krylov basis vectors, forming columnwise an orthogonal matrix V_{k+1}, and a small $(k+1)\times k$ projection matrix $\underline{\tilde{H}}$ are constructed, with

$$V_{k+1}(I-\tau J)V_k = \underline{\tilde{H}}.$$

It is easy to check that $\underline{\tilde{H}}$ is of the form

$$\begin{aligned}\underline{\tilde{H}} &= I-\tau \underline{H},\\ V_{k+1}JV_k &= \underline{H},\end{aligned} \tag{4}$$

which means that $\underline{H}$ is a projection matrix for J. It is also easy to show that the Krylov basis matrix V_{k+1} does not depend on τ.

Next, the step size control is applied. Let $\tilde{H}$ denote the matrix $\underline{\tilde{H}}$ of which the last row is skipped. As it has been argued in [5], the scheme is stable provided that the smallest

eigenvalue $\lambda_{\min}$ of matrix $\tilde{H}^{-T}(\underline{\tilde{H}}^T \underline{\tilde{H}})$ satisfies

$$\lambda_{\min} \leqslant 8.$$

In fact, the eigenvalues of $\tilde{H}^{-T}(\underline{\tilde{H}}^T \underline{\tilde{H}})$ increase monotonically with increasing τ, so we are looking for a τ such that function $\lambda_{\min}(\tau)$ remains below 8 but not too small, since we want a τ as large as possible. For a proper τ we simply solve a scalar equation

$$\lambda_{\min}(\tau) = 8 - \varepsilon, \qquad 0 < \varepsilon \ll 1 \tag{5}$$

numerically, for instance with the secant iterative method. Each iteration involves computation of $\underline{\tilde{H}}$ for a new trial value of τ (cf. (4)) and the solution of a small eigenvalue problem for the matrix $\tilde{H}^{-T}(\underline{\tilde{H}}^T \underline{\tilde{H}})$. This can be implemented with LAPACK [1]. Note that we do not have to solve (5) accurately because any τ for which $\lambda_{\min} \leqslant 8$ guarantees stability. Usually a practical value of τ is found with three secant iterations.

We emphasize that the overall work for the step size control is proportional to k^3 only. Moreover, if in (3) we take $\boldsymbol{f}(t_n, \boldsymbol{y}^n)$ instead of $\boldsymbol{f}(t_{n+1}, \boldsymbol{y}^{n+1}_{(0)})$, which does not reduce the order of the scheme, a new good value of τ can be immediately applied for the current step. A small adjustment in the order of computations is then needed: we compute the initial vector $\boldsymbol{y}^{n+1}_{(0)} = \boldsymbol{y}^n + \tau \boldsymbol{f}(t_n, \boldsymbol{y}^n)$ already when a good τ has been found and, at the beginning of the step, we start the GMRES process with vector $J\boldsymbol{f}(t_n, \boldsymbol{y}^n)$. (Note that the residual for $\boldsymbol{y}^{n+1}_{(0)}$, substituted in (3), is $\tau^2 J\boldsymbol{f}(t_n, \boldsymbol{y}^n)$).

A second order MRAI scheme with a similar economical step size control is described in [5]. For other higher order schemes it is not always possible to apply a recent value of τ immediately on the current time step while still avoiding work of order N. But, of course, it is always possible to apply the computed τ for the next time step, and this works well in practice.

Numerical tests and comparisons of the MRAI schemes with other time-stepping strategies (as in [32, 9]) can be found in [4, 5]. The MRAI time stepping has been used with success in the general purpose MHD solver VAC [25, 33].

Note that if the Jacobian J is not available explicitly, then its action on a vector is approximated by the directional difference,

$$J(t_{n+1}, \boldsymbol{y}^{n+1}_{(j)})\boldsymbol{v} \approx \frac{\boldsymbol{f}(t_{n+1}, \boldsymbol{y}^{n+1}_{(j)} + \epsilon \boldsymbol{v}) - \boldsymbol{f}(t_{n+1}, \boldsymbol{y}^{n+1}_{(j)})}{\epsilon}, \qquad \epsilon = \frac{\sqrt{\delta}\, \boldsymbol{v}^T \boldsymbol{y}^{n+1}_{(j)}}{\|\boldsymbol{v}\|}, \tag{6}$$

where δ is the floating point relative machine accuracy.

4 Parallel aspects

The structure of MRAI with respect to parallelism is fairly simple. There are two CPU-time intensive components

1. function evaluations with $\boldsymbol{f}$ (FEVALs);

Table 1: Speed-up for modified and classical Gram-Schmidt on the Cray T3E for the orthogonalization part of GMRES(7); $N = 80\,000$, PE stands for "processor element".

# of PEs	modified Gram-Schmidt (35+1 communications)	classical Gram-Schmidt (7+1 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

2. the GMRES part.

Parallelization of GMRES has been well studied (see e.g. [3, 37, 15]). The main problem is the communications for the inner products but, as we see from Table 1, this part of the algorithm scales perfectly well for low values of k. For larger values of k, other tricks for the inner products are possible if the vector lengths are not large enough [2, 13].

Now we consider in some more detail the parallelism of the complete MRAI scheme for ODE's where the right-hand side is a partial differential operator. In this case, we will assume that (6) is used for the evaluation of Jacobians. Let T_p denote the CPU time required to advance one time step with the MRAI scheme in parallel on p processor elements (PEs). We will derive estimates for the speed-up $S_p = T_1/T_p$.

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 that is spent for the inner products. As we have seen, this part of the computations is well parallelizable. 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 CPU-time, with $0 \leqslant f \leqslant 1$, to compute all of them by one PE. The remainder of T_1 is necessary for $k+2$ FEVALs, each of which takes t_1^{feval} by 1 PE. This means that

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

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

$$T_p = \frac{fT_1}{p} + (k+2)t_p^{\mathsf{feval}}. \tag{7}$$

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

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

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

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

$$S_p = \frac{p}{1 + (1-f)p\, t_p^{\mathsf{comm}}/t_1^{\mathsf{feval}}}. \tag{8}$$

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

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

$$1-f = \frac{(k+2)F}{k(k+1)/2+2k+1+(k+2)F}. \tag{9}$$

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

$$1-f = \frac{7F}{26+7F}. \tag{9'}$$

We now consider the situation that corresponds to the model problem described in the next section. We will specify the values t_p^{comm} and F in the expressions (8),(9). It will be clear from the presentation how the speed-up analysis can be applied for other cases.

Suppose that (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 grid nodes in one direction. Assume for simplicity that $N=n^3$ and $p=\sqrt{p}\times\sqrt{p}$, with $\sqrt{p}$ an integer. 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

$$t_p^{\mathsf{comm}} = c_1\frac{N^{2/3}}{\sqrt{p}} + c_2, \tag{10}$$

where c_1 and c_2 are computer dependent 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 a factor of 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 a factor of two. As we see, the communication time decreases as the number of PEs increases. According to Table 2, this is also the case for other discretized PDEs, provided that the PEs are logically organized in square 2D or cubic 3D grids.

Because the seven-point stencil leads to at least 7 multiplications and 6 additions for the FEVAL operation at each grid point, the FEVAL expenses are $F \geqslant 6.5$. We took $F=7$. With (9') we get $1-f \approx 0.7$. Substitution of this, in combination with (10), into the speed-up estimate (8) leads to

$$S_p = \frac{p}{1+0.7\,\dfrac{c_1N^{2/3}\sqrt{p}+c_2p}{t_1^{\mathsf{feval}}}}. \tag{11}$$

To determine c_1 and c_2, we have run a simple code with a single call to the FEVAL subroutine, where t_p^{comm} is measured explicitly. The code has been executed twice, with different numbers

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	$\mathrm{const}(N;p)$
2D	1D	$\sim \sqrt{N}, \quad \mathrm{const}(p)$
2D	2D	$\sim \sqrt{N}, \quad \sim \frac{1}{\sqrt{p}}$
3D	1D	$\sim N^{2/3}, \quad \mathrm{const}(p)$
3D	2D	$\sim N^{2/3}, \quad \sim \frac{1}{\sqrt{p}}$
3D	3D	$\sim N^{2/3}, \quad \sim \frac{1}{p^{2/3}}$

p, and this resulted in a system of two equations in c_1 and c_2. The value of t_1^{feval} can also be measured directly. Such a direct timing has the 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, it is safer to redo the timings for a new value of N.

However, it is often reasonable to assume that the performance depends only mildly on N, so that t_1^{feval} is directly proportional to the problem size: $t_1^{\mathsf{feval}} = c_3 N$, c_3 a constant. Substitution of the last expression into (11) gives an explicit dependence of the speed-up on the problem size N and the number p of PEs:

$$S_p(N;p) = \frac{p}{1 + 0.7\,\dfrac{c_1 N^{2/3}\sqrt{p} + c_2 p}{c_3 N}}. \tag{12}$$

In Figure 1, we have depicted the dependence (12) 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%, N should be at least 10^5 for $p = 16$, and $7{\cdot}10^5$ for $p = 64$.

To adapt the speed-up estimates for a different problem (i.e. for a different FEVAL), one has only to estimate the FEVAL expenses according to (9), and adjust the communication time expression (10) (see Table 2).

We note that the estimate (10), and similar estimates as presented in Table 2, can be reformulated in terms of the hardware parameters r_∞^c and t_0^c (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 further information, we refer the reader to [22, 36].

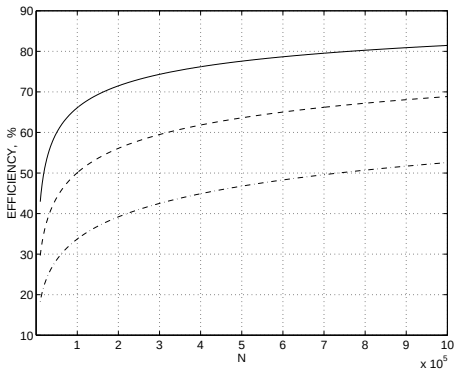

Figure 1: Ratio (MRAI speed-up) / (ideal speed-up)·100% versus the problem size N on the IBM SP2 for different number of PEs (solid line—$p = 4$, dashed line—$p = 16$, dash-dotted line—$p = 64$)

5 MRAI performance

In this section we test how the MRAI strategy competes with other time-stepping techniques on a sequential computer.

In [5], we presented numerical experiments with 2D and 3D problems demonstrating superiority of the MRAI-based Euler Backward (EB) scheme tested against EB with direct and iterative linear solvers. In these experiments, different stopping criteria for the iterative solver were tried and, for the whole range of the stopping criteria, the iteration-based EB performed worse than the MRAI-based EB scheme.

The fact that for grid based 2D and 3D problems MRAI strategy outperformed implicit time stepping with direct linear solves can be explained as follows. An attractive feature of the direct methods is that the LU factors can typically be reused for several time steps. According to the estimates in [10], 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 $\mathcal{O}(N^{4/3})$ flops to this amount. Let us assume that for the corresponding MRAI scheme the step size is in average 20 times smaller than the fully implicit variant of which the MRAI is derived (which is in practice often a pessimistic estimate for MRAI), and that an LU factorization is made only once per 10 time steps. Even for this strongly biased, in favor of direct methods, situation (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) 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. Moreover, direct sparse methods are much more difficult to parallelize [15], so that the picture will be even less favorable for them on a parallel computer.

Here we present tests where an experimental MRAI-modified stiff ODE solver LSODE (the LSODE / MRAI code) is compared with RKC [32, 31] and VODPK [9] codes.

The LSODE code is a black-box stiff integrator [18], in which the variable-order implicit backward differentiation formulas are used with a Newton process, and the inner linear solves

are done by direct methods from LINPACK. In the MRAI version of the code, linear solves have been replaced by a fixed number of GMRES steps, and the Jacobian evaluation (i.e. the Jacobian action on a vector) can be done according to (6). These techniques are similar to those employed in the VODPK code [9], 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).

Combination of two step-size control mechanisms in the LSODE / MRAI code, namely the MRAI stability step-size control and the LSODE accuracy step-size control, often leads to a too stringent control. Therefore, in the LSODE / MRAI code, the MRAI step-size control is incorporated in a relaxed form: only when the actual step size exceeds the step size Δt_{MRAI} suggested by MRAI by factor 5 or more, the allowable step size is restricted by $5\Delta t_{\mathrm{MRAI}}$.

RKC is based on Runge-Kutta-Chebyshev formulas [34] and is intended for mildly stiff problems with real-spectrum Jacobians. The code needs at most 7 vectors of storage (typically 2-3 times less than for VODPK or LSODE / MRAI) and thus is especially attractive when the memory requirements are an issue.

5.1 3D heat conduction

This test problem is a linear heat conduction problem over the 3D unit cube [32]. The inhomogeneous term is chosen in order to have the analytical solution $\tanh(5(x + 2y + 1.5z - 0.5 - t))$ which provides initial and boundary conditions for the test. The spatial discretization with central differences on a grid of $79 \times 39 \times 39$ yields a system of $N = 120\,159$ equations. The integration was done for $0 \leqslant t \leqslant t_{\mathrm{END}} = 5.0$.

The LSODE / MRAI code was used with matrix-free Jacobian evaluation. In this mode the code required 16 N-vectors to store (6 of which for the MRAI part).

We used VODPK in two modes: with diagonal scaling preconditioning (D-VODPK) and without preconditioning. All VODPK parameters were set to the default values. The code required than 18 N-vectors of storage.

For the RKC code, all the parameters were set to their default values, except that we explicitly told the code an estimate for the spectral radius of the Jacobian (this is not crucial for the performance of the code, however). The RKC code required only 4 N-vectors to store.

The results of comparative runs on one processor of the SGI Origin 2000 are presented in Table 3 and on Figure 2. The absolute and relative tolerances are equal and given by parameter `tol`. In the Table, columns "error", "CPU" and "fevals/steps" contain the maximum difference in computed and exact solution at $t = t_{\mathrm{END}} = 5.0$, CPU time in seconds, number of calls to the right hand side function $\boldsymbol{f}$ and steps made, respectively. The error reported in the Table is measured with respect to a reference solution computed with a stricter tolerance [32].

As can be seen from the results presented, the LSODE / MRAI code is the best for not stringent tolerance requirements `tol` $\geqslant 10^{-3}$, when stability is of more concern than accuracy. For these tolerances, LSODE / MRAI outperforms even preconditioned VODPK (note that LSODE / MRAI is used without preconditioning).

For more stringent tolerances performance of LSODE / MRAI is less impressive though still better than performance of preconditioned VODPK. We note that the LSODE / MRAI code can be tuned to have a better performance for more stringent tolerances; this can be achieved

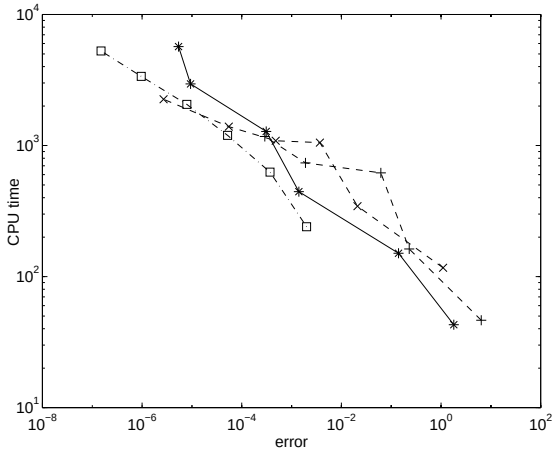

Figure 2: A log-log plot of CPU time versus error for the 3D heat conduction Example (the $*$-line is LSODE/MRAI, the $+$-line is preconditioned VODPK, the $\times$-line is VODPK, and the $\square$-line is RKC).

by making the MRAI stability step size control more stringent for higher accuracy requirements.

The RKC code appears to be the best when high accuracy is needed. However, for relaxed tolerances RKC has serious difficulties. That the problem is too stiff for RKC can be seen from the number of used internal stages reported by the code [32, 31]: for `tol` $= 10^{-1}$ number of internal stages in RKC exceeds 200.

5.2 Testing parallel performance of MRAI

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 one is the LSODE / MRAI code described in the previous section. This choice is quite representative since EB is a simple implicit scheme (still actively used in practice), whereas LSODE is a quite advanced code based on higher-order schemes.

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 is not applicable for problems with Jacobians with complex spectrum (as, for example, in advection-diffusion problems). The VODPK concept is specially developed for the inexact Newton method framework, to be applied for higher order time stepping. Our simpler MRAI approach is of interest for a wider class of schemes.

Our model problem is the 3D heat equation problem from the previous section. For the same standard seven-point stencil finite difference discretization, the spatial grid $40 \times 40 \times 40$ leads to a system of size $N = 64\,000$. The numerical integration was done for $t \in [0, 0.7]$. In those runs, speed-ups of which are presented below, the tolerance parameter in the

Table 3: Results for the 3D heat conduction Example

code	error	CPU	fevals/steps	error	CPU	fevals/steps
	tol $= 10^{-1}$			tol $= 10^{-4}$		
LSODE/ MRAI	1.8	42.9	307/25	$3.1\,10^{-4}$	1280	10567/606
D-VODPK	6.4	46.4	322/25	$1.9\,10^{-3}$	737	5001/570
VODPK	1.1	116.7	760/79	$4.8\,10^{-4}$	1087	7449/876
RKC	$2.0\,10^{-3}$	240	2687/22	$7.9\,10^{-6}$	2062	9716/338
	tol $= 10^{-2}$			tol $= 10^{-5}$		
LSODE/ MRAI	0.14	151	1165/75	$9.4\,10^{-6}$	2995	24373/1959
D-VODPK	0.23	163	1120/77	$2.9\,10^{-4}$	1172	7848/1050
VODPK	$2.1\,10^{-2}$	344	2296/233	$5.5\,10^{-5}$	1395	8798/1062
RKC	$3.7\,10^{-4}$	627	4340/61	$9.6\,10^{-7}$	3371	14638/763
	tol $= 10^{-3}$			tol $= 10^{-6}$		
LSODE/ MRAI	$1.4\,10^{-3}$	445	3427/204	$5.4\,10^{-6}$	5693	45979/6070
D-VODPK	$6.2\,10^{-2}$	619	4228/399	code failed		
VODPK	$3.7\,10^{-3}$	1052	7126/716	$2.7\,10^{-6}$	2255	12903/1525
RKC	$5.2\,10^{-5}$	1195	6375/143	$1.5\,10^{-7}$	5270	21197/1557

LSODE/MRAI code was chosen as 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 technique described in Section 3.

For our model problem, we have parallelized the LSODE/MRAI and EB/MRAI codes using the MPI communication library [12]. The 3D grid was distributed among the PEs in two dimensions, so that each PE has 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 (11), with the estimated parameters c_1, c_2 (Section 4), which were

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

We estimated the parameter F (cf. (9)) 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 4 and in Figure 3. Exactly the same codes have been executed on the Cray T3E and IBM SP2, but, as we see, the speed-ups for the IBM SP2

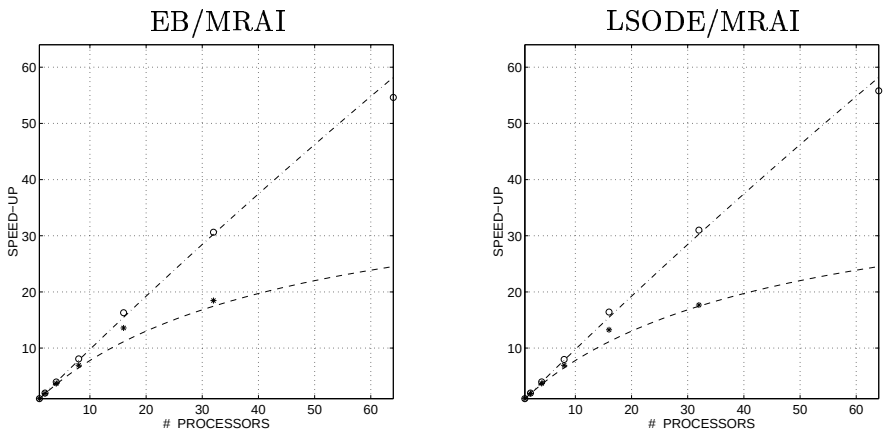

Figure 3: Speed-up results for the Cray T3E (predicted: -·-, observed: o) and IBM SP2 (predicted: ---, observed: *)

Table 4: 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	—

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 (10), 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 less than codes based on the Cray's native communication library SHMEM [11]).

6 Conclusions

The recently proposed MRAI time stepping approach can be viewed as an attempt to obtain a parallelizable cheap alternative for implicit schemes, preserving stability properties as much

as possible [5].

Experiments on the Cray T3E and the IBM SP2 parallel computers and analysis 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 an attractive tool for parallel time stepping for the situations where the step size is restricted by stability rather than accuracy.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK user's guide.* SIAM, Philadelphia, PA, 2 edition, 1995. URL http://www.netlib.org/lapack/.

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

[3] 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. URL http://www.netlib.org/templates/.

[4] 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. Available at URL http:// www.math.uu.nl/publications/.

[5] M. A. Botchev, G. L. G. Sleijpen, and H. A. van der Vorst. Stability control for approximate implicit time stepping schemes with minimum residual iterations. *Appl. Numer. Math.*, 31(3):239–253, 1999.

[6] J. Burmeister and G. Horton. Time-parallel multigrid solution of the Navier-Stokes equations. In W. Hackbusch and U. Trottenberg, editors, *Multigrid methods, III (Bonn, 1990)*, pages 155–166. Birkhäuser, Basel, 1991.

[7] K. Burrage. *Parallel and sequential methods for ordinary differential equations.* The Clarendon Press Oxford University Press, New York, 1995. Oxford Science Publications.

[8] K. Burrage, editor. *Parallel methods for ODEs.* Baltzer Science Publishers BV, Amsterdam, 1997. Adv. Comput. Math. **7** (1997), no. 1-2.

[9] 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 URL http://www.netlib.org, 1997.

[10] 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.

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

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

[13] 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.

[14] J. J. de Swart. *Parallel Software for Implicit Differential Equations*. PhD thesis, University of Amsterdam, Nov. 1997. ISBN 90-74795-77-3.

[15] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, PA, 1998.

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

[17] W. Hackbusch. Parabolic multigrid methods. In *Computing methods in applied sciences and engineering, VI (Versailles, 1983)*, pages 189–197. North-Holland, Amsterdam, 1984.

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

[19] M. Hochbruck. Exponential integrators. Oral presentation at the NUMDIFF'2000, Halle, www.mathematik.uni-halle.de/~numdiff/, 2000.

[20] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 34(5):1911–1925, Oct. 1997.

[21] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput.*, 19(5):1552–1574 (electronic), 1998.

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

[23] G. Horton and S. Vandewalle. A space-time multigrid method for parabolic partial differential equations. *SIAM J. Sci. Comput.*, 16(4):848–864, 1995.

[24] G. Horton, S. Vandewalle, and P. Worley. An algorithm with polylog parallel complexity for solving parabolic partial differential equations. *SIAM J. Sci. Comput.*, 16(3):531–541, 1995.

[25] 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*, 30:335–352, 1999.

[26] C. Lubich. Exponential integrators. Oral presentation at the Woudschoten Conference of the Dutch-Flemish Numerical Analysis Community, Zeist, 2000.

[27] C. Lubich and A. Ostermann. Multigrid dynamic iteration for parabolic equations. *BIT*, 27(2):216–234, 1987.

[28] 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.

[29] B. A. Schmitt and R. Weiner. Matrix-free W-methods using a multiple Arnoldi iteration. *Appl. Num. Math.*, 18(1–3):307–320, 1995.

[30] B. P. Sommeijer. *Parallelism in numerical integration of initial-value problems.* PhD thesis, University of Amsterdam, 1992. Also published as CWI tracts. 99, Amsterdam, CWI.

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

[32] B. P. Sommeijer, L. F. Shampine, and J. G. Verwer. RKC: An explicit solver for parabolic PDEs. *J. Comput. Appl. Math.*, 88:315–326, 1997.

[33] G. Tóth, R. Keppens, and M. A. Botchev. Implicit and semi-implicit schemes in the Versatile Advection Code: numerical tests. *Astronomy and Astrophysics*, 332:1159–1170, 1998.

[34] P. van der Houwen and B. P. Sommeijer. On the internal stability of explicit m-stage Runge–Kutta methods for large values of m. *Z. Angew. Math. Mech.*, 60:479–485, 1980.

[35] P. J. van der Houwen. Parallel aspects of integration methods for initial value problems. In *Special functions and differential equations (Madras, 1997)*, pages 403–431. Allied Publ., New Delhi, 1998.

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

[37] 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.

[38] S. Vandewalle and G. Horton. Fourier mode analysis of the multigrid waveform relaxation and time-parallel multigrid methods. *Computing*, 54(4):317–330, 1995.

[39] J. G. Verwer, W. Hundsdorfer, and J. G. Blom. Numerical time integration for air pollution models. CWI Report MAS-R9825, Amsterdam, 1998. To appear in "Surveys for Mathematics in Industry".

[40] D. E. Womble. A time-stepping algorithm for parallel computers. *SIAM J. Sci. Statist. Comput.*, 11(5):824–837, 1990.