Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

P.J. van der Houwen, B.P. Sommeijer

Parallel ODE solvers

Department of Numerical Mathematics Report NM-R9008 April

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Parallel ODE Solvers

P.J. van der Houwen & B.P. Sommeijer
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1980 Mathematics Subject Classification: 65M10, 65M20
1982 CR Categories: 5.17
Key Words and Phrases: Parallelism, ordinary differential equations, stability, diagonally implicit Runge-Kutta methods, Richardson-extrapolation.

1. INTRODUCTION

In August 1987 C.W. Gear wrote in his survey paper [9] on parallel methods for solving the initial-value problem (IVP) for ordinary differential equations (ODEs): "Compared to the enormous amount of work done on parallelism in other areas of scientific computation, relatively little has appeared on ODEs". In spite of the increasing popularity of parallel computers for scientific computing, this situation has not been changed in the last three years. The main reason is that the greater part of the computational intensive IVPs originate from the semi-discretization of partial differential equations, that is, the system of ODEs is usually extremely large.

When using *explicit* IVP methods, a straightforward approach is offered by segmenting the equations into separate subsystems, still of considerable size, which are to be handled by the various processors. This form of parallelism inherent to the problem is called *parallelism across the problem.* In the case of *implicit* IVP solvers, it is the linear algebra problem associated with the implicit relations to be solved, that requires the bulk of the computational work. Of course, a lot of research has been done in developing parallel linear algebra methods which plays a central role in parallel ODE solvers, but the achievements in this area are not considered as contributions to the ODE literature.

Orthogonal to the two forms of parallelism indicated above, one may think of *parallelism across the ODE method.* However, most methods for solving IVPs are by nature step-by-step methods and leave little scope for this form of parallelism. An alternative to this step-by-step approach is given by Bellen et al. [2], but it appears that these methods are efficient only when implemented on a (very) large number of processors. Thus, in order to achieve parallelism across the ODE method and thereby a further speed-up of the integration process, new algorithms have to be designed. This is a challenge to the numerical analyst and requires an approach which differs from the usual approach in developing sequential algorithms.

Report NM-R9008
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In designing parallel ODE methods, we shall distinguish two categories of methods: (i) Methods with reduced 'wall-clock time' (turn-around time) per step, but producing numerical solutions with comparable quality when compared with existing 'sequential' methods (i.e., methods without parallelism across the method). When using such 'wall-clock time reducers', one should be prepared to accept a certain amount of redundancy, that is, one should not require that the wall-clock time of the parallel algorithm on κ processors equals T/κ, where T is the wall-clock time on one processor of the 'best' sequential algorithm producing a comparable numerical result. (ii) Methods with improved quality of the numerical solution, without increasing the wall-clock time per step in comparison with numerical solutions of the same quality furnished by sequential methods. However, improved quality (for example, methods of higher order of accuracy or with larger stability regions) should result in larger stepsizes, so that the ultimate effect of 'quality improvers' is again a reduction of the (total) wall-clock time.

In this contribution, we shall concentrate on parallel, one-step ODE solvers for integrating the first-order IVP

$$(1.1)\quad y'(t) = f(t,y(t)),\ \ y(t_0)=y_0,\ \ t_0 \le t \le T_{end}.$$

We shall discuss two approaches of constructing parallel ODE methods that are more efficient (in terms of wall-clock time) than the conventional methods. These approaches are based on extrapolation of basic ODE solvers and on iteration of implicit Runge-Kutta (RK) methods.

In order to facilitate a comparison of the parallel methods presented in this paper with sequential and already available parallel methods, we list below the 'best' sequential methods belonging to two popular families of one-step ODE solvers.

Explicit Runge-Kutta methods. In the case of explicit RK methods of order p, it is the number of stages (or: righthand side evaluations) per step that is the main characteristic for the efficiency of the method. The minimal number of stages k of methods actually constructed is listed in Table 1.1. Parallel versions of such methods have been discussed in [20], [21], [24] and [26]. In Subsection 3.2.1, we shall present the parallel methods derived in [16] and in Subsections 2.2.1 and 2.2.3, we present a new family of parallel explicit RK methods. These methods are designed

with the purpose to achieve a given order with less stages per step than listed in Table 1.1.

Table 1.1. Sequential explicit RK methods.

Order p:	≤4	5	6	7	8	9	10
Stages k:	p	p+1	p+1	p+2	p+3	-	p+7

Diagonally-implicit Runge-Kutta methods. For stiff ODEs we need highly stable methods which leads us automatically to implicit methods. Within the class of implicit RK methods, the singly diagonally-implicit RK (SDIRK) methods are most popular, because the dimension of the implicit systems to be solved is equal to the dimension of the system of ODEs and all systems can be solved using Newton-Raphson iteration with the same Jacobian matrix and LU-decomposition. The literature provides only SDIRK methods of order p≤4 of which a few examples are listed in Table 1.2.

Table 1.2. SDIRK methods.

Order	Stages	Stability	Reference
p=3	p-1	A-stable	Nørsett [25]
p=3	p-1	Strongly A-stable	Crouzeix [4]
p=4	p-1	A-stable	Crouzeix [4], Alexander [1]
p=4	p	L-stable	Iserles & Nørsett [20]

In this table, we also added the two-processor method constructed by Iserles and Nørsett. The sequential costs of these methods are half the number of stages (as far as we know they are the first parallel SDIRK methods constructed in the literature). Until 1989 SDIRK methods (either sequential or parallel) of order higher than 4 were not yet available. This motivated us to develop parallel versions of order higher than four to improve the quality of this type of method (see Subsections 2.2.2, 2.2.4, 3.2.2 and 3.2.3).

2. METHODS BASED ON EXTRAPOLATION

Many times it has been remarked that extrapolation methods possess a high degree of parallelism and offer an extremely simple technique for generating high-order methods (cf., e.g. Deuflhard [6], Hairer [13], Gladwell [10]). First, we shall summarize the basic properties of Richardson extrapolation (see, e.g., Hairer, Nørsett and Wanner [15]), and we shall specify a Romberg sequence that is suitable for use on parallel computers. Since Richardson extrapolation is not restricted to IVP solvers, we consider more generally extrapolation of numerical approximations $u(\Delta)$ to obtain approximations to the true solution value u_{true} with increased order of accuracy. The application to IVP solvers is given in the Section 2.2.

2.1. Richardson extrapolation

Suppose that

$$(2.1)\quad u(\Delta) = u_{true} + C(\Delta)\,\Delta^q,\ q \geq 1,$$

where $C(\Delta)$ is some (unknown) function of Δ with $C(0)\neq 0$. Δ denotes a control parameter that determines the accuracy of the approximation.
Computing the numerical approximations $u(\Delta/m_i)$, $i=1,\ldots,r$, where the m_i are distinct, increasing, positive numbers, we can define the numerical extrapolation

$$(2.2)\quad u_{Rich}(\Delta) := \sum_{i=1}^{r} c_i\, u\left(\frac{\Delta}{m_i}\right),$$

where the c_i are to be determined such that $u_{Rich}(\Delta)$ is more accurate than $u(\Delta)$. Evidently, the approximations $u(\Delta/m_i)$ can all be computed in parallel and therefore extrapolation formulas are ideal for parallel execution. The sequence $\{m_i\}$ represents the *Romberg sequence*, and the methods providing $u(\Delta)$ and $u_{Rich}(\Delta)$ respectively represent the generating method and the extrapolation formula. Since we can fix one of the numbers m_i in advance, we may assume without loss of generality that $m_1=1$.
If the numerical solution $u(\Delta)$ is sufficiently smooth, so that $C(\Delta)$ possesses a Taylor expansion in Δ, then the coefficients c_i can be chosen such that the order of the extrapolation formula (2.2) equals $q^*=q+r-1$. The corresponding equations for the coefficients c_i are

$$(2.3)\quad \sum_{i=1}^{r} c_i = 1,\quad \sum_{i=1}^{r} \frac{c_i}{(m_i)^j} = 0,\ j = q, \ldots, q+r-2.$$

In the particular case where $q=1$, the value of $u_{Rich}(\Delta)$ can be generated by the Aitken-Neville recursion

$$T_{i,1} := u\left(\frac{\Delta}{m_i}\right),\ i = 1, 2, \ldots, r;$$

$$(2.4)\quad T_{i,j} = T_{i,j-1} + \frac{T_{i,j-1} - T_{i-1,j-1}}{\frac{m_i}{m_{i-j+1}} - 1},$$

$$j = 2, 3, \ldots, r;\ i = j, j+1, \ldots, r$$

and is obtained by setting

$$u_{Rich}(\Delta) = T_{r,r}.$$

Since we aim at extrapolation formulas that are suitable for use on parallel computers, we have to make assumptions on the computational complexity of the numerical approximation $u(\Delta)$. Throughout, we shall assume that the computational effort involved in computing $u(\Delta)$ is essentially proportional to $\Delta^{-1/s}$. For example, in the case of IVP solvers, we have $\Delta=h^s$, where h is the stepsize, and s=2 for symmetric methods and s=1 otherwise. Usually, the computational effort of applying an IVP solvers is proportional to $1/h=\Delta^{-1/s}$. Thus, given κ processors, we should partition the tasks of computing the approximations $u(\Delta/m_i)$ such that they roughly get the same amount of

work, that is, we have to specify κ sets M_j of integers i which minimize the sum S defined by

$$S := \max \{S_1, S_2, \ldots, S_\kappa\}, \quad S_j := \sum_{i \in M_j} (m_i)^{1/s}.$$

Table 2.1. Distribution of $u(\Delta/i^s)$ over parallel processors for the Romberg sequence $\{i^s\}$ in terms of the index sets M_j.

Processors	I	II	III	IV
r = 2	{1}	{2}		
r = 3	{1 , 2}	{3}		
r = 4	{1 , 4}	{2 , 3}		
	{1 , 3}	**{2}**	**{4}**	
r = 5	{1 , 2 , 5}	{3 , 4}		
	{1 , 4}	**{2 , 3}**	**{5}**	
r = 6	{1 , 4 , 6}	{2 , 3 , 5}		
	{1 , 6}	{2 , 5}	{3 , 4}	
	{1 , 5}	**{2 , 4}**	**{3}**	**{6}**
r = 7	{1 , 6 , 7}	{2 , 3 , 4 , 5}		
	{1 , 3 , 6}	{2 , 7}	{4 , 5}	
	{1 , 6}	**{2 , 5}**	**{3 , 4}**	**{7}**

In many applications, the approximations $u(\Delta/m_i)$ are only defined for integer values of $(m_i)^{1/s}$. This leads us to set $(m_i)^{1/s}=i$, by which we achieve that the largest value of the m_i, i.e. the value of m_r, is as small as possible, so that the computational work involved in computing the most expensive $u(\Delta/m_i)$ is minimized. The optimal partitioning of the evaluations of the r quantities $u(\Delta/i^s)$ over κ processors is listed in Table 2.1. It is easily shown that the minimal number of processors such that S=r is given by $\kappa=[(r+2)/2]$, where, [.] denotes the integer-part function. The corresponding index sets are defined by (indicated by bold face in Table 2.1)

$$(2.5) \quad M_j := \{j , r-j\}, \; j = 1, 2, \ldots , \kappa-1; \; M_\kappa := \{\kappa\}.$$

2.2. Application to IVP solvers

We now apply Richardson extrapolation to IVP solvers. It will be assumed that we are given a method of order p for integrating (1.1) from t_0 until $t_1 := t_0+H$ with (not necessarily uniform) stepsizes which depend on a parameter h. The numerical approximation to the exact solution value $y(t_0+H)$ will be denoted by $y(t_0+H,h)$. The method producing this approximation will be called the *generating method* and $y(t_0+H,h)$ will be called the *generating function*.

If the generating function possesses an asymptotic expansion in powers of h^s, then we may identify $y(t_0+H,h)$ with the function $u(\Delta)$ and Δ with h^s. Here, s=2 if the method providing the values $y(t_0+H,h)$ is a symmetric method, and s=1 otherwise. The values $u(\Delta/m_i)$ in the extrapolation formula (2.2) are defined by

$$(2.6) \quad u\left(\frac{\Delta}{m_i}\right) := y\left(t_0 + H, \frac{h_0}{(m_i)^{1/s}}\right) = y\left(t_0+H, \frac{h_0}{i}\right).$$

By means of the extrapolation formula defined by {(2.2),(2.3)} we can extrapolate the r values $y(t_0+H,h_0/i)$ to establish the first step of the extrapolation method:

$$(2.7) \quad y_1 = \sum_{i=1}^{r} c_i \, y\left(t_0 + H, \frac{h_0}{i}\right).$$

Evidently, y_1 approximates y(t) at the point t_1. Having computed y_1, we can perform a second step by using y_1 as the new initial value at t_1, etc.. The quantities h_0 and H are called the *internal* and *basic* stepsizes, respectively.

Theorem 2.1. Let the generating method providing the values $y(t_0+H,h_0/i)$ be of order p, then the extrapolation method defined by (2.7) has order $p^*=p+s(r-1)$. □

We remark that the r grids in the interval $[t_0,t_0+H]$ associated with the r values $y(t_0+H,h_0/i)$ often contain common gridpoints $t_{1,j}$, and that the numerical values obtained at $t_{1,j}$ again can be thought of being derived from some generating function possessing an asymptotic expansion. In such cases, we may compute at each common point $t_{1,j}$ additional numerical solution values of order $p^*=p+s(r-1)$.

Two often used versions of the above extrapolation process set $H=h_0$ and $H=T_{end}-t_0$. They are respectively called *local extrapolation* and *global extrapolation*. In the case of global extrapolation, the stability of the integration process is determined by the stability properties of the generating method, whereas the stability of the local extrapolation method also depends on the extrapolation formula (2.7).

If we want to specify the computational effort of extrapolation methods on parallel computers we need the notion of the number of sequential (or effective) stages associated with numerical results. The following definition parallels a similar definition dealing with the number of sequential stages of explicit RK methods (cf. Iserles and Nørsett [20]):

Definition 2.1. Let the numerical result produced by a numerical method require the computation of k righthand side functions and let the required wall-clock time on κ processors be equal to T_κ, then this numerical result is said to require kT_κ/T_1 *sequential* (or *effective*) stages. □

Assuming that the computational complexity of computing $y(t_0+H,h)$ is proportional to 1/h, we see that the sequential costs per step of length H of the method {(2.5),(2.7)} on [(r+2)/2] processors equals that of computing $y(t_0+H,h_0/r)$. Furthermore, if $y(t_0+H,h_0/r)$ requires k righthand side functions, then the number of sequential stages needed for integrating from t_0 until T_{end} on [(r+2)/2] processors equals $k(T_{end}-t_0)/H=kr(T_{end}-t_0)/h_0$, and is independent of the basic stepsize H. Hence, global or local extrapolation with the same internal step is equally expensive.

In order to compare parallel methods with sequential methods, we introduce the redundancy factor

$$R_\kappa := \kappa \frac{T_\kappa(\text{parallel method})}{T_1(\text{sequential method})}.$$

Ideally, this factor should be 1, but in practice this will seldom be achieved.
In the following subsections, we investigate the sequential costs, the order of accuracy, and the stability properties of extrapolation methods generated by the Euler methods, the Midpoint rule, Gragg's method and the trapezoidal rule. For notational convenience, we shall assume that the equation (1.1) is a scalar, autonomous equation. However, all considerations below are straightforwardly extended to systems of ODEs, and therefore, also to nonautonomous equations.

2.2.1. Explicit Richardson-Euler method. Consider the case where the generating method is defined by the forward Euler method with constant stepsizes h:

$$\begin{aligned} &Y_0 = y_0,\ Y_j = Y_{j-1} + hf(Y_{j-1}),\ j = 1, 2, \ldots, m \\ &y(t_0 + H,h) = Y_m,\ m = H/h. \end{aligned} \tag{2.8}$$

Here, it is assumed that H/h is always integer. Evidently, we have p=s=1. We observe that (2.8) and the generated Richardson-Euler method may be considered as explicit RK methods with respect to a step of size H. Before discussing this Richardson-Euler method, we mention a few results from the literature. Firstly, we have the theorem (cf. Iserles and Nørsett [20]):

Theorem 2.2. The numerical results produced by explicit RK methods of order p at the step points necessarily require at least p sequential stages per step point. □

This statement justifies the following definition:

Definition 2.2. An explicit RK method is said to be *optimal on* κ *processors* if its order equals the number of sequential stages per step point on κ processors. □

In Nørsett and Simonsen [26] the question was posed whether it is always possible to find explicit RK methods of any order p which are optimal assuming that sufficiently many processors are available. The following theorem answers this question.

Theorem 2.3. The Richardson-Euler method {(2.5), (2.7), (2.8)} with $H=h_0$ is an explicit Runge-Kutta method of order $p^*=r$ with $r(r-1)/2+1$ stages, which is optimal on $[(r+2)/2]$ processors. □

On one-processor computers, this Richardson-Euler method is much more expensive than the 'conventional' RK methods (cf. Table 1.1). However, on multi-processor computers, the higher-order ones are of interest. From Table 1.1 we see that there exist RK methods of orders $p \le 4$ which are optimal on one processor, but higher order RK methods cannot be optimal on one-processor computers. For example, the 'best' RK method of order p=10 available in the literature (cf. Hairer [12]) requires 17 stages. From Theorem 2.3 it follows that there is a 10th-order Richardson-Euler method with 46 stages requiring 10 sequential stages on six processors. At first sight, a gain factor of 1.7 on 6 processors, that is, a redundancy factor $R_6=3.53$ is rather disappointing. On the other hand, together with the numerical approximation of order 10, a whole set of additional (embedded) approximations of orders 1 until 9 are provided. This enables us to compute error estimates without additional computational effort which can be used for varying-order, varying-step implementations. Nevertheless, it is a challenge to find optimal RK methods of order p^* requiring less than $[(p^*+2)/2]$ processors. This is possible by using s=2 extrapolation, that is, by choosing a *symmetric* generating method, and will be discussed in Subsection 2.2.3.

Table 2.2. Real and imaginary stability boundaries of optimal RK methods.

Order p^*	1	2	3	4	5	6	7	8	9	10
Real	2.0	2.0	2.5	2.7	3.2	3.5	3.9	4.3	4.7	5.0
Imaginary	0.0	0.0	1.7	2.8	0.0	0.0	1.7	3.3	0.0	0.0

Finally, we briefly mention the stability of the Richardson-Euler methods {(2.5), (2.7), (2.8)}. Being optimal RK methods, they possesses a nonempty stability region. In Table 2.2, the real and imaginary stability boundaries of optimal RK methods are listed for orders $p^*=1$ until 10. The methods of orders $p^*=7$ and $p^*=8$ are particularly interesting because they possess not only a substantial real stability interval, but also a nonempty imaginary stability interval.

2.2.2. Implicit Richardson-Euler method. Consider the generating method defined by the backward Euler method with constant stepsizes:

$$\begin{aligned} &Y_0 = y_0,\ Y_j = Y_{j-1} + hf(Y_j),\ j = 1, 2, \ldots, m, \\ &y(t_0 + H,h) = Y_m,\ m = H/h, \end{aligned} \tag{2.9}$$

where H/h is again integer and p=s=1. Method (2.9) may be considered as an SDIRK method. The generated Richardson-Euler method is a DIRK method, but on parallel computers, it has the same computational complexity as an SDIRK method, because each processor needs to compute only one LU-decomposition. In estimating the sequential costs of this Richardson-Euler method, we observe that the bulk of the computational work goes in solving the implicit relations occurring in (2.9), that is, in computing the *singly diagonal-implicit stages.*

Theorem 2.4. The Richardson-Euler method {(2.5), (2.7), (2.9)} with $H=h_0$ is a DIRK method of order $p^*=r$ with $r(r+1)/2$ diagonal-implicit stages and r sequential singly diagonal-implicit stages on $[(r+2)/2]$ processors per step of length H. □

Table 2.3. L(α,β,γ)-stable Richardson-Euler methods of order p*=r requiring r sequential, diagonally implicit stages on [(r+2)/2] processors.

p*	r=2	r=3	r=4	r=5	r=6	r=7	r=8	r=9	r=10
α:	90°	89.85°	89.77°	89.76°	89.78°	89.79°	89.80°	89.82°	89.83°
β:	0	0.829	1.535	2.235	2.955	3.696	4.455	5.223	6.005
γ:	0	$1.7_{10^{-3}}$	$4.9_{10^{-3}}$	$7.5_{10^{-3}}$	$9.6_{10^{-3}}$	$12_{10^{-3}}$	$13_{10^{-3}}$	$15_{10^{-3}}$	$16_{10^{-3}}$

On one-processor computers, these implicit versions are, like the explicit Richardson-Euler methods, much more expensive than their sequential counterparts (cf. Table 1.2). On parallel computers, they may be of interest because their order can be made arbitrarily high. However, it is crucial that their stability region is sufficiently large. For large values of r we expect a large stability region because the stability function of {(2.5), (2.7), (2.9)} converges to the L-acceptable function exp(z) as $r\to\infty$. In order to characterize the size of the stability region for finite values of r, we extend the notion of A(α)-stability and introduce the notions of A(α,β,γ)-stability and L(α,β,γ)-stability:

Definition 2.3. Let S be the stability region in the complex z-plane where the characteristic polynomial of the numerical method has its zeros within the unit circle.
The method is said to be *A(α,β,γ)-stable* if
(i) S contains the regions $\{z: -\alpha<\pi - \arg(z)<\alpha,\ 0<\alpha\le\pi/2\}$, and $\{z: \mathrm{Re}(z)<0,\ |z|>\beta\}$
(ii) $1+\gamma$ is the maximum absolute value of the zeros of the characteristic polynomial when z runs through the region of instability lying in the nonpositive halfplane.
It is said to be *L(α,β,γ)-stable* if
(i) it is A(α,β,γ)-stable
(ii) all zeros of the characteristic polynomial vanish at infinity. □

It is easily verified that implicit Richardson-Euler methods are always L(α,β,γ)-stable. In Table 2.3 the values of (α,β,γ) are listed for r=2 until r=10. In view of the fact that there are no sequential, L-stable SDIRK methods of order higher than four available in the literature (see Table 1.2), we conclude that the 'almost' L-stable Richardson-Euler methods of arbitrarily high order p* requiring p* sequential stages are a substantial improvement on the present situation.

2.2.3. Richardson-Midpoint-Gragg methods. Gragg [11] proved that the Midpoint rule

$$Y_1 = y_0 + hf(y_0),$$

$$(2.10)\quad Y_j = Y_{j-2} + 2hf(Y_{j-1}),\ j=2, 3, \dots, m,\ m = H/h,$$

$$y(t_0 + H,h) = Y_m,$$

and the 'smoothed' Midpoint rule (or: Gragg method)

$$Y_1 = y_0 + hf(y_0),$$

$$(2.11)\quad Y_j = Y_{j-2} + 2hf(Y_{j-1}),\ j=2, 3, \dots, m,\ m = H/h,$$

$$y(t_0 + H,h) = \tfrac{1}{2}[Y_{m-1} + Y_m + hf(Y_m)],$$

are symmetric provided that m=H/h is an *even* integer. Both methods are easily verified to be second-order, explicit RK methods using respectively 2m and 2m+1 stages.

Theorem 2.5. The Richardson-Midpoint method {(2.5), (2.7), (2.10)} and the Richardson-Gragg method {(2.5), (2.7), (2.11)} with $H=2h_0$ are explicit Runge-Kutta methods of order $p^*=2r$ with r^2+1 and r^2+r+1 stages, respectively, and requiring 2r and 2r+2 sequential stages on [(r+2)/2] processors per step of length H. □

Thus, (2.10) generates optimal RK methods of even order $p^*\ge 4$ on [(p*+4)/4] processors which is about half the number of processors needed by the explicit Richardson-Euler methods to be optimal. For example, there exists an optimal tenth-order RK method that needs only three processors, so that, when compared with the tenth-order one-processor method of Hairer, we now have a gain factor 1.7 on 3 processors, that is, a redundancy factor $R_3\approx 1.8$.
The methods {(2.5),(2.7),(2.10)} and {(2.5),(2.7), (2.11)} both possess, like any consistent RK method, nonempty stability regions. However, the recursion for the 'intermediate' approximations Y_j, when applied to the stability test equation $y'=\lambda y$ ($\mathrm{Re}(\lambda)<0$), is unstable in both cases, because its characteristic equation $\zeta^2-h\lambda\zeta-1=0$ has roots outside the unit circle for all values of $h\lambda$ in the left halfplane. Hence, these extrapolation methods are both 'internally' unstable and will develop instabilities unless m is sufficiently small, say $m\le 10$. As a consequence, these methods should only be used in *local* extrapolation mode, rather than in *global* extrapolation mode. In the literature, the Richardson-Gragg method {(2.5),(2.7),(2.11)} is recommended in spite of the two extra sequential stages (cf. [23, p.175] and [15, p.225]).

2.2.4. Richardson-Trapezoidal method. Finally, we consider extrapolation methods generated by the trapezoidal rule:

$$Y_0 = y_0,$$

$$(2.12)\quad Y_j = Y_{j-1} + \tfrac{1}{2}h\,[f(Y_{j-1}) + f(Y_j)],\ j = 1, 2, \dots, m,$$

$$y(t_0 + H,h) = Y_m,\ m = H/h.$$

As in the implicit Richardson-Euler case, the generating method (2.12) is an SDIRK method and the Richardson-Trapezoidal method {(2.5),(2.7),(2.12)} is a DIRK method with singly diagonal-implicit stages when run on a parallel computer. Since (2.12) is symmetric, we can generate relatively high order extrapolation methods by setting $H=h_0$. However, it can be shown that $A(\alpha,\beta,\gamma)$-stability is not possible. Therefore, we set $H=2h_0$ and obtain:

Theorem 2.6. The Richardson-Trapezoidal method {(2.5), (2.7), (2.12)} with $H=2h_0$ is a DIRK method of order $p^*=2r$ with r^2+1 diagonal-implicit stages and 2r sequential singly diagonal-implicit stages on [(r+2)/2] processors per step of length H. []

Table 2.4. A(α)-stable Richardson-Trapezoidal methods of order $p^*=2r$ requiring 2r sequential, diagonally implicit stages on [(r+2)/2] processors.

$p^*=2r$	4	6	8	10	12	14	16
α:	79.2°	70.4°	64.3°	59.8°	56.3°	53.2°	50.6°

When compared with the implicit Richardson-Euler method, we see from this theorem that we have again a method of which the number of sequential stages per step equals its order, but the number of processors and the internal stepsize are about halved. On the other hand, instead of $L(\alpha,\beta,\gamma)$-stability we now have only $A(\alpha,\beta,\gamma)$-stability with $\beta=\infty$. In Table 2.4, the values of α are listed for r=2 until r=8. It should be remarked that the original Romberg sequence $\{m_i\}=\{2^i\}$ also leads to $A(\alpha,\beta,\gamma)$-stability with $\beta=\infty$, however, with considerably larger values of α (see Stetter [27]).

3. METHODS BASED ON ITERATION

Ideally, we want ODE solvers that are highly accurate and highly stable. For example, such methods are provided by the implicit RK methods based on Gaussian quadrature formulas (such as Gauss-Legendre, Lobatto and Radau methods), which are known to be A-stable for any order of accuracy. However, the high degree of implicitness of these methods implies that solving the implicit relations is rather costly. In general, a k-stage implicit RK method applied to a d-dimensional system of ODEs requires in each step the solution of a system of dimension kd, so that the computational complexity is of order $(kd)^3$. This compares unfavourably with implicit linear multistep methods (LM methods) which require in each step the solution of a system of dimension d. Although implicit LM methods have less favourable stability properties (particularly the higher-order ones) than implicit RK methods, they are more popular, just because they are cheap. However, on parallel computers this situation may change. In this section, we shall show that the implicitness of implicit RK methods can be tackled rather efficiently on multi-processor computers.

In Section 3.1, we give a brief description of the parallel iteration methods to be used and in Section 3.2 they are applied to implicit RK methods.

3.1. Parallel iteration

Suppose that we want to solve the system of equations

$$(3.1)\quad \mathbf{Y} = \mathbf{F}(\mathbf{Y}),\ \mathbf{F}: \mathbb{R}^{d\kappa} \to \mathbb{R}^{d\kappa},$$

where $\mathbf{Y}$ is the unknown vector, $\mathbf{F}$ is a nonlinear function, and where $\mathbf{F}$, κ and d are such that the computational complexity of the first set of d components of $\mathbf{F}$ is about equal to that of the second, third, ... sets of d components of the function $\mathbf{F}$.

Consider the iteration method

$$(3.2)\quad \mathbf{Y}_j - \mathbf{G}(\mathbf{Y}_j) = \mathbf{F}(\mathbf{Y}_{j-1}) - \mathbf{G}(\mathbf{Y}_{j-1}),\ j = 1, 2, \ldots,$$

where we assume that an initial approximation $\mathbf{Y}_0$ is given and where $\mathbf{G}$ is a 'free' function with *block diagonal* Jacobian matrix the blocks of which are of dimension d. Evidently, if $\mathbf{Y}_j$ is solved from (3.2) by Newton iteration, then each set of d components of $\mathbf{Y}_j$ can be computed independently of the other sets of d components, that is, they can be computed in parallel provided that κ processors are available. If this Jacobian of $\mathbf{G}$ is a *nonzero* matrix, then we shall call (3.2) diagonally-implicit iteration, or briefly *diagonal iteration*. If the Jacobian vanishes, then we speak of *explicit iteration*. For notational convenience, we assume in the following that d=1, but all considerations are straightforwardly extended to the case where d>1.

There are several options for choosing the function $\mathbf{G}$:

(i) If the implicit relation (3.1) is well-conditioned, we may simply choose $\mathbf{G}=\mathbf{0}$ to obtain explicit iteration.

(ii) Another possibility consists in choosing $\mathbf{G}$ such that after a fixed number of iterations we have optimal stability properties for the resulting ODE solver. This leads to diagonal iteration.

(iii) A third possibility is improving the rate of convergence. In first approximation, we have the error equation

$$\mathbf{Y}_j - \mathbf{Y} = Z\,[\mathbf{Y}_{j-1} - \mathbf{Y}],\ Z := [I - J_G]^{-1}[J_F - J_G],$$

where J_F and J_G denote the Jacobian matrices of $\mathbf{F}$ and $\mathbf{G}$ at $\mathbf{Y}_0$. We now choose $\mathbf{G}$ such that the norm of the matrix Z is in some sense small in magnitude. This leads again to diagonal iteration.

3.2. Application to implicit Runge-Kutta methods

As before, we shall assume in the various formulas that (1.1) is scalar and autonomous. It is convenient to write the general RK method in the form

$$(3.3)\quad \begin{aligned} y_{n+1} &= y_n + hb_0 f(y_n) + h\mathbf{b}^T f(\mathbf{Y}), \\ \mathbf{Y} &= y_n\mathbf{e} + ha f(y_n) + hA f(\mathbf{Y}), \end{aligned}$$

where $\mathbf{e}$ is the column vector of dimension k with unit entries, $\mathbf{a}$ and $\mathbf{b}$ are k-dimensional vectors and A is a k-by-k matrix. Furthermore, we have used the convention that for any given vector $v=(v_j)$, $f(v)$ denotes the vector with entries

$f(v_j)$. The RK method (3.3) will be called the *generating corrector* and will always be assumed to be of order p.
Choosing $\mathbf{G}(\mathbf{Y})=hDf(\mathbf{Y})$, where D is a diagonal matrix (and therefore, the Jacobian of $\mathbf{G}$ is diagonal), and using a one-step method for computing the initial approximation to $\mathbf{Y}$, the iteration method (3.2) assumes the form

$$\begin{aligned} &\mathbf{Y}_0 - hBf(\mathbf{Y}_0) = y_n\mathbf{e} + hCf(y_n\mathbf{e}), \\ &\mathbf{Y}_j - hDf(\mathbf{Y}_j) = y_n\mathbf{e} + haf(y_n) + h[A-D]\, f(\mathbf{Y}_{j-1}), \end{aligned} \tag{3.4}$$

where $j=1, \ldots, m$, B is an arbitrary *diagonal* matrix and C is an arbitrary matrix. We assume that $\kappa=k$ processors are available, so that (3.4) is a parallel iteration method.
Two versions of computing y_{n+1} will be considered. The most natural one sets

$$y_{n+1} = y_n + hb_0 f(y_n) + h\mathbf{b}^T f(\mathbf{Y}_m). \tag{3.5}$$

A related method computes y_{n+1} from the relation

$$y_{n+1} = \mathbf{e}_k^T \mathbf{Y}_m, \tag{3.5*}$$

where $\mathbf{e}_k$ is the kth unit vector. If the generating corrector happens to be *stiffly accurate*, that is, if b_0 equals the last element of $\mathbf{a}$ and if $\mathbf{b}^T$ equals the last row of A (cf. [1]), then the method {(3.4),(3.5*)} has better stability properties for stiff equations than {(3.4),(3.5)}. We shall call {(3.4),(3.5)} the *nonstiffly accurate* version and, if the generating corrector is itself stiffly accurate, then we call {(3.4),(3.5*)} the *stiffly accurate* version. Both versions lead to RK-type methods with a large number of stages depending on k, m and the matices B, C and D.
In the case of nonvanishing matrices D and a fixed number of iterations, the resulting RK method is a DIRK method since only diagonally-implicit relations are to be solved. On parallel computers, the method only requires the computation of singly diagonal-implicit stages (cf. the Richardson-Euler and Richardson-Trapezoidal methods). In [18], these special (S)DIRK methods were called *PDIRK methods* (Parallel, Diagonal-implicitly Iterated RK). In estimating the sequential costs of PDIRK methods, we have again that the bulk of the computational work goes in computing the singly diagonal-implicit stages. Evidently, the number of sequential singly diagonal-implicit stages on k processors equals m if B=O and m+1 otherwise.
Given the generating corrector (3.3), the methods {(3.4),(3.5)} and {(3.4),(3.5*)} still contain the 'free' iteration matrices B, C and D. In the following subsections, we discuss a number of special choices leading to highly stable RK methods with a relatively small number of sequential singly diagonal-implicit stages on k-processor computers. First methods with vanishing D matrix are considered. The resulting RK method is explicit and will be called *PIRK method* (Parallel, Iterated RK).

3.2.1. PIRK methods with fixed number of iterates. We consider PIRK methods with

$$\mathbf{a} = \mathbf{0},\ b_0 = 0,\ D = B = C = O,\ m \text{ fixed}. \tag{3.6}$$

The construction of ODE solvers of this type has been suggested in many papers (see, e.g., [26], [24], [21], [20], [16]). In [16] the following result was presented:

Theorem 3.1. The PIRK method {(3.4),(3.5),(3.6)} has order $p^*:=\min\{p,m+1\}$ with km+1 stages and is optimal on k processors if $m \le p-1$. There exist optimal PIRK methods of order p on [(p+1)/2] processors and these methods can be generated by the Gauss-Legendre methods. □

A comparison with the Richardson-Euler methods of Theorem 2.3 reveals that the Gauss-Legendre-based PIRK methods require one processor less for p even and the same number of processors for p odd. However, the PIRK methods are expected to be much more efficient because of the high accuracy of the generating correctors (see Section 4). On the other hand, the implementation of the Richardson-Euler method is extremely simple, while the implementation of the PIRK methods is somewhat more complicated.

Table 3.1. PDIRK methods of order p*=p on [(p+1)/2] processors.

Corrector	Iteration matrices	Order	Seq. stages	Stability	Stiff accuracy
1. Radau IIA	B=D=diag(Ae-Ce), A^2e=BAe	p=3	p-1	A-stable	Yes
2. Radau IIA	B=D=diag(Ae), C=O	p=3	p-1	Strongly A-stable	No
3. Radau IIA	B=D=diag(Ae), C=O	p=3	p	L(89.75°)-stable	Yes
4. Gauss-Legendre	B=D=diag(Ae), C=O	p=4	p-1	Strongly A-stable	No
5. Radau IIA	B=D=diag(Ae-Ce), A^2e=BAe	p=5	p-1	A(89.997°)-stable	Yes
6. Radau IIA	B=D=diag(Ae), C=O	p=5	p-1	Strongly A-stable	No
7. Radau IIA	B=D=diag(Ae), C=O	p=5	p	L(89.12°)-stable	Yes
8. Gauss-Legendre	B=D=diag(Ae), C=O	p=6	p-1	Strongly A(89.97°)-stable	No
9. Radau IIA	B=D=diag(Ae-Ce), A^2e=BAe	p=7	p-1	A(89.95°)-stable	Yes
10. Radau IIA	B=D=diag(Ae), C=O	p=7	p-1	Strongly A(83.3°)-stable	No
11. Radau IIA	B=D=diag(Ae), C=O	p=7	p	L(89.02°)-stable	Yes

Table 3.2. PDIRK methods of order $p^* \le p$ on $[(p+1)/2]$ processors with B=bI and D=dI.

Corrector	Iteration matrices	Order	Seq. stages	Stability	Stiff accuracy
1. Gauss-Legendre	B=O, D=dI, C=O	$p^* \le 4$	p^*-1	A-stable	No
2. Gauss-Legendre	B=D=dI=diag(Ae-Ce), C≠O	$p^* \le 4$	p^*-1	A-stable	No
3. Radau IIA	B=O, D=dI, C=O	$p^* \le 6$	p^*	L-stable	Yes
4. Radau IIA	B=O, D=dI, C=O	$p^* = 8$	p^*	L-stable	Yes
5. Radau IIA	B=D=dI, C=O	$p^* \le 8$	p^*+1	L-stable	Yes
6. Radau IIA	B=D=dI, C=O	$p^* = 10$	p^*+1	L-stable	Yes

Matters are different with the Richardson-Gragg methods. According to Theorem 2.5, they need about half the processors required by the PIRK methods at the cost of two additional sequential stages. The PIRK methods are again expected to be more efficient, but are also more difficult to implement.
Finally, we remark that PIRK methods (like the Richardson-type methods) contain a whole set of embedded formulas of lower order which can be used for order and stepsize control (cf. [16] for a discussion of strategies).

3.2.2. PDIRK methods with fixed numbers of iterates. In [18] the following family of PDIRK methods has been investigated:

(3.7) $\mathbf{a} = \mathbf{0}$, $b_0 = 0$, $D \neq O$, m fixed.

For these PDIRK methods we have:

Theorem 3.2. The order p^* of the nonstiffly accurate PDIRK method {(3.4),(3.5),(3.7)} is given by

$$
\begin{aligned}
& p^* = \min\{p, m+1\} && \text{for all B, C and D.} \\
(3.8)\quad & p^* = \min\{p, m+2\} && \text{for } Ae=(C+B)e. \\
& p^* = \min\{p, m+3\} && \text{for } Ae=(C+B)e,\ A^2e=BAe. \ \square
\end{aligned}
$$

Theorem 3.2*. Let the generating corrector be stiffly accurate, then the PDIRK method {(3.4),(3.5*),(3.7)} is also stiffly accurate and its order is given by (3.8) with m replaced by m-1. □

Several types of PDIRK methods can now be distinguished depending on the particular choice of the iteration matrices B, D and C. If the generating corrector is the Gauss-Legendre or Radau IIA for even and odd values of p, respectively, then the number of processors for exploiting the parallelism across the method equals $[(p+1)/2]$. Choosing m such that $p^*=p$ we derived a number of highly stable PDIRK methods which are collected in Table 3.1. In the stiffly accurate cases, the PDIRK method is defined by {(3.4),(3.5*),(3.7)}, otherwise by {(3.4),(3.5),(3.7)}. As before, all PDIRK methods contain embedded formulas of lower order.
In actual computation, there is some advantage by choosing the elements of the diagonal matrices B and D constant, so that all k processors can use the same LU-decomposition when solving the implicit relations during the iteration process. In such cases, these decompositions, as well as the evaluation of the Jacobian matrix $\partial f/\partial y$, may be performed by an additional processor, providing an updated Jacobian and decomposition for *all* processors as soon as it is available. In Table 3.2, methods of this type are listed. The relevant values of d occurring in this table can be found in [18]. Again, all PDIRK methods contain embedded formulas of lower order.

3.2.3. PDIRK methods with variable numbers of iterates. For most stiff ODEs, RK methods exhibit the phenomenon of order reduction which means that for realistic stepsizes the *observed* (or: effective) order is, in most cases, much smaller than the *classical* (or: algebraic, asymptotic) order. In fact the observed order is dictated by the so-called *stage order* (cf., e.g., [5]). Since the stage order is usually substantially lower than the classical order, this order reduction phenomenon is a serious problem for all RK methods (only if the ODE is of a special singular perturbation form and if the RK method is stiffly accurate, then this order reduction is not shown in numerical experiments (cf. [14])). In particular, DIRK methods, and hence the PDIRK methods discussed in the preceding subsection, have a low stage order of one or two. This puts a question mark at the relevance of constructing RK methods with the highest possible *algebraic* order and motivated us to look for generating correctors with the highest possible *stage* order, which are then iterated until convergence. As a consequence, we can rely on the stability and accuracy behaviour of the (solved) corrector and thus, in particular, on its high stage order.
In Table 3.3, we have collected a few families of generating implicit RK methods with high stage order together with their stability properties (here, k is the order of the matrix A when presented in the form (3.3)). In [17] we have analysed the diagonal iteration method (3.4) of these correctors with

(3.9) $B = C = O$, $D \neq O$, m variable.

We investigated convergence of the iterates $\mathbf{Y}_m$ and of the stability functions associated with (3.5) and (3.5*) on the basis of the linear test equation $y'=\lambda y$.

Theorem 3.3. Let y(t) denote the exact local solution with $y(t_n)=y_n$, let r and $R_{corr}(z)$ be the stage order and stability function of the generating corrector, and define

$$\mathbf{c} := \mathbf{a} + Ae,\quad Z := zD\,[I - zD]^{-1}[D^{-1}A - I],\quad z := \lambda h.$$

(a) The method {(3.4),(3.8)} satisfies the order relation

$$\mathbf{Y}_m = y(t_n+ch) + O(h^{r+1}) + hZ^m[af(y_n) + Af(\mathbf{Y})].$$

Table 3.3. Summary of characteristics of implicit RK methods.

Method	Stages	Order p	Stage order r	Stability	Stiff accuracy	Reference
Gauss-Legendre	k	2k	k	A-stable for all k	no	[3]
Lobatto IIIA	k+1	2k	k+1	A-stable for all k	yes	[5]
Radau IIA	k	2k-1	k	L-stable for all k	yes	[3]
Newton-Cotes	k+1	2[(k+2)/2]	k+1	A-stable for k≤8	yes	[28]
Lagrange	k+1	k+1	k+1	Strongly A-stable	yes	[17]

Table 3.4. Stiffly accurate PDIRK methods {(3.4),(3.5*),(3.8)} of order $p^* = \min\{p,m\}$.

Corrector	Order p	Stage order r	Processors	$\rho(D^{-1}A-I)$	σ_m-range	$m_{A\text{-stable}}$
1. Radau IIA	3	2	2	0	[0.27, 0.35]	1
2. Lagrange	3	3	2	0	[0.21, 0.33]	2
3. Radau IIA	5	3	3	0.0047	[0.52, 1.00]	5
4. Lagrange	4	4	3	0.01	[0.49, 0.69]	3
5. Radau IIA	7	4	4	0.024	[0.74, 1.31]	7
6. Lagrange	5	5	4	0.045	[0.59, 0.93]	6

(b) The methods {(3.4),(3.5),(3.8)} and {(3.4),(3.5*),(3.8)} respectively possess the stability functions

$$R_m(z) := R_{corr}(z) - z^2\mathbf{b}^T Z^m [I - zA]^{-1}\mathbf{c},$$

$$R^*_m(z) := R_{corr}(z) - z\mathbf{e}_k^T Z^m [I - zA]^{-1}\mathbf{c}. \quad \square$$

In the first place, this theorem shows that the stability function $R_m(z)$ of the method {(3.4),(3.5),(3.8)} grows beyond all bounds as z tends to infinity. Hence, we shall from now on concentrate on the method {(3.4),(3.5*),(3.8)}. Secondly, Theorem 3.3 illustrates the central role of the matrix Z(z) both for the accuracy and the stability of the method. In order to achieve that Z^m converges rapidly to the zero matrix for a given problem, we should choose the matrix D such that the spectral radius $\rho(Z(z))$ of Z(z) is as small as possible in the region of relevant z-values. Let $\mathbb{R}$ denote the region of z-values where we want fast convergence, then this implies the minimization over all possible diagonal matrices D of the function $\mu=\mu(D)$ where μ denotes the maximum of $\rho(Z(z))$ in the region $\mathbb{R}$. In [17] this minimax problem was considered for the case where $\mathbb{R}$ is the region of 'stiff' values of z, that is, we required fast convergence for $|z|>>1$ (notice that $Z^m(z)$ behaves as $z^m(A-D)^m$ as $z\to 0$, so that we have automatically fast convergence for small values of z). Since $\rho(Z(z))\to \rho(D^{-1}A-I)$ as $z\to\infty$, we were led to minimize the spectral radius of $D^{-1}A-I$. Having found a suitable matrix D with minimal $\rho(D^{-1}A-I)$-value, it is of interest to know the rate of convergence of R^*_m to R_{corr} for that particular matrix D. The rate of convergence can be measured by the quantity

$$\sigma_m := \max_{Re z<0} \sqrt[m]{|R^*_m(z) - R_{corr}(z)|}\,.$$

Finally, we checked for what value of m the method becomes (and remains) A-stable. This value is denoted by $m_{A\text{-stable}}$.
Table 3.4 presents for a number of generating correctors the values of $\rho(D^{-1}A-I)$, the range of σ_m-values for $1\le m\le 10$, and the critical value $m_{A\text{-stable}}$. We only listed cases with finite $m_{A\text{-stable}}$, which restricted the PDIRK methods to methods generated by the Radau IIA and Lagrange correctors.

4. NUMERICAL EXPERIMENTS

We shall present numerical results obtained by the parallel methods described in the preceding sections and by existing sequential and parallel methods. From each family of methods we tested a representative example. Throughout, we use the following notations:

- L length of the integration interval $[t_0,T_{end}]$
- N total number of sequential stages
- p* order of the method
- D number of correct decimal digits at the endpoint, i.e., we write the maximum norm of the error at $t=T_{end}$ in the form 10^{-D}.

All methods were applied with constant stepsizes.

4.1. Explicit Runge-Kutta methods

We performed a few tests by integrating the equation of motion for a rigid body without external forces (cf. Problem B5 from [19]):

$$\begin{aligned} y_1' &= y_2y_3, & y_1(0) &= 0, \\ y_2' &= -y_1y_3, & y_2(0) &= 1, \quad 0 \le t \le T_{end} = 60. \\ y_3' &= -.51y_1y_2, & y_3(0) &= 1, \end{aligned} \tag{4.1}$$

Curtis and Hairer used this test problem for testing and comparing their 10th-order RK methods. In Table 4.1 the results of the experiments performed by Curtis and Hairer are reproduced (cf. [12]), together with results obtained by a few parallel methods. This table shows the superiority of the PIRK method.

Table 4.1. Results for Problem (4.1).

Method	p*	D	N
Runge-Kutta	4	9.6	48000
Adams-Moulton-Bashforth	4	8.1	12000
Runge-Kutta-Curtis	10	9.9	4320
Runge-Kutta-Hairer	10	10.1	4080
Richardson-Gragg with $H=2h_0=1/3$	10	9.5	2160
Richardson-Midpoint, $H=2h_0=1/3$	10	9.6	1800
Gauss-Legendre-based PIRK, m=9	10	10.0	1560

4.2. Richardson-Gragg and Richardson-Midpoint methods
Table 4.1 shows that the Richardson-Gragg method and the Richardson-Midpoint method produce comparable accuracies. A second example confirms this behaviour: Table 4.2 lists results for the Fehlberg problem [8]

$$\begin{aligned} y_1' &= 2ty_1\log(\max\{y_2, 10^{-3}\}), \; y_1(0) = 1, \\ y_2' &= -2ty_2\log(\max\{y_1, 10^{-3}\}), \; y_2(0) = e, \end{aligned} \qquad 0\le t\le 5 \tag{4.2}$$

for the methods:

RM: Richardson-Midpoint: $p^*=2r$, $H=2h_0=2rL/N$.
RG: Richardson-Gragg: $p^*=2r$, $H=2h_0=2(r+1)L/N$.

Table 4.2. Results for Problem (4.2).

Method	H	p*=2	p*=4	p*=6	p*=8	p*=10	p*=12
RM	0.1	-0.2	0.6	2.3	4.5	6.3	7.6
RG	0.1	0.6	0.9	2.3	4.4	6.7	8.5
RM	0.05	0.9	2.1	4.3	7.1	9.2	11.4
RG	0.05	1.6	2.5	4.5	7.1	9.7	11.4

4.3. Diagonally-implicit methods
Finally, we compare a number of diagonally-implicit methods by integrating a few stiff test problems. The first test problem is taken from Kaps [22]:

$$\begin{aligned} y_1' &= -(2+\varepsilon^{-1})y_1 + \varepsilon^{-1}(y_2)^2, \; y_1(0) = 1, \\ y_2' &= y_1 - y_2(1 + y_2), \; y_2(0) = 1, \end{aligned} \qquad 0 \le t \le 1. \tag{4.3}$$

Table 4.3 clearly shows the superiority of the Richardson-Trapezoidal methods. Also, notice that the high (classical) order of the PDIRK method 9 from Table 3.1 becomes only apparent for the smallest stepsizes.
Our second example is taken from the test set of Enright et al. [7] and is given by the following system of ODEs describing a chemical reaction:

$$\frac{dy}{dt} = -\begin{pmatrix} .013+1000y_3 & 0 & 0 \\ 0 & 2500y_3 & 0 \\ .013 & 0 & 1000y_1+2500y_2 \end{pmatrix} y, \tag{4.4a}$$

with $y(0)=(1,1,0)^T$. Following [17], we avoided the initial phase by choosing the starting point at $t_0=1$ and we used the corresponding initial values

$$y(1) \approx \begin{pmatrix} 0.990731920827 \\ 1.009264413846 \\ -.366532612659\;10^{-5} \end{pmatrix}. \tag{4.4b}$$

This enables us to integrate with constant stepsizes. Results at $T_{end}=51$ are given in Table 4.3. Notice that some methods that were the more accurate methods in the case of Problem (4.3), are now less accurate. For the case of extrapolation methods for sequential machines, a detailed comparitive study was carried out by Deuflhard [6]. A performance analysis of PDIRK methods will be future research of the present authors.

Table 4.3. Results for the Problem (4.3) with $\varepsilon=10^{-8}$ and Problem (4.4).

Method	p*	Problem (4.3) N=6	N=12	N=24	N=48	Problem (4.4) N=6	N=12	N=24	N=48
Crouzeix-Alexander	4	2.0	3.8	3.6	4.1	4.4	5.5	6.7	7.8
Iserles-Nørsett	4	2.7	3.5	4.3	4.9	5.1	6.2	7.4	8.6
Richardson-Impl. Euler: $H=h_0=6L/N$	6	5.2	6.6	8.1	9.7	7.1	8.6	10.3	
Richardson-Trapezoidal: $H=2h_0=6L/N$	6	6.7	8.4	10.1	11.9	9.7	7.3	7.4	7.8
Richardson-Trapezoidal: $H=L=Nh_0/3$	6	6.7	8.5	10.3		8.2	10.0	11.7	
PDIRK method 9 from Table 3.1	7	5.2	6.8	8.6	10.5	7.4	9.4	11.5	12.0
PDIRK method 5 from Table 3.2	5	4.2	5.6	7.1	8.6	5.3	6.8	8.3	9.8
PDIRK method 6 from Table 3.4	5	6.9	7.7	8.9	10.4	6.9	8.2	9.7	11.2

REFERENCES

[1] **Alexander, R. (1977)**: Diagonally implicit Runge-Kutta methods for stiff ODEs, SIAM J. Numer. Anal. **14**, 1006-1021.

[2] **Bellen, A. & M. Zennaro (1989)**: Parallel algorithms for initial-value problems for difference and differential equations, J. Comp. Appl. Math. **25**, 341-350.

[3] **Butcher, J.C. (1987)**: The numerical analysis of ordinary differential equations, Runge-Kutta and general linear methods, Wiley, New York.

[4] **Crouzeix, M. (1975)**: Sur l'approximation des équations différentielles opérationnelles lineáires par des méthodes de Runge-Kutta, Ph. D. Thesis, Université de Paris.

[5] **Dekker, K. & Verwer, J.G. (1984)**: Stability of Runge-Kutta methods for stiff nonlinear differential equations, CWI Monograph **2**, North-Holland, Amsterdam.

[6] **Deuflhard, P. (1985)**: Recent progress in extrapolation methods for ordinary differential equations, SIAM Review **27**, 505-535.

[7] **Enright, W.H., Hull, T.E. & Lindberg, B. (1975)**: Comparing numerical methods for stiff systems of ODEs, BIT **15**, 10-48.

[8] **Fehlberg, E. (1968)**: Classical fifth-, sixth-, seventh-, and eighth-order Runge-Kutta formulas with stepsize control, NASA Technical Report **287**, extract published in Computing **4** (1969), 93-106.

[9] **Gear, C.W. (1987)**: Parallel methods for ordinary differential equations, Report No. UIUCDCS-R-87-1369, University of Illinois, Urbana Champaign, Urbana.

[10] **Gladwell, I. (1989)**: IBM Workshop on Numerical Software in Garmisch (FRD), August 1989.

[11] **Gragg, W.B. (1964)**: Repeated extrapolation to the limit in the numerical solution of ordinary differential equations, Thesis, Univ. of California.

[12] **Hairer, E. (1978)**: A Runge-Kutta method of order 10, J. Inst. Math. Applics. **21**, 47-59.

[13] **Hairer, E. (1989)**: NUMDIFF 5 meeting in Halle (GDR), May 1989.

[14] **Hairer, E., Lubich, Ch. & Roche, M. (1988)**: Error of Runge-Kutta methods for stiff problems studied via differential algebraic equations, BIT **28**, 678-700.

[15] **Hairer, E., Nørsett, S.P. & Wanner, G. (1987)**: Solving ordinary differential equations I. Nonstiff problems, Springer-Verlag, Berlin-Heidelberg-New York-London-Paris-Tokyo.

[16] **Houwen, P.J. van der & Sommeijer, B.P. (1990)**: Parallel iteration of high-order Runge-Kutta methods with stepsize control, J. Comp. Appl. Math. **29**, 111-127.

[17] **Houwen, P.J. van der & Sommeijer, B.P. (1990)**: Iterated Runge-Kutta methods on parallel computers, Report **NM-R9001**, Centre for Mathematics and Computer Science, Amsterdam (submitted for publication).

[18] **Houwen, P.J. van der, Sommeijer, B.P. & Couzy, W. (1989)**: Embedded diagonally implicit Runge-Kutta algorithms on parallel computers, Report **NM-R8912**, Centre for Mathematics and Computer Science, Amsterdam (submitted for publication).

[19] **Hull, T.E., Enright, W.H. , Fellen, B.M. & Sedgwick, A.E. (1972)**: Comparing numerical methods for ordinary differential equations, SIAM J. Numer. Anal. **9**, 603-637.

[20] **Iserles, A. & Nørsett, S.P. (1988)**: On the theory of parallel Runge-Kutta methods, Report DAMTP **1988/NA12**, University of Cambridge.

[21] **Jackson, K. & Nørsett, S.P. (1988)**: Parallel Runge-Kutta methods, to appear .

[22] **Kaps, P. (1981)**: Rosenbrock-type methods, in: Numerical methods for stiff initial value problems (eds.: G. Dahlquist & R. Jeltsch), Bericht Nr. **9**, Inst. für Geometrie und Praktische Mathematik der RWTH Aachen.

[23] **Lambert, J.D. (1973)**: Computational methods in ordinary differential equations, John Wiley, New York.

[24] **Lie, I. (1987)**: Some aspects of parallel Runge-Kutta methods, Report No. **3/87**, University of Trondheim, Division Numerical Mathematics.

[25] **Nørsett, S.P. (1974)**: Semi-explicit Runge-Kutta methods, Report Mathematics and Computation No. **6/74**, Dept. of Mathematics, University of Trondheim.

[26] **Nørsett, S.P. & Simonsen, H.H. (1989)**: Aspects of parallel Runge-Kutta methods, in: Numerical methods for ordinary differential equations, A. Bellen, C.W. Gear & E. Russo (eds.), Proceedings L'Aquila 1987, Lecture Notes in Mathematics **1386**, Springer-Verlag, Berlin.

[27] **Stetter, H.J. (1973)**: Analysis of discretization methods for ordinary differential equations, Springer-Verlag, Berlin-Heidelberg-New York.

[28] **Watts, H.A. & Shampine, L.F. (1972)**: A-stable block implicit one-step methods, BIT **12**, 252-266.