Applied Numerical Mathematics 11 (1993) 69–81
North-Holland

APNUM 357

Parallel step-by-step methods

P.J. van der Houwen

Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, Netherlands

Abstract

Van der Houwen, P.J., Parallel step-by-step methods, Applied Numerical Mathematics 11 (1993) 69–81.

In this contribution we survey recent research at CWI on step-by-step methods for solving initial value problems (IVPs) on parallel computers. More general surveys of parallel IVP solvers are given in [5,17,18]. The present paper is organized according to the following sections and subsections: 1. nonstiff problems, 1.1. explicit block methods, 1.2. predictor–corrector iteration, 1.2.1. Runge–Kutta correctors, 1.3. Richardson extrapolation; 2. stiff problems, 2.1. diagonally implicit block methods, 2.2. diagonally implicit iteration, 2.2.1. Runge–Kutta correctors, 2.3. Richardson extrapolation.

Keywords. Initial value problems; step-by-step methods; parallelism; Runge–Kutta methods; block methods.

1. Nonstiff problems

The methods surveyed in this section are explicit step-by step methods for nonstiff first-order IVPs:

$$\frac{dy}{dt} = f(y), \qquad y(t_0) = y_0. \tag{1.1}$$

We shall consider three techniques for constructing methods that are suitable for use on parallel computers, viz. (i) block-by-block calculations, (ii) predictor–corrector iteration (PC iteration), and (iii) Richardson extrapolation. In the terminology introduced by Gear, these techniques result into parallel methods in which the parallelism is called *parallelism across the method*. Methods possessing this form of parallelism can already profit from parallel architectures in the case of *scalar* IVPs. Alternative parallel techniques based on *parallelism across time* and *across space* (including waveform relaxation) are discussed in [1–3,11,12] where further references can be found.

In order to demonstrate more clearly that the methods of this section do possess the property of method parallelism, we shall describe the various methods for scalar IVPs. We remark that the methods discussed below can also be applied (with appropriate changes) to *second-order* IVPs without first derivatives. For example, an analysis of parallel Runge–Kutta–Nyström PC methods can be found in Sommeijer [27].

Correspondence to: P.J. van der Houwen, Centre for Mathematics and Computer Science, P.O. Box, 4079, 1009 AB Amsterdam, Netherlands.

1.1. Explicit block methods

Let (1.1) be a scalar IVP and let us define the k-dimensional vector

$$Y_{n+1} := (y_{n,c_1}, y_{n,c_2}, \ldots, y_{n,c_k})^T, \quad c_k = 1, \tag{1.2}$$

where $y_{n,c}$ denotes a numerical approximation to the exact solution value $y(t_{n+c})$. A rather wide class of explicit methods for solving the IVP (1.1) is given by the block method

$$Y_{n+1} = AY_n + hBf(Y_n), \quad y_{n+1} := e_k^T Y_{n+1}, \qquad n = 0, 1, 2, \ldots, \tag{1.3}$$

where A and B are $k \times$ k matrices, e_k is the kth unit vector, and where for any given vector $v = (v_j)$, $f(v)$ denotes the vector with entries $f(v_j)$. Given the initial vector Y_0, (1.3) completely determines a sequence of numerical approximations to the exact solution values at the step points t_n, $n \geqslant 1$. Thus, in general, methods of the form (1.3) require k starting values.

Since the k components of the vectors Y_{n+1} can be computed in parallel (provided that k processors are available), the computational time (wall-clock time) needed for one step of (1.3) is roughly equal to the time required to evaluate one right-hand side function on a sequential computer. In the following, we mean by "sequential costs per step" the computational time required per step if k processors are available, and an explicit method is said to have σ^* *sequential* stages if the computation time required for evaluating all right-hand sides in one step is about σ^* times the computation time required for evaluating one right-hand side evaluation. We always assume that we have k processors at our disposal.

Many (explicit) methods from the literature can be cast into the form (1.3). Table 1 lists examples of explicit block methods together with the required number of starting values k, their order p at the step points, the block vector c, the number of processors P needed to reduce the sequential costs to just a single f-evaluation per step (i.e., $\sigma^* = 1$), and the real and imaginary stability boundaries β_{real} and β_{imag} (cf. [28]). If S denotes the (theoretical) speed-up factor if the computation times on one and P processors are compared, then all methods of Table 1 have $P = S$. Notice that the number of processors needed for implementing (1.3) is often less than k.

Table 1
Explicit stable block methods of the form (1.3)

References	k	p	c^T	$P = S$	β_{real}	β_{imag}
Two-step Adams–Bashforth	2	2	(0, 1)	1	1.00	0.00
Miranker–Liniger [22]	2	2	(2, 1)	2	0.59	0.60
Van der Houwen–Sommeijer [33, method (4.1)]	2	3	(5/3, 1)	2	0.64	0.65
Three-step Adams–Bashforth	3	3	(−1, 0, 1)	1	0.55	0.72
Van der Houwen–Sommeijer [33, method (4.7)]	3	4	(0, 17/10, 1)	2	0.53	0.05
Four-step Adams–Bashforth	4	4	(−2, −1, 0, 1)	1	0.30	0.43
Miranker–Liniger [22]	4	4	(−1, 0, 2, 1)	2	0.50	0.04

1.2. Predictor–corrector iteration

Consider implicit block methods of the form

$$Y_{n+1} = AY_n + hBf(Y_n) + hCf(Y_{n+1}), \quad n = 0, 1, 2, \ldots, \tag{1.4}$$

where A, B, and C are $k \times k$ matrices, and Y_n is defined as before. A large number of methods, including Runge–Kutta (RK) methods and linear multistep methods, can be rewritten in this form (in fact, (1.4) fits into the class of the *general linear methods* introduced by Butcher, see [6]). If (1.4) is equivalent with an RK method, then we shall call (1.4) a *block method of RK type*.

The most simple method for solving equation (1.4) is PC iteration (or fixed-point iteration):

$$\begin{aligned} &Y^{(j)} = Y^{(j-1)} - \left[Y^{(j-1)} - AY_n - hBf(Y_n) - hCf(Y^{(j-1)})\right], \quad j = 1, 2, \ldots, m, \\ &Y_{n+1} := Y^{(m)}, \qquad y_{n+1} := e_k^{\mathrm{T}} Y_{n+1}, \end{aligned} \tag{1.5}$$

where $Y^{(0)}$ is an initial approximation to the exact solution of the corrector (1.4) to be provided by some predictor formula (in order to avoid confusion, we denote from now on the exact solution of the corrector (1.4) by U_{n+1}). Assuming that the predictor formula is explicit, we obtain an *explicit* step-by-step method for approximating the exact solution at the step points t_{n+1}. Predictors of the form (1.3) can be used, provided that the PC pair {(1.3), (1.4)} employ identical block point vectors c.

The k components of the vectors $Y^{(j)}$ can be computed in parallel, so that the computational time (wall-clock time) needed for one iteration of (1.5) roughly equals the time required tc evaluate one right-hand side function on a sequential computer. Hence, given the initial prediction $Y^{(0)}$, the *sequential* costs of (1.5) per integration step are $m + 1$ sequential right-hand side evaluations originating from the evaluation of $f(Y_n)$ and $f(Y^{(j-1)})$, $j = 1, 2, \ldots, m$. To these $m + 1$ sequential corrector stages one should add the sequential predictor stages required by the predictor. If (1.3) is used as a predictor, then no additional stages are required because $f(Y_n)$ is both predictor and corrector stage. Hence, the PC pair {(1.3), (1.4)} has $\sigma^* = m + 1$.

A rigorous convergence analysis of a general class of PC iteration methods, including the iteration (1.5), may be found in Burrage [5] and in Jackson and Nørsett [17]. However, a first indication of the rate of convergence of PC methods can be obtained by considering the recursion

$$\begin{aligned} Y^{(j)} - U_{n+1} &= hC\left[f(Y^{(j-1)}) - f(U_{n+1})\right] \\ &\approx Z\left[Y^{(j-1)} - U_{n+1}\right], \quad Z := h\lambda C, \end{aligned} \tag{1.6}$$

where λ denotes an approximation to the derivative $\partial f/\partial y$ at y_n (we remark that in the case of *systems* of ODEs, λ should be understood to run through the spectrum of the Jacobian $\partial f/\partial y$). This recursion shows that each iteration reduces the iteration error by a factor of $O(h)$. Therefore, one usually chooses PC pairs in which the order of the predictor is not much less than that of the corrector so that the order of the corrector is attained within a few iterations. Table 2 is the analogue of Table 1 for PC methods with a single iteration (PECE methods with $\sigma^* = 2$).

This table shows that, for a given number of starting values, increasing the order of accuracy reduces the size of the stability regions.

Table 2
Stable PECE methods of the form (1.5)

References	k	p	c^T	$P=S$	β_{real}	β_{imag}
Two-step Adams–Bashforth–Moulton	2	3	(0, 1)	1	2.40	1.20
Van der Houwen–Sommeijer [33, method {(4.3), (4.6)}]	2	4	$(1+1/\sqrt{5}, 1)$	2	0.12	0.11
Three-step Adams–Bashforth–Moulton	3	4	(−1, 0, 1)	1	1.93	1.18
Donelson–Hansen [10, Table 2]	3	6	(1/3, 2/3, 1)	3		
Van der Houwen–Sommeijer [33, method {(4.12), (4.13)}]	3	6	(0, 4, 1)	2	1.77	0.58
Chu–Hamilton [8, method {(2.7), (2.9)}]	4	3	(1/4, 1/2, 3/4, 1)	2	4.98	
Shampine–Watts–Worland [26, 39]	4	4	(−1/2, 0, 1/2, 1)	2	0.88	1.16
Chu–Hamilton [8, method {(2.11), (2.13)}]	4	4	(1/4, 1/2, 3/4, 1)	2	3.34	
Four-step Adams–Bashforth–Moulton	4	5	(−2, −1, 0, 1)	1	1.41	0.92
Donelson–Hansen [10, Table 2]	4	8	(1/4, 1/2, 3/4, 1)	4		
Van der Houwen–Sommeijer [33, method {(4.14), (4.15)}]	4	8	(−1, 0, 5/2, 1)	2	0.30	0.14

A possible approach for improving the stability region starts with correctors with a large stability region which is then sufficiently often iterated to obtain more or less the corrector solution. In order to achieve that (1.6) converges rapidly to the corrector solution, some norm of the iteration matrix Z should be small. Taking the spectral radius of Z, i.e. $\rho(Z) = h\,|\,\lambda\,|\,\rho(C)$, as a measure for the rate of convergence, we are led to find correctors with small $\rho(C)$ possessing large stability regions. For example, in [27] we find the block θ-method of order k:

$$\begin{aligned} &Y_{n+1} = Y_n + (1-\theta)hBf(Y_n) + \theta hBf(Y_{n+1}), \quad 0 < \theta \leqslant 1, \\ &c^T := (1/k, 2/k, \ldots, 1), \end{aligned} \tag{1.7}$$

where B has all its eigenvalues at 1, and whose linear stability region is identical with that of the conventional θ-method. Since $\rho(C) = \theta$, convergence is improved if θ is decreased. A second example is the iteration of highly stable RK-type block methods (see Section 1.2.1).

A generalization of the PC iteration (1.5) is based on the widely used technique of preconditioning (or smoothing) of the residual term in (1.5), that is, the premultiplication of the term in square brackets by a k-by-k preconditioning matrix P (possibly depending on j). This leads to

$$Y^{(j)} - U_{n+1} \approx Z\big[Y^{(j-1)} - U_{n+1}\big], \quad Z := I - P + h\lambda PC. \tag{1.6'}$$

Let us choose P such that $P = I + hPC\Omega$, where Ω is a $k \times k$ matrix with bounded elements. Then

$$Z = hPC(\lambda I - \Omega), \quad P := (I - hC\Omega)^{-1},$$

so that each iteration reduces the magnitude of the iteration error by h. Ideally, the matrix Ω should be chosen such that $\rho(Z)$ is minimized on the spectrum of $\partial f/\partial y$. A perhaps more practical approach is to compute the dominant value of λ during the iteration in one step and to choose Ω in the next step such that its eigenvalues coincide with this dominant λ-value.

Another possibility allows P (and therefore Ω and Z) to depend on j and sets $\Omega_j = \omega_j I$, with ω_j scalar. Then we can write $\rho(\Pi Z_j)$ in the factorized form

$$\rho(\Pi Z_j) = |P_m(\lambda)|\, h^m \rho\big(C^m \Pi (I - \omega_j hC)^{-1}\big), \quad P_m(\lambda) := \Pi(\lambda - \omega_j).$$

In first approximation (i.e., for $\omega_j h$ small), this leads to a minimax problem for the polynomial $P_m(\lambda)$ which can be solved in terms of shifted Chebyshev polynomials and which leads to explicit expressions for the parameters ω_j (see Manteuffel [21]). Notice that the introduction of P_j hardly increases the computational costs of the iteration scheme.

1.2.1. Runge–Kutta correctors

Of particular interest is the case where the corrector (1.4) is a block method of RK type. We shall call such RK-based PC methods *PIRK methods* (parallel iterated RK methods). The idea of PC iteration of implicit RK methods (IRK methods) to exploit parallelism goes back to Nørsett and Simonsen [24] and Jackson and Nørsett [16] and was elaborated in [15,17–19,30].

Here, we shall restrict our considerations to PIRK methods without preconditioning and we shall use the "last step value predictor"

$$Y^{(0)} = y_n e. \tag{1.8}$$

For s-stage RK correctors this PIRK method is itself an (explicit) RK method with $s^* = ms + 1$ stages, but with only $\sigma^* = m + 1$ *sequential* stages. It can be proved (see Jackson and Nørsett [16–18]) that the (global) order of y_{n+1} equals $p^* := \min\{p, m + 1\}$. Thus, we have the theorem:

Theorem 1.1. *Let* (1.4) *define an s-stage RK method of order p. Then the PIRK method* {(1.5), (1.8)} *represents an* $(ms + 1)$*-stage explicit RK method of order* $p^* := \min\{p, m + 1\}$ *requiring* $\sigma^* = m + 1$ *sequential stages.*

The observation that explicit RK methods of order p^* require at least p^* sequential stages per step point (see Iserles and Nørsett [15]) justifies the following definition:

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

In Nørsett and Simonsen [24] the question was posed whether it is possible to find optimal RK methods of any order p^*. Setting $m = p - 1$, it follows from Theorem 1.1 that this question can be answered in the affirmative: any pth-order RK method of the form (1.4) generates an optimal RK method of the form {(1.5), (1.8)}.

The next question is to find the least number of processors needed to implement an optimal explicit RK method of given order p. For example, the fifth-order, six-stage RK method of Butcher mentioned in [24] is an example of such a "minimal processor" method: it can be implemented on two processors requiring only five sequential stages. So far, the question of least number of necessary processors is not yet answered. However, we can immediately deduce a lower bound for the number of processors needed to implement optimal RK methods of the form {(1.5), (1.8)}: it is well known that, within the class of RK methods, those of Gauss–

Legendre type require the least number of stages to obtain a given order; to be more precise, s-stage Gauss–Legendre methods have order $p = 2s$. Hence, we have the following theorem [30]:

Theorem 1.2. *The PIRK method* {(1.5), (1.8)} *with* $m = p - 1$, *generated by the pth-order Gauss–Legendre method with* $s = p/2$ *stages* (p *even*) *or by the pth-order Radau IIA method with* $s = (p+1)/2$ *stages* (p *odd*), *is an explicit Runge–Kutta method of order* $p^* = p$ *with* $s^* = ps - s + 1$ *stages, which is optimal on* $\lfloor (p^*+1)/2 \rfloor$ *processors.*

1.2. Richardson 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 [9]). Here, we describe the use of extrapolation for the construction of optimal RK methods.

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 stepsize 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*. Let the generating function possess an asymptotic expansion in powers of h^q, where $q = 2$ if the method providing the values $y(t_0 + H, h)$ is a symmetric method and $q = 1$ otherwise. Using the harmonic Romberg sequence $\{1, 2, 3, \ldots\}$, the first step of the corresponding r-point extrapolation method is defined by (see e.g. [13])

$$y_1 = \sum_{i=1}^{r} c_i y\left(t_0 + H, \frac{h_0}{i}\right), \qquad \sum_{i=1}^{r} c_i = 1, \qquad \sum_{i=1}^{r} \frac{c_i}{i^j} = 0, \quad j = p, p+q, \ldots, p+(r-2)q. \tag{1.9}$$

Evidently, y_1 approximates $y(t)$ at the point $t_1 = t_0 + H$. 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. If H is fixed (for example, H is the whole integration interval), then (1.9) is said to define a *global* extrapolation process. If H is a function of h_0 (for example, $H = h_0$), then (1.9) is said to define a *local* extrapolation process.

It is clear that the computation of the various terms in the formula (1.9) for y_1 can be performed in parallel. Assuming that the computational effort for computing $y(t_0 + H, h_0/i)$ is proportional to i, we are led to compute $y(t_0 + H, h_0/r)$ on the first processor, $y(t_0 + H, h_0)$ and $y(t_0 + H, h_0/(r-1))$ on the second processor, etc. In this way the number of processors is minimized and given by $\lfloor (r+2)/2 \rfloor$.

The following theorem holds for the extrapolated method (1.9) (see e.g. [13]):

Theorem 1.3. *Let the generating method providing the values* $y(t_0 + H, h_0/i)$ *be of order* p, *then the extrapolation method defined by* (1.9) *has order* $p^* = p + (r-1)q$.

Let us consider the case where the function $y(t_0 + H, h)$ is defined by the midpoint rule:

$$\begin{aligned} Y_1 &= y_0 + hf(y_0), \\ Y_j &= Y_{j-2} + 2hf(Y_{j-1}), \qquad j = 2, 3, \ldots, m, \quad m = H/h, \\ y(t_0 + H, h) &= Y_m \end{aligned} \tag{1.10}$$

and let us apply local Richardson extrapolation with $H = 2h_0$. Then the following theorem holds [31]:

Theorem 1.4. *The Richardson–midpoint method* {(1.9), (1.10)} *with* $H = 2h_0$ *is an explicit Runge–Kutta method of order* $p^* = 2r$ *with* $s^* = r^2 + 1$ *stages per step of length* H, *which is optimal on* $\lfloor (p^* + 4)/4 \rfloor$ *processors.*

A comparison with Theorem 1.2 reveals that for $p^* > 5$ the Richardson–midpoint method requires less processors to be optimal than the Gauss–Legendre-based PIRK methods. For example, an optimal RK method of order ten requires only three processors when using Richardson extrapolation of (1.10) and five processors when using PC iteration of the tenth-order Gauss–Legendre method.

2. Stiff problems

In this section we shall consider parallel step-by-step methods that are suitable for integrating *stiff* first-order IVPs. Such methods are necessarily implicit. However, all methods discussed below require the solution of systems whose dimension does not exceed the dimension of the IVP. The methods are respectively based on (i) diagonally implicit block-by-block calculations, (ii) diagonally implicit iteration of (1.4), and (iii) local Richardson extrapolation of the implicit Euler method and of the trapezoidal rule. For similar methods for integrating special second-order IVPs, we refer to [36,37].

2.1. Diagonally implicit block methods

We shall call the method a *diagonally implicit block* method (DIB method) if $C = D$, where D is a diagonal matrix. On parallel processors, DIB methods require the same sequential costs as required by the celebrated backward differentiation formulas (BDFs).

The particular family of DIB methods of RK type is identical with the family of "strictly diagonal" IRK methods studied by Jackson and Nørsett [17]. They proved that this family contains only methods of order at most two (for linear problems, the order can be raised to $s + 1$, s being the number of stages of the IRK method). However, in the class of *general* DIB methods we can find methods of higher order and with (linear) stability regions that are considerably larger than those of the higher-order BDFs. In order to characterize the stability region we use the stability definition:

Definition. A method is said to be $A(\alpha, \beta, \gamma)$-*stable* if

(i) its region of stability contains the infinite wedge $\{z: -\alpha < \pi - \arg(z) < \alpha\}$, $0 < \alpha \leqslant \pi/2$, and all points in the nonpositive halfplane with $|z| > \beta$;

Table 3
Values of α, β, γ, and δ for the BDFs and for DIB methods

Method	c^{T}	Order p	α	β	γ	δ
BDF_3	$(-1, 0, 1)$	3	88.4°	1.94	1.046	0
DIB_3	$(21/10, 1)$	3	90°	0	0	0.94
BDF_4	$(-2, -1, 0, 1)$	4	73.2°	4.72	0.191	0
DIB_4	$(5, 13/4, 1)$	4	90°	0	0	0.92
DIB_4	$(3, 5, 1)$	4	90°	0	0	0.37
BDF_5	$(-3, -2, -1, 0, 1)$	5	51.8°	9.94	1.379	0
DIB_5	$(-2.747, -2.122, 1)$	5	$>89.9°$	0.16	1.0000026	0.993
DIB_5	$(1.6153, 4.7871, 1)$	5	$>89.9°$	0.30	1.000069	0.89

(ii) γ is the maximum value of the spectral radius of $M(z) := (I - zC)^{-1}(A + zB)$ when z runs through the region of instability lying in the nonpositive halfplane.

Table 3 compares the values α, β, γ, and $\delta := \rho(M(-\infty))$ of the BDFs and of a few DIB methods derived in [29].

Iserles and Nørsett [15] and Jackson and Nørsett [17] have generalized DIB methods by replacing the entries of C by (possibly rectangular) matrices and by requiring the matrices appearing on the diagonal to be (square) diagonal matrices. For RK-type methods they derived a number of order results and derived examples of parallel methods with good stability properties.

2.2. Diagonally implicit iteration

In this section, the PC iteration (1.5) is replaced by the diagonally implicit iteration scheme

$$Y^{(j)} - hDf(Y^{(j)}) = Y^{(j-1)} - hDf(Y^{(j-1)}) - \left[Y^{(j-1)} - AY_n - hBf(Y_n) - hCf(Y^{(j-1)})\right], \quad j = 1, 2, \ldots, m, \quad (2.1)$$

$$Y_{n+1} := Y^{(m)}, \qquad y_{n+1} := e_k^{\mathrm{T}} Y_{n+1},$$

where D is a diagonal matrix with positive entries. Evidently, by virtue of the diagonal structure of D, the method (2.1) is highly parallel, because in each iteration the k components of the iterate $Y^{(j)}$ can be computed concurrently. It will be assumed that the (components of the) iterates are computed by one or more iterations in a modified Newton process. This Newton iteration process will be called *inner* iteration and the iteration process (2.1) with iteration index j will be called *outer* iteration. As in the case of PC iteration, predictors of the form (1.3) can be used, provided that their block points equal those of the corrector. However, as the iteration scheme is itself implicit, one may also consider implicit predictors. We shall say that the method has σ^* sequential (diagonally implicit) stages per step if in that step σ^* sequential implicit relations are to be solved.

As in the case of PC iteration, one may again apply residual conditioning by inserting a matrix P in front of the residual term (in square brackets) occurring in (2.1). An alternative modification of (2.1) was suggested by Butcher [7]. In the notation used here, this modification

inserts a $k \times k$ matrix Q in front of those vectors $Y^{(j)}$ and $Y^{(j-1)}$ that appear outside the residual term. A combination of preconditioning and the Butcher modification seems to be an efficient approach for accelerating the convergence of diagonal iteration. The corresponding iteration scheme reads

$$\begin{aligned} &Y^{(j)} = Q^{-1}X^{(j)}, \quad j = 1, 2, \ldots, m, \\ &X^{(j)} - hDf(X^{(j)}) = X^{(j-1)} - hDf(X^{(j-1)}) \\ &\qquad - P\big[Q^{-1}X^{(j-1)} - AY_n - hBf(Y_n) - hCf(Q^{-1}X^{(j-1)})\big] \qquad (2.1') \\ &Y_{n+1} := Y^{(m)}, \qquad y_{n+1} := e_k^{\mathrm{T}} Y_{n+1}. \end{aligned}$$

Notice that in the case of PC iteration (i.e., $D = O$), the Butcher modification reduces to preconditioning of the residual by $Q^{-1}P$. The analogue of the error recursion (1.6') is given by

$$\begin{aligned} &Y^{(j)} - U_{n+1} \approx Z\big[Y^{(j-1)} - U_{n+1}\big], \\ &Z := Q^{-1}[I - hD\lambda]^{-1}[Q - P + h(PC - DQ)\lambda]. \end{aligned}$$

The choice $Q - P = h(DQ - PC)\Omega$ is of particular interest and leads to

$$\begin{aligned} &Z = Q^{-1}h\lambda D(I - h\lambda D)^{-1}(D^{-1}PC - Q)(I - \lambda^{-1}\Omega), \\ &P := (Q - hDQ\Omega)(I - hC\Omega)^{-1}, \end{aligned}$$

where the matrices D, Ω, and Q are still free. Assuming that the diagonal entries of D are positive, we find

$$\|Z\| \leqslant \zeta(h, \lambda), \quad \zeta(h, \lambda) := \|Q^{-1}\| \, \|D^{-1}PC - Q\| \, \|I - \lambda^{-1}\Omega\| \, \frac{d|h\lambda|}{1 + d|h\lambda|},$$

where d is the maximal diagonal entry of D.

2.2.1. Runge–Kutta correctors

So far, we only investigated diagonal iteration of the form (2.1) using correctors of RK type. In order to be compatible with the notation used in earlier papers on this topic, we shall change to the familiar notation adopted for RK methods, that is, the corrector (1.4) and the step point formula in (1.3) are written in the form

$$Y = y_n e + hAf(Y), \qquad y_{n+1} = y_n + hb^{\mathrm{T}} f(Y),$$

and the method (2.1) assumes the form

$$Y^{(j)} - hDf(Y^{(j)}) = y_n e + h(A - D)f(Y^{(j-1)}), \quad j = 1, 2, \ldots, m, \qquad (2.2)$$

$$y_{n+1} = y_n + hb^{\mathrm{T}} f(Y^{(m)}). \qquad (2.3)$$

For stiff problems, the right-hand side evaluations in the step point formula (2.3) may give rise to instabilities and, consequently, the accuracy may be reduced considerably (cf. [25]). However, if the underlying corrector in (2.2) is stiffly accurate, that is $b^{\mathrm{T}}A^{-1} = e_s^{\mathrm{T}}$ (e_s denoting the sth unit vector and s being the number of stages of the corrector), then one may replace the step point formula by the "last component" formula

$$y_{n+1} = e_s^{\mathrm{T}} Y^{(m)}, \qquad (2.4)$$

Table 4
Stability results for PDIRK methods

Predictor (2.5)	Type	Corrector	$p^* = p$	$\sigma^* = m+1$	Stability
$B = D = \text{diag}(Ae - Ce)$,	I	Radau IIA	p	$p-2$	Finite stability region
$BAe = A^2e$	II	Radau IIA	3	2	A-stable
		Radau IIA	5	4	A(α)-stable, $\alpha = 89.997°$
		Radau IIA	7	6	A(α)-stable, $\alpha = 89.95°$
$B = D = \text{diag}(Ae)$,	I	Radau IIA	3	2	Strongly A-stable
$C = O$		Radau IIA	5	4	Strongly A-stable
		Radau IIA	7	6	Strongly A(α)-stable, $\alpha = 83.3°$
	II	Radau IIA	3	3	L(α)-stable, $\alpha = 89.75°$
		Radau IIA	5	5	L(α)-stable, $\alpha = 89.12°$
		Radau IIA	7	7	L(α)-stable, $\alpha = 89.02°$

which does not contain right-hand side evaluations anymore. This formula is much more stable and therefore we shall confine our considerations to stiffly accurate correctors. The methods using (2.3) and (2.4) will be called *PDIRK methods* (parallel diagonally implicitly iterated RK methods) of type I and type II, respectively.

In [35], RK correctors were iterated using predictors of the form

$$\mathbf{Y}^{(0)} - hBf(\mathbf{Y}^{(0)}) := y_n e + hCf(y_n e), \tag{2.5}$$

where either $B = O$ or $B = D$ and where C is an arbitrary full matrix. For given m, this PDIRK method belongs to the class of DIRK methods. However, when implemented on a parallel system, the method is effectively an SDIRK method because each processor has to compute just one LU decomposition for the inner iteration process.

Given the underlying corrector, there remains the choice of the matrix D, the number of iterations m, and the matrices B and C in the predictor formula. If we do not want to solve the corrector until convergence, but instead want an efficient method after a *fixed* number of iterations, then the stability of the resulting method is crucial. Restricting ourselves to Radau IIA correctors, we find for such "fixed-number-of-iterations" methods the stability results listed in Table 4 (here, p and p^* denote the order of the corrector and the PDIRK method, respectively).

Next we consider fixed-number-of-iterations PDIRK methods where $B = D = dI$ with d a free parameter. For such methods, the following theorem can be proved:

Theorem 2.1. *For any corrector, there exist values of d such that the methods listed in Table 5 are L-stable.*

Table 5
L-stable PDIRK methods

Predictor (2.5)	Type	$p^* = p$	$\sigma^* = m+1$
$B = D = dI = \text{diag}(Ae - Ce)$	II	$1 \leqslant p \leqslant 6$, $p = 8$	p
$B = D = dI$, $C = O$	I	$1 \leqslant p \leqslant 6$, $p = 8$	p
$B = D = dI$, $C = O$	II	$1 \leqslant p \leqslant 8$, $p = 10$	$p+1$

A further possibility is to define PDIRK methods which leave the whole matrix D as a set of free parameters. We could try to exploit the increased freedom for improving the order of accuracy for a given value of m while preserving A- or L-stability (for example, we still miss an L-stable method of order seven requiring seven sequential stages). However, there is a drawback associated with all DIRK methods and, therefore, also with any PDIRK method using the predictor (2.5) and a fixed number of iterations. This is caused by the phenomenon of *order reduction* (c.f., e.g. [14]). Order reduction reduces the observed order of RK methods to their stage order (or their stage order plus one). Most DIRK methods are particularly sensitive to order reduction because their stage order is only one or two. Thus, instead of keeping the number of iterations fixed, it may be more efficient to iterate until the corrector solution is approximated sufficiently close, so that the stage order of the PDIRK method may be considered to be equal to that of the corrector (recall that s-stage Gauss–Legendre, Lobatto IIIA and Radau IIA methods all have stage order s). In this approach, the matrix D should be chosen such that the rate of convergence is improved, rather than increasing the order of accuracy for a given value of m. In [32] fast converging PDIRK methods have been constructed be minimizing the spectral radius of the matrix $D^{-1}A - I$. These "minimal spectral radius" methods were tested on a number of stiff problems and turn out to be much more efficient than the fixed-number-of-iterations methods discussed above. The main results derived in [32] and an error analysis of minimal-spectral-radius PDIRK methods are presented in [34].

2.3. Richardson extrapolation

Similar to the construction of parallel nonstiff methods by Richardson extrapolation (see Section 1.3), we can construct parallel stiff methods, just by choosing a stiff method for defining the function $y(t_0 + H, h)$ (see [31]). For example, consider the generating method defined by the backward Euler method

$$\begin{aligned} &Y_0 = y_0, \\ &Y_j = Y_{j-1} + hf(Y_j), \qquad j = 1, 2, \ldots, m, \quad m = H/h, \\ &y(t_0 + H, h) = Y_m, \end{aligned} \tag{2.6}$$

and let us apply local Richardson extrapolation with $H = h_0$.

Theorem 2.2. *The Richardson–Euler method* {(1.9), (2.6)} *with* $H = h_0$ *is a DIRK method of order* $p^* = r$ *with* $s^* = r(r+1)/2$ *diagonally implicit stages per step of length* H *and* $\sigma^* = p^*$ *sequential singly diagonally implicit stages on* $\lfloor (p^* + 2)/2 \rfloor$ *processors.*

These Richardson–Euler methods can be shown to be L(α)-stable where α decreases almost monotonically from 90° for $r = 2$ to 89.83° for $r = 10$.

Finally, we consider extrapolation methods generated by the trapezoidal rule:

$$\begin{aligned} &Y_0 = y_0, \\ &Y_j = Y_{j-1} + \tfrac{1}{2}h\big[f(Y_{j-1}) + f(Y_j)\big], \qquad j = 1, 2, \ldots, m, \quad m = H/h, \\ &y(t_0 + H, h) = Y_m. \end{aligned} \tag{2.7}$$

Theorem 2.3. *The Richardson–trapezoidal method* {(1.9), (2.7)} *with* $H = 2h_0$ *is a DIRK method of order* $p^* = 2r$ *with* $s^* = r^2 + 1$ *diagonally implicit stages and* $\sigma^* = p^*$ *sequential singly diagonally implicit stages on* $\lfloor (p^* + 4)/4 \rfloor$ *processors.*

These methods are A(α)-stable with α decreasing from 79.2° for $r = 2$ to 50.6° for $r = 8$.

References

[1] A. Bellen, Parallelism across the steps for difference and differential equations, in: A. Bellen, C.W. Gear and E. Russo, eds., *Numerical Methods for Ordinary Differential Equations, Proceedings L'Aquila Symposium,* Lecture Notes in Mathematics 1386 (Springer, Berlin, 1989) 22–35.

[2] A. Bellen, Z. Jackiewicz and M. Zennaro, Stability analysis of time-point relaxation Heun method, Rept. 1/30, Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo", C.N.R. (1990)

[3] A. Bellen, R. Vermiglio and M. Zennaro, Parallel ODE-solvers with stepsize control, *J. Comput. Appl. Math.* 31 (1990) 277–293.

[4] K. Burrage, The error behaviour of a general class of predictor–corrector methods, CMSR Rept., University of Liverpool, England (1989).

[5] K. Burrage, Solving nonstiff IVPs in a transputer environment, *Appl. Numer. Math.* 8 (1991) 201–216.

[6] J.C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge–Kutta and General Linear Methods* (Wiley, New York, 1987).

[7] J.C. Butcher, Private communication, Pre-Dundee Symposium, Manchester, England (1991).

[8] M.T. Chu and Hamilton, Parallel solution of ODE's by multi-block methods, *SIAM J. Sci. Statist. Comput.* 3 (1987) 342–353.

[9] P. Deuflhard, Recent progress in extrapolation methods for ordinary differential equations, *SIAM Rev.* 27 (1985) 505–535.

[10] J. Donelson and E. Hansen, Cyclic composite multistep predictor-corrector methods, *SIAM Numer. Anal.* 8 (1971) 137–157.

[11] C.W. Gear, Massive parallelism across time in ODEs, in: *Proceedings International Conference on Parallel Methods for Ordinary Differential Equations*, Grado, Italy (1991); also: *Appl. Numer. Math.* 11 (1993) 27–43 (this issue).

[12] C.W. Gear and Xu Xuhai, Parallelism across space in ODEs, in *Proceedings International Conference on Parallel Methods for Ordinary Differential Equations*, Grado, Italy (1991); also: *Appl. Numer. Math.* 11 (1993) 45–68 (this issue).

[13] E. Hairer, S.P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations* I: *Nonstiff problems* (Springer, Berlin, 1987).

4] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations,* II: *Stiff and Differential-Algebraic Problems* (Springer, Berlin, 1991).

5] A. Iserles and S.P. Nørsett, On the theory of parallel Runge–Kutta methods, *IMA J. Numer. Anal.* 10 (1990) 463–488.

[16] K.R. Jackson and S.P. Nørsett, Parallel Runge–Kutta methods, Manuscript (1988).

[17] K.R. Jackson and S.P. Nørsett, The potential for parallelism in Runge–Kutta methods, Part 1: RK formulas in standard form, Tech. Rept. No. 239/90, Department of Computer Science, University of Toronto, Toronto, Ont. (1990).

[18] K.R. Jackson and S.P. Nørsett, The potential for parallelism in Runge–Kutta methods, Part 2: RK predictor–corrector formulas (in preparation).

[19] I. Lie, Some aspects of parallel Runge–Kutta methods, Rept. No. 3/87, Division Numerical Mathematics, University of Trondheim, Norway (1987).

[20] Lianhua Lu, The stability of block θ-methods, *IMA J. Numer. Anal.* (submitted).

[21] T.A. Manteuffel, The Tchebyshev iteration for nonsymmetric linear systems, *Numer. Math.* 28 (1977) 307–327.

[22] W.L. Miranker and W. Liniger, Parallel methods for the numerical integration of ordinary differential equations, *Math. Comp.* 21 (1967) 303–320.

[23] S.P. Nørsett, Semi-explicit Runge–Kutta methods, Rept. Mathematics and Computation No. 6/74, Department of Mathematics, University of Trondheim, Norway (1974).
[24] S.P. Nørsett and H.H. Simonsen, Aspects of parallel Runge–Kutta methods, in: A. Bellen, C.W. Gear and E. Russo, eds., *Numerical Methods for Ordinary Differential Equations, Proceedings L'Aquila Symposium,* Lecture Notes in Mathematics 1386 (Springer, Berlin, 1989) 103–107.
[25] L.F. Shampine, Implementation of implicit formulas for the solution of ODEs, *SIAM J. Sci. Statist. Comput.* 1 (1980) 103–118.
[26] L.F. Shampine and H.A. Watts, (1969) Block implicit one-step methods, *Math. Comp.* 23 (1969) 731–740.
[27] B.P. Sommeijer, Explicit high-order Runge–Kutta–Nyström methods for parallel computers (in preparation).
[28] B.P. Sommeijer, Stability boundaries of block Runge–Kutta methods (in preparation).
[29] B.P. Sommeijer, W. Couzy and P.J. van der Houwen, A-stable parallel block methods for ordinary and integro-differential equations, *Appl. Numer. Math.* 9 (1992) 267–281.
[30] P.J. van der Houwen and B.P. Sommeijer, Parallel iteration of high-order Runge–Kutta methods with stepsize control, *J. Comput. Appl. Math.* 29 (1990) 111–127.
[31] P.J. van der Houwen and B.P. Sommeijer, Parallel ODE solvers, in: *Proceedings International Conference on Supercomputing*, Amsterdam, (ACM, New York, 1990).
[32] P.J. van der Houwen and B.P. Sommeijer, Iterated Runge–Kutta methods on parallel computers, *SIAM J. Sci. Statist. Comput.* 12 (1991) 1000–1028.
[33] P.J. van der Houwen and B.P. Sommeijer, Block Runge–Kutta methods on parallel computers, *Z. Angew. Math. Mech.* 72 (1992) 3–18.
[34] P.J. van der Houwen and B.P. Sommeijer, Analysis of parallel diagonally-implicit iteration of Runge–Kutta methods, in: *Proceedings International Conference on Parallel Methods for Ordinary Differential Equations,* Grado, Italy (1991); also: *Appl. Numer. Math.* 11 (1993) 169–188 (this issue).
[35] P.J. van der Houwen, B.P. Sommeijer and W. Couzy, Embedded diagonally implicit Runge–Kutta algorithms on parallel computers, *Math. Comp.* 58 (1992) 135–159.
[36] P.J. van der Houwen, B.P. Sommeijer and Nguyen huu Cong, Stability of collocation-based Runge–Kutta–Nyström methods, *BIT* 31 (1991) 469–481.
[37] P.J. van der Houwen, B.P. Sommeijer and Nguyen huu Cong, Parallel diagonally implicit Runge–Kutta–Nyström methods, *Appl. Numer. Math.* 9 (1992) 111–131.
[38] A. Wolfbrandt, A study of Rosenbrock processes with respect to order conditions and stiff stability, Ph.D. Thesis, Chalmers University of Technology, Göteborg, Sweden (1977).
[39] P.B. Worland, Parallel methods for the numerical solution of ordinary differential equations, *IEEE Trans. Comput.* 25 (1976) 1045–1048.