Performance of explicit pseudo two-step RKN methods on a shared memory computer *

Nguyen Huu Cong †, Helmut Podhaisky ‡ and Rüdiger Weiner ‡

† *Faculty of Mathematics, Mechanics and Informatics, Hanoi University of Science*
334 Nguyen Trai, Thanh Xuan, Hanoi, Vietnam
‡ *FB Mathematik und Informatik, Martin-Luther-Universität Halle-Wittenberg*
Theodor-Lieser-Str. 5, D-06120 Halle, Germany

Abstract

The aim of this paper is to investigate the performance of two explicit pseudo two-step RKN methods (EPTRKN methods) of order 6 and 10 for nonstiff problems $\mathbf{y}''(t) = \mathbf{f}(t, \mathbf{y}(t))$, $\mathbf{y}(t_0) = \mathbf{y}_0$, $\mathbf{y}'(t_0) = \mathbf{y}'_0$. For expensive problems, the parallel implementation on a shared memory computer gives a good speed-up with respect to the sequential one. Furthermore, for numerical comparisons, we solve three test problems taken from the literature by these two EPTRKN methods and the efficient nonstiff code ODEX2 running on the same shared memory parallel computer. Comparing computing times for accuracies received shows that the two new EPTRKN methods are superior to ODEX2 code for all three test problems.

Key words: Parallel Runge-Kutta-Nyström methods, two-step Runge-Kutta-Nyström methods.

1 Introduction

The arrival of parallel computers influences the development of numerical methods for a nonstiff initial value problem (IVP) for systems of special second-order ordinary differential equations (ODEs)

$$\mathbf{y}''(t) = \mathbf{f}(t, \mathbf{y}(t)), \qquad \mathbf{y}(t_0) = \mathbf{y}_0, \qquad \mathbf{y}'(t_0) = \mathbf{y}'_0, \qquad \mathbf{y}, \mathbf{f} \in \mathbb{R}^d. \tag{1.1}$$

The most efficient numerical methods for solving this problem are the explicit Runge-Kutta-Nyström (RKN) and extrapolation methods. In the literature, sequential explicit RKN methods up to order 11 can be found in e.g., [14, 15, 16, 17, 18, 19, 21, 26]. In order to exploit the facilities of parallel computers, a number of parallel explicit methods have been investigated, for example in [2, 3, 4, 5, 6, 9, 10, 11, 12]. A common challenge in the latter mentioned works is to reduce, for a given order of accuracy, the required number of effective sequential $\mathbf{f}$-evaluations per step, using parallel processors.
In previous work of Cong et al. [12], a general class of explicit pseudo two-step RKN methods (EPTRKN methods) for solving problems of the form (1.1) has been investigated. These

*This work was partly supported by NRPFS and the University of Halle.

EPTRKN methods are ones of the cheapest parallel explicit methods in terms of number of effective sequential **f**-evaluations per step. They can be easily equipped with embedded formulas for a variable stepsize implementation (cf. [9]). With respect to the number of effective sequential **f**-evaluations for a given accuracy, the EPTRKN methods have been shown to be much more efficient than most efficient sequential and parallel methods currently available for solving (1.1) (cf. [9, 12]).

Most numerical comparisons of parallel and sequential methods are done by means of the number of effective sequential **f**-evaluations for a given accuracy on a sequential computer ignoring the communication time between processors (cf. e.g., [1, 3, 5, 6]). In comparisons of different codes running on parallel computers, the parallel codes often give disappointing results. However, in our recent work [13], two parallel codes EPTRK5 and EPTRK8 of oder 5 and 8, respectively, have been proposed. These codes are based on EPTRK methods considered in [7, 8] which are a "first-order" version of the EPTRKN methods. The EPTRK5 and EPTRK8 codes have been shown to be more efficient than the DOPRI5 and DOP853 codes for solving expensive nonstiff first-order problems on a shared memory parallel computer. We have also obtained a similar performance of a parallel implementation of the BPIRKN codes for nonstiff special second-order problems (see [11]). These promising results encourage us to pursue the performance investigation of a real implementation of the EPTRKN methods on a parallel computer. This investigation consists of choosing relatively good embedded EPTRKN pairs, defining reasonable error estimate for stepsize strategy and comparing the resulting EPTRKN methods with the code ODEX2 which is among the most efficient sequential nonstiff integrators for special second-order ODE systems of the form (1.1). Although the class of EPTRKN methods contains methods of arbitrary high order, we consider only two EPTRKN methods of order 6 and 10 for numerical comparisons with the ODEX2 code.

We have to note that the choice of an implementation on a shared memory parallel computer is due to the fact that such a computer can consist of several processors sharing a common memory with fast data access requiring less communication times, which is suited to the features of the EPTRKN methods. In addition, there are the advantages of compilers which attempt to parallelize codes automatically by reordering loops and sophisticated scientific libraries (cf. e.g., [1]).

In order to see a possible speed-up of a parallel code, the test problems used in Section 3 should be expensive. Therefore, the relatively small problems have been enlarged by scaling.

2 Variable stepsize embedded EPTRKN methods

The EPTRKN methods have been recently introduced and investigated in [9, 12]. For an implementation with stepsize control, we consider variable stepsize embedded EPTRKN methods. Because EPTRKN methods are of a two-step nature, there is an additional difficulty in using these methods with variable stepsize mode. We overcome this difficulty by deriving methods with variable parameters (cf. e.g., [22, p. 397], [1, p. 44]). Thus, we consider the variable stepsize EPTRKN method (cf. [9])

$$\mathbf{Y}_n = \mathbf{e}\otimes\mathbf{y}_n + h_n\mathbf{c}\otimes\mathbf{y}'_n + h_n^2(A_n\otimes I)\mathbf{F}(t_{n-1}\mathbf{e}+h_{n-1}\mathbf{c},\mathbf{Y}_{n-1}), \tag{2.1a}$$

$$\begin{aligned}\mathbf{y}_{n+1} &= \mathbf{y}_n + h_n\mathbf{y}'_n + h_n^2(\mathbf{b}^T\otimes I)\mathbf{F}(t_n\mathbf{e}+h_n\mathbf{c},\mathbf{Y}_n),\\ \mathbf{y}'_{n+1} &= \mathbf{y}'_n + h_n(\mathbf{d}^T\otimes I)\mathbf{F}(t_n\mathbf{e}+h_n\mathbf{c},\mathbf{Y}_n),\end{aligned} \tag{2.1b}$$

with variable stepsize $h_n = t_{n+1} - t_n$ and variable parameter matrix A_n. This EPTRKN method is conveniently specified by the following tableau:

$$\begin{array}{c|c|c} A_n & \mathbf{c} & O \\ \hline \mathbf{0}^T & \mathbf{y}_{n+1} & \mathbf{b}^T \\ \mathbf{0}^T & \mathbf{y}'_{n+1} & \mathbf{d}^T \end{array}.$$

At each step, $2s$ $\mathbf{f}$-evaluations of the components of the big vectors $\mathbf{F}(t_{n-1}\mathbf{e}+h_{n-1}\mathbf{c},\mathbf{Y}_{n-1}) = (\mathbf{f}(t_{n-1}+c_ih_{n-1},\mathbf{Y}_{n-1,i}))$ and $\mathbf{F}(t_n\mathbf{e}+h_n\mathbf{c},\mathbf{Y}_n) = (\mathbf{f}(t_n+c_ih_n,\mathbf{Y}_{n,i}))$, $i=1,\ldots,s$ are used in the method. However, s $\mathbf{f}$-evaluations of the components of $\mathbf{F}(t_{n-1}\mathbf{e}+h_{n-1}\mathbf{c},\mathbf{Y}_{n-1})$ are already available from the preceding step. Hence, we need to compute only s $\mathbf{f}$-evaluations of the components of $\mathbf{F}(t_n\mathbf{e}+h_n\mathbf{c},\mathbf{Y}_n)$, which can be done in parallel. Consequently, on an s-processor computer, just one $\mathbf{f}$-evaluation is required per step. In this way, parallelization in an EPTRKN method is achieved by sharing the $\mathbf{f}$-evaluations of s components of the big vector $\mathbf{F}(t_n\mathbf{e}+h_n\mathbf{c},\mathbf{Y}_n)$ over a number of available processors. An additional computational effort consists of a recomputation of the variable parameter matrix A_n defined by (2.2e) below when the stepsize is changed.

2.1 Method parameters

The matrix A_n and the weight vectors $\mathbf{b}^T$ and $\mathbf{d}^T$ of the method (2.1) are derived by the order conditions (see [9, 12])

$$\frac{\tau_n^{j-1}\mathbf{c}^{j+1}}{j+1} - A_n j(\mathbf{c}-\mathbf{e})^{j+1} = \mathbf{0}, \quad j=1,\ldots,q, \tag{2.2a}$$

$$\frac{1}{j+1} - \mathbf{b}^T j\mathbf{c}^{j-1} = 0, \quad j=1,\ldots,p, \tag{2.2b}$$

$$\frac{1}{j} - \mathbf{d}^T\mathbf{c}^{j-1} = 0, \quad j=1,\ldots,p, \tag{2.2c}$$

where $\tau_n = h_n/h_{n-1}$ is the stepsize ratio. Notice that the conditions (2.2b), (2.2c) for $p=s$ define the weight vectors of a direct collocation-based IRKN method (cf. [24]).
For $q=p=s$, by defining the matrices and vectors

$$P = \Big(\frac{c_i^{j+1}}{j+1}\Big),\quad Q = \big(j(c_i-1)^{j-1}\big),\quad R = \big(jc_i^{j-1}\big),\quad S = \big(c_i^{j-1}\big),$$
$$D_n = \mathrm{diag}(1,\tau_n,\ldots,\tau_n^{s-1}),\quad \mathbf{v} = \Big(\frac{1}{j}\Big),\quad \mathbf{w} = \Big(\frac{1}{j+1}\Big),\quad i,j=1,\ldots,s,$$

the conditions (2.2) can be written in the form

$$A_nQ - PD_n = O, \quad \mathbf{w}^T - \mathbf{b}^T R = \mathbf{0}^T, \quad \mathbf{v}^T - \mathbf{d}^T S = \mathbf{0}^T, \tag{2.2d}$$

which implies the explicit formulas for the parameters of a EPTRKN method

$$A_n = PD_nQ^{-1}, \quad \mathbf{b}^T = \mathbf{w}^T R^{-1}, \quad \mathbf{d}^T = \mathbf{v}^T S^{-1}. \tag{2.2e}$$

If the step ratio τ_n is bounded from above (i.e., $\tau_n \le \Omega$), the function $\mathbf{f}$ is Lipschitz continuous, and if the conditions (2.2) are verified, then the method (2.1) has step point order p and stage order q at least equal to s for any collocation vector $\mathbf{c}$. It can attain the higher stage order $s+1$ and step point order up to $s+2$ if the vector $\mathbf{c}$ satisfies an orthogonality relation (cf. [9, Theorem 2.1]).

2.2 Embedded formulas

With the aim to have a cheap error estimate used in the stepsize selection for an implementation of EPTRKN methods with stepsize control, we shall equip the pth-order EPTRKN method (2.1) with the following embedded formula

$$\begin{aligned}\widehat{\mathbf{y}}_{n+1} &= \mathbf{y}_n + h_n\mathbf{y}'_n + h_n^2(\widehat{\mathbf{b}}^T \otimes I)\mathbf{F}(t_n\mathbf{e} + h_n\mathbf{c}, \mathbf{Y}_n),\\ \widehat{\mathbf{y}}'_{n+1} &= \mathbf{y}'_n + h_n(\widehat{\mathbf{d}}^T \otimes I)\mathbf{F}(t_n\mathbf{e} + h_n\mathbf{c}, \mathbf{Y}_n),\end{aligned} \tag{2.3}$$

where, the weight vectors $\widehat{\mathbf{b}}$ and $\widehat{\mathbf{d}}$ are determined by satisfying the following conditions which come from (2.2b) and (2.2c)

$$\frac{1}{j+1} - \widehat{\mathbf{b}}^T j\mathbf{c}^{j-1} = 0, \quad j = 1, \ldots, s-2, \quad \frac{1}{s} - \widehat{\mathbf{b}}^T (s-1)\mathbf{c}^{s-2} \neq 0, \tag{2.4a}$$

$$\frac{1}{j} - \widehat{\mathbf{d}}^T \mathbf{c}^{j-1} = 0, \quad j = 1, \ldots, s-1, \quad \frac{1}{s} - \widehat{\mathbf{d}}^T \mathbf{c}^{s-1} \neq 0. \tag{2.4b}$$

In the two EPTRKN codes considered in this paper, we use these embedded weight vectors defined as

$$\widehat{\mathbf{b}}^T = \big(\mathbf{w}^T - \frac{1}{10}\mathbf{e}_{s-1}^T\big)R^{-1}, \quad \widehat{\mathbf{d}}^T = \big(\mathbf{v}^T - \frac{1}{10}\mathbf{e}_s^T\big)S^{-1}, \tag{2.5}$$

where $\mathbf{e}_s^T = (0, \ldots, 0, 1)$ and $\mathbf{e}_{s-1}^T = (0, \ldots, 1, 0)$ are the s-th and $(s-1)$-th unit vectors. In this way we have an estimate for the local error of order $\widehat{p} = s-1$ without additional $\mathbf{f}$-evaluations given by

$$\mathbf{y}_{n+1} - \widehat{\mathbf{y}}_{n+1} = O(h_n^{\widehat{p}+1}), \qquad \mathbf{y}'_{n+1} - \widehat{\mathbf{y}}'_{n+1} = O(h_n^{\widehat{p}+1}). \tag{2.6}$$

Thus, we have defined the embedded EPTRKN method of orders $p(\widehat{p})$ given by (2.1) and (2.3) which can be specified by the tableau

$$\begin{array}{c|c|c} A_n & \mathbf{c} & O \\ \hline \mathbf{0}^T & \mathbf{y}_{n+1} & \mathbf{b}^T \\ \mathbf{0}^T & \mathbf{y}'_{n+1} & \mathbf{d}^T \\ \hline \mathbf{0}^T & \widehat{\mathbf{y}}_{n+1} & \widehat{\mathbf{b}}^T \\ \mathbf{0}^T & \widehat{\mathbf{y}}'_{n+1} & \widehat{\mathbf{d}}^T \end{array}.$$

Finally, we have to note that the approach used in the derivation of the embedded formula above is slightly different from the one used in [8, 9, 11, 13]. By this approach of constructing embedded EPTRKN methods, there exist several embedded formulas for an EPTRKN method.

2.3 Stability properties

Stability of (constant stepsize) EPTRKN methods was investigated by applying them to the model test equation $y''(t) = \lambda y(t)$, where λ runs through the eigenvalues of the Jacobian matrix $\partial\mathbf{f}/\partial\mathbf{y}$ which are assumed to be negative real. It is characterized by the spectral radius $\rho(M(x))$, $x = \lambda h^2$, of the $(s+2) \times (s+2)$ *amplification matrix* $M(x)$ defined by (cf. [12, Section 2.2])

$$M(z) = \begin{pmatrix} xA & \mathbf{e} & \mathbf{c} \\ x^2\mathbf{b}^T A & 1 + x\mathbf{b}^T\mathbf{e} & 1 + x\mathbf{b}^T\mathbf{c} \\ x^2\mathbf{d}^T A & x\mathbf{d}^T\mathbf{e} & 1 + x\mathbf{d}^T\mathbf{c} \end{pmatrix}. \tag{2.7a}$$

The stability interval of an EPTRKN method is given as

$$(-\beta_{stab}, 0) := \{x : \rho(M(x)) \leq 1\}. \tag{2.7b}$$

The stability intervals of the EPTRKN methods used in our numerical codes can be found in Section 3.

3 Numerical experiments

In this section we shall report the numerical results obtained by the sequential code ODEX2 and two our new parallel EPTRKN codes for comparing their efficiency.

3.1 Specifications of the codes

ODEX2 is an extrapolation code for special second-order ODEs of the form (1.1). It uses variable order and stepsize and is implemented in the same way as the ODEX code for first-order ODEs (cf. [22, p. 294, 298]). This code is recognized as being one of the most efficient sequential integrators for nonstiff problems like (1.1) (see [22, p. 484]). In the numerical experiments, we apply ODEX2 code with standard parameter settings.
Our first code uses a variable stepsize embedded EPTRKN method based on collocation vector $\mathbf{c} = (c_1, c_2, c_3, 1)^T$ which satisfies the relations

$$\int_0^1 x^{j-1} \prod_{i=1}^{4} (x - c_i) dx = 0, \qquad j = 1, 2 \tag{3.1a}$$

$$(\mathbf{b}^T + \mathbf{d}^T)\Big[\frac{\mathbf{c}^{s+2}}{s+2} - A(s+1)(\mathbf{c} - \mathbf{e})^s\Big] = 0, \tag{3.1b}$$

where (3.1a) is an orthogonality relation (cf [22, p. 212]), and (3.1b) is introduced for minimizing the stage error coefficients (cf. [27]). The resulting method is of step point order

6 and stage order 5 (cf. [9, Theorem 2.1], [12, Theorem 2.2]). It has 4 as the optimal number of processors, and an embedded formula of order 3 (see Section 2). Its stability interval as defined in Section 2.3 is determined by numerical search techniques to be $(-0.720, 0)$. This first code is denoted by EPTRKN4.

Our second code uses a variable stepsize embedded EPTRKN method based on collocation vector $\mathbf{c} = (c_1, \ldots, c_8)^T$ which is obtained by solving the system of equations

$$\int_0^1 x^{j-1} \prod_{i=1}^{4} (x - c_i) dx = 0, \qquad j = 1, 2, 3, \tag{3.2a}$$

$$c_4 = 1, c_{4+k} = 1 + c_k, \qquad k = 1, 2, 3, 4. \tag{3.2b}$$

Here (3.2a) is again an orthogonality relation. The resulting method is of step point order 10 and stage order 9 (cf. also [9, Theorem 2.1], [12, Theorem 2.2]). It has 8 as the optimal number of processors, and an embedded formula of order 7 (see also Section 2). Its stability interval is also determined by the numerical search techniques to be $(-0.598, 0)$. This second code is denoted by EPTRKN8.

Table 1 summarizes the main characteristics of the codes: the step point order p, the embedded order $\widehat{p}$, the optimal number of processors np and the stability interval $(-\beta_{stab}, 0)$. Both codes EPTRKN4 and EPTRKN8 are implemented using local extrapolation and direct

Table 1: EPTRKN codes used in the numerical experiments

Code names	p	$\widehat{p}$	np	$(-\beta_{stab}, 0)$
EPTRKN4	6	3	4	(-0.720, 0)
EPTRKN8	10	7	8	(-0.598, 0)

PIRKN methods based on the same collocation points (cf. [3]) as a starting procedure. The local error of order $\widehat{p}$ denoted by $LERR$ is estimated as

$$LERR = \sqrt{\frac{1}{d} \sum_{i=1}^{d} \Big[\Big(\frac{y_{n+1,i} - \widehat{y}_{n+1,i}}{ATOL + RTOL|y_{n+1,i}|} \Big)^2 + \Big(\frac{y'_{n+1,i} - \widehat{y}'_{n+1,i}}{ATOL + RTOL|y'_{n+1,i}|} \Big)^2 \Big]}.$$

The new stepsize h_{n+1} is chosen as

$$h_{n+1} = h_n \cdot \min\big\{2, \max\big\{0.5, 0.85 \cdot LERR^{-1/(\widehat{p}+1)}\big\}\big\}. \tag{3.3}$$

The constants 2 and 0.5 serve to keep the stepsize ratios $\tau_{n+1} = h_{n+1}/h_n$ to be in the interval $\big[0.5,\, 2\big]$.

The computations were performed on a HP-Convex X-Class Computer. The parallel EPTRKN4 and EPTRKN8 codes were implemented in sequential and parallel modes. They can be downloaded from http://www.mathematik.uni-halle.de/institute/numerik/software.

3.2 Numerical comparisons

The numerical comparisons in this section are mainly made in terms of computing time for an accuracy received. However, since the parameters of the two EPTRKN methods used in this paper are new, we would like to test the performance of these methods by comparing the number of **f**-evaluations for a given accuracy.

Test problems

For comparing the number of **f**-evaluations, we take two very well-known small test problems from the RKN literature.
FEHL - the nonlinear Fehlberg problem (cf. e.g., [14, 15, 17, 18])

$$\frac{d^2\mathbf{y}(t)}{dt^2} = \begin{pmatrix} -4t^2 & -\frac{2}{\sqrt{y_1^2(t)+y_2^2(t)}} \\ \frac{2}{\sqrt{y_1^2(t)+y_2^2(t)}} & -4t^2 \end{pmatrix} \mathbf{y}(t),$$

$$\mathbf{y}(\sqrt{\pi/2}) = (0,1)^T, \qquad \mathbf{y}'(\sqrt{\pi/2}) = (-2\sqrt{\pi/2},0)^T \qquad \sqrt{\pi/2} \le t \le 10,$$

with highly oscillating exact solution given by $\mathbf{y}(t) = (\cos(t^2), \sin(t^2))^T$.

NEWT - the two-body gravitational problem for Newton's equation of motion (see e.g., [28, p. 245], [25, 18])

$$\frac{d^2y_1(t)}{dt^2} = -\frac{y_1(t)}{\left(\sqrt{y_1^2(t)+y_2^2(t)}\right)^3}, \quad \frac{d^2y_2(t)}{dt^2} = -\frac{y_2(t)}{\left(\sqrt{y_1^2(t)+y_2^2(t)}\right)^3},$$

$$y_1(0) = 1-\varepsilon, \quad y_2(0) = 0, \quad y_1'(0) = 0, \quad y_2'(0) = \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}, \quad 0 \le t \le 20.$$

The solution components are $y_1(t) = \cos(u(t)) - \varepsilon$, $y_2(t) = \sqrt{(1+\varepsilon)(1-\varepsilon)}\,\sin(u(t))$, where $u(t)$ is the solution of Kepler's equation $t = u(t) - \varepsilon\,\sin(u(t))$ and ε denotes the eccentricity of the orbit. In this example, we set $\varepsilon = 0.9$.

For comparing the computing time, we take the following three "expensive" problems:
PLEI - the celestial mechanics problem from [22] which models the gravity forces between seven stars in 2D space. This modelling leads to a second-order ODE system of dimension 14. Because this system is too small, it is enlarged by a scaling factor $ns = 500$ to become the new one

$$\mathbf{e} \otimes \mathbf{y}''(t) = \mathbf{e} \otimes \mathbf{f}(t, \mathbf{y}(\mathbf{t})), \qquad \mathbf{e} \in \mathbb{R}^{ns}.$$

MOON - the second celestial mechanics example which is formulated in a similar way for 101 bodies in 2D space with coordinates x_i, y_i and masses m_i $(i = 0, \ldots, 100)$

$$x_i'' = \gamma \sum_{j=0, j\ne i}^{100} m_j(x_j - x_i)/r_{ij}^3, \quad y_i'' = \gamma \sum_{j=0, j\ne i}^{100} m_j(y_j - y_i)/r_{ij}^3,$$

where

$$r_{ij} = ((x_i - x_j)^2 + (y_i - y_j)^2)^{1/2}, \quad i,j = 0,\dots,100$$
$$\gamma = 6.672, m_0 = 60, m_i = 7\cdot 10^{-3}, \quad i = 1,\dots,100.$$

We integrate for $0 \le t \le 125$ with the initial data

$$x_0(0) = y_0(0) = x_0'(0) = y_0'(0) = 0$$
$$x_i(0) = 30\cos(2\pi/100i) + 400, \quad x_i'(0) = 0.8\sin(2\pi/100i)$$
$$y_i(0) = 30\sin(2\pi/100i), \quad y_i'(0) = -0.8\cos(2\pi/100i) + 1.$$

Here no scaling was needed because the right-hand side functions are very expensive.

WAVE - the semidiscretized problem for 1D hyperbolic equations (see [23]).

$$\frac{\partial^2 u}{\partial t^2} = gd(x)\frac{\partial^2 u}{\partial x^2} + \frac{1}{4}\lambda^2(x,u), \quad 0 \le x \le b, \quad 0 \le t \le 10,$$
$$\frac{\partial u}{\partial x}(t,0) = \frac{\partial u}{\partial x}(t,b) = 0,$$
$$u(0,t) = \sin\big(\frac{\pi x}{b}\big), \qquad \frac{\partial u}{\partial t}(0,x) = -\frac{\pi}{b}\cos\big(\frac{\pi x}{b}\big)$$

with

$$d(x) = 10\big[2 + \cos\big(\frac{2\pi x}{b}\big)\big], \quad \lambda = \frac{4\cdot 10^{-4} g|u|}{d(x)}, \quad g = 9.81, \quad b = 1000.$$

By using second-order central spatial discretization on an uniform grid with 40 inner points we obtain a nonstiff ODE system. In order to make this problem more expensive, we enlarge it by a scaling factor $ns = 100$.

Results and discussion

The three codes ODEX2, EPTRKN4 and EPTRKN8 were applied to the above test problems with $ATOL = RTOL = 10^{-1}, 10^{-2}, \dots, 10^{-11}, 10^{-12}$. The number of sequential **f**-evaluations (for **FEHL** and **NEWT** problems) and the computing time (for **PLEI**, **MOON** and **WAVE** problems) are plotted as a function of the global error ERR at the end point of the integration interval defined by

$$ERR = \sqrt{\frac{1}{d}\sum_{i=1}^{d}\Big(\frac{y_{n+1,i} - y(t_{n+1})_i}{ATOL + RTOL|y(t_{n+1})_i|}\Big)^2}.$$

For problems **PLEI**, **MOON** and **WAVE** without exact solutions in a closed form, we use the reference solution obtained by ODEX2 using $ATOL = RTOL = 10^{-14}$.
For problems **FEHL** and **NEWT** where we compare the number of **f**-evaluations for a given

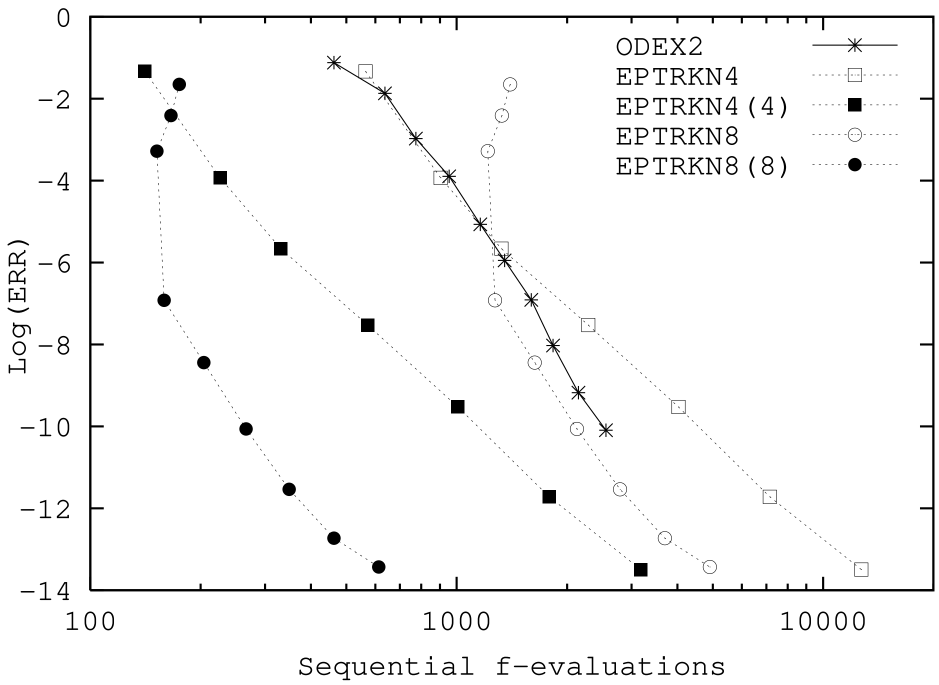

Figure 1: Results for **FEHL**

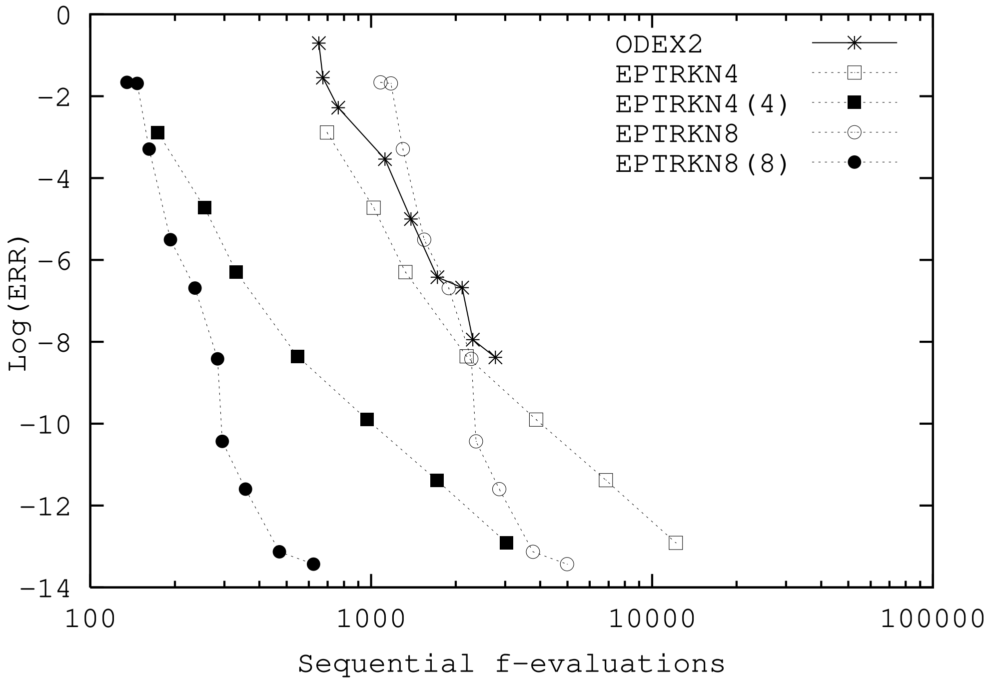

Figure 2: Results for **NEWT**

accuracy, the results in Fig. 1 - 2 show that on a sequential implementation mode (symbols associated with ODEX2, EPTRKN4 and EPTRKN8) the three codes are comparable. But on a parallel implementation mode, the two parallel codes EPTRKN4 and EPTRKN8 using the optimal number of processors 4 and 8, respectively (symbols associated with EPTRKN4(4) and EPTRKN8(8)) are by far superior to ODEX2, and the code EPTRKN8 is the most efficient.
For **PLEI**, **MOON** and **WAVE** problems where we compare the computing time, from the results plotted in Fig. 3 - 5 we see that for **PLEI** and **WAVE** problems, the two EPTRKN codes are competitive with or even more efficient than ODEX2 in the sequential implementation mode. But the parallelized EPTRKN4 and EPTRKN8 codes are again superior to ODEX2, and the results for EPTRKN4 and EPTRKN8 are almost comparable. For **MOON**, EPTRKN4 is again competitive with ODEX2 in the sequential implementation mode. Compared with ODEX2, the parallelized EPTRKN4 and EPTRKN8 show the same efficiency as for the problems **PLEI** and **WAVE**.

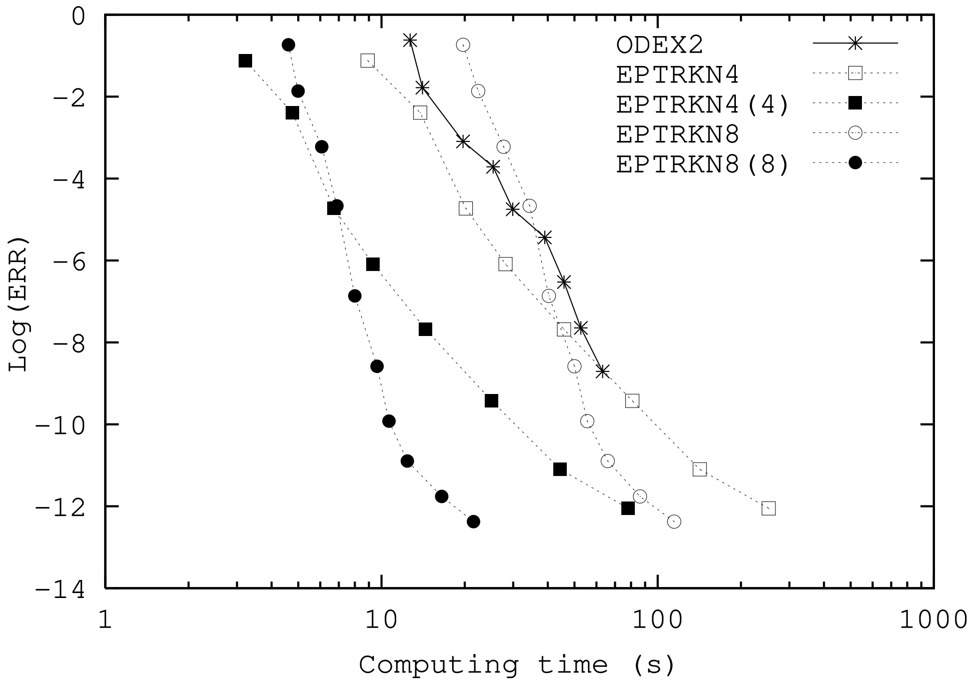

Figure 3: Results for **PLEI**

The parallel speedup (cf. e.g., [11]) with $ATOL = RTOL = 10^{-8}$ as shown in Fig. 6, is very problem dependent. Using the optimal number of processors, the best speedup obtained by the codes EPTRKN4 and EPTRKN8 for the problem **MOON** is approximately 3.3 and 5.5, respectively. We can also see from the results in Fig. 3 - 5 that for more stringent tolerances, a better speedup can be achieved.

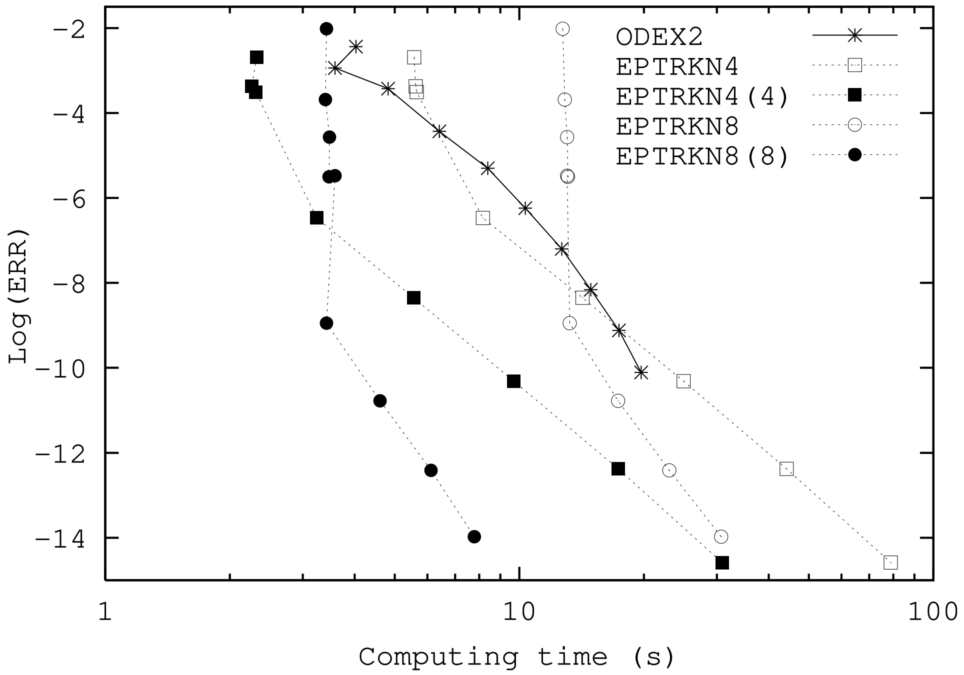

Figure 4: Results for **WAVE**

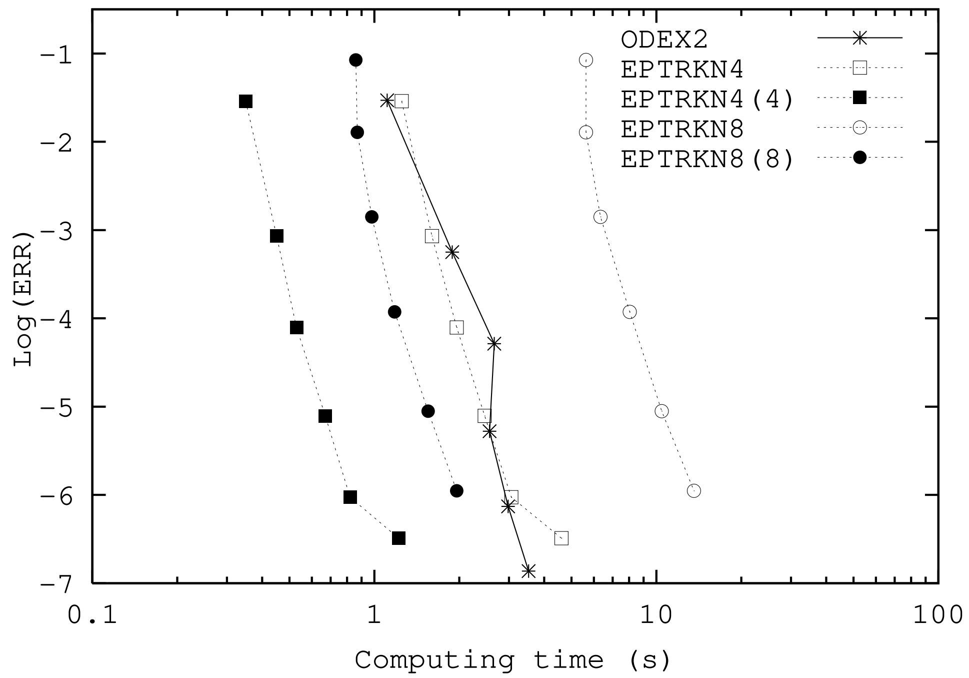

Figure 5: Results for **MOON**

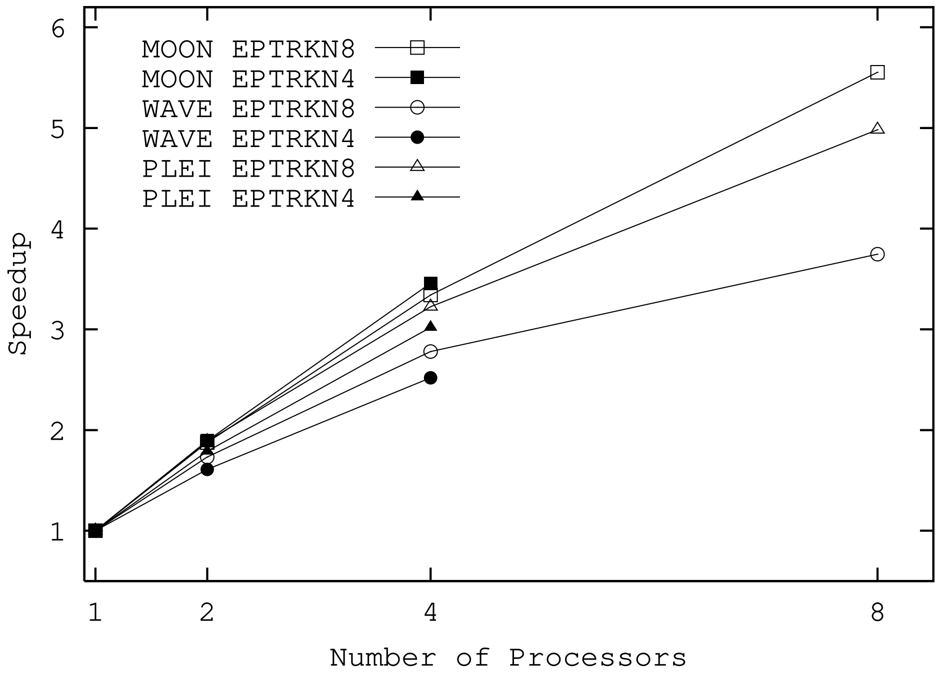

Figure 6: Parallel speedup

4 Concluding remarks

In this paper we have considered the performance of a class of parallel explicit pseudo two-step RKN methods (EPTRKN methods) by comparing two new codes from this class EPTRKN4 and EPTRKN8 with the highly efficient sequential code ODEX2. By using nonstiff, expensive problems and by implementing these codes on a shared memory computer, we have shown the superiority of the new parallel codes over ODEX2. In the future we shall further improve these new parallel codes by some optimal choices of method parameters.

References

[1] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations*, (Clarendon Press, Oxford, 1995).

[2] N.H. Cong, *An improvement for parallel-iterated Runge-Kutta-Nyström methods*, Acta Math. Viet. **18** (1993), 295-308.

[3] N.H. Cong, *Note on the performance of direct and indirect Runge-Kutta-Nyström methods*, J. Comput. Appl. Math. **45** (1993), 347-355.

[4] N.H. Cong, *Direct collocation-based two-step Runge-Kutta-Nyström methods*, SEA Bull. Math. **19** (1995), 49-58.

[5] N.H. Cong, *Explicit symmetric Runge-Kutta-Nyström methods for parallel computers*, Comput. Math. Appl. **31** (1996), 111-122.

[6] N.H. Cong, *Explicit parallel two-step Runge-Kutta-Nyström methods*, Comput. Math. Appl. **32** (1996), 119-130.

[7] N.H. Cong, *Explicit pseudo two-step Runge-Kutta methods for parallel computers*, Intern. J. Comput. Math.**73** (1999), 77-91.

[8] N.H. Cong, *Continuous variable stepsize explicit pseudo two-step RK methods*, J. Comput. Appl. Math. **101** (1999), 105-116.

[9] N.H. Cong, *Explicit pseudo two-step RKN methods with stepsize control*, to appear in Appl. Numer. Math.

[10] N.H. Cong and N.T. Hong Minh, *Parallel block PC methods with RKN-type correctors and Adams-type predictors*, Intern. J. Comput. Math.**74** (2000), 509-527.

[11] N.H. Cong, K. Strehmel, R. Weiner and H. Podhaisky, *Runge-Kutta-Nyström-type parallel block predictor-corrector methods*, Adv. Comput. Math. **38** (1999), 17-30.

[12] N.H. Cong, K. Strehmel and R. Weiner, *A general class of explicit pseudo two-step RKN methods on parallel computers*, Comput. Math. Appl. **38** (1999), 17-30.

[13] N.H. Cong, H. Podhaisky and R. Weiner, *Numerical experiments with some explicit pseudo two-step RK methods on a shared memory computer*, Comput. Math. Appl. **36** (1998), 107-116.

[14] E. Fehlberg, *Klassische Runge-Kutta-Nyström-Formeln mit Schrittweitenkontrolle für Differentialgleichungen* $x'' = f(t,x)$, Computing **10** (1972), 305-315.

[15] E. Fehlberg, *Eine Runge-Kutta-Nyström-Formel 9-ter Ordnung mit Schrittweitenkontrolle für Differentialgleichungen* $x'' = f(t,x)$, Z. Angew. Math. Mech. **61** (1981), 477-485.

[16] E. Fehlberg, S. Filippi und J. Gräf, *Ein Runge-Kutta-Nyström-Formelpaar der Ordnung 10(11) für Differentialgleichungen* $y'' = f(t,y)$, Z. Angew. Math. Mech. **66** (1986), 265-270.

[17] S. Filippi und J. Gräf, *Ein Runge-Kutta-Nyström-Formelpaar der Ordnung 11(12) für Differentialgleichungen der Form* $y'' = f(t,y)$, Computing **34** (1985), 271-282.

[18] S. Filippi and J. Gräf, *New Runge-Kutta-Nyström formula-pairs of order 8(7), 9(8), 10(9) and 11(10) for differential equations of the form* $y'' = f(t,y)$, J. Comput. Appl. Math. **14** (1986), 361-370.

[19] E. Hairer, *Méthodes de Nyström pour l'équations différentielles* $y''(t) = f(t,y)$, Numer. Math. **27** (1977), 283-300.

[20] E. Hairer, *Unconditionally stable methods for second order differential equations*, Numer. Math. **32** (1979), 373-379.

[21] E. Hairer, *A one-step method of order 10 for* $y''(t) = f(t,y)$, IMA J. Numer. Anal. **2** (1982), 83-94.

[22] E. Hairer, S.P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations, I. Nonstiff Problems*, (second revised edition, Springer-Verlag, Berlin, 1993).

[23] P.J. van der Houwen and B.P. Sommeijer, *Explicit Runge-Kutta- (Nyström) methods with reduced phase error for computing oscillating solutions*, SIAM J. Numer. Anal. **24** (1987), 595-617.

[24] P.J. van der Houwen, B.P. Sommeijer and N.H. Cong, *Stability of collocation-based Runge-Kutta-Nyström methods*, BIT **31** (1991), 469-481.

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

[26] E.J. Nyström, *Über die numerische Integration von Differentialgleichungen*, Acta Soc. Sci. Fenn. **50** (13) (1925), 1-54.

[27] H. Podhaisky, R. Weiner and J. Wensch, *High order explicit two-step Runge-Kutta methods for parallel computers*, CIT **8** (2000), 13-18.

[28] L.F. Shampine and M.K. Gordon, *Computer Solution of Ordinary Differential Equations, The Initial Value Problems*, (W.H. Freeman and Company, San Francisco, 1975).

[29] B.P. Sommeijer, *Explicit, high-order Runge-Kutta-Nyström methods for parallel computers*, Appl. Numer. Math. **13** (1993), 221-240.