ELSEVIER

Parallel Computing 20 (1994) 275–295

PARALLEL COMPUTING

Parallel multiple shooting for the solution of initial value problems [†]

M. Kiehl *

Mathematisches Institut, Technische Universität München, Arcisstr. 21, 80290 München, Germany

(Received 6 March 1993; revised 22 October 1993; 13 November 1993)

Abstract

The computing time for the numerical solution of initial-value problems is closely related to the number of evaluations of the right-hand side. In general this number can only be reduced slightly on parallel computers, even if simultaneous evaluations of right-hand side are counted as one evaluation.

For special problems, however, it is possible to construct special methods which show a remarkable speedup on parallel computers. Multiple shooting, a method for boundary-value problems with an inherent parallelism, can also be applied efficiently to linear initial-value problems and to non-linear initial-value problems if good approximations are available.

Key words: Multiple shooting; Parallel computation; Ordinary differential equations; Linear differential equations; Stiff differential equations

1. Introduction

The solution of an initial-value problem (IVP)

$$y'(x) = f(x, y), \quad y(a) = y_a, \quad a \le x \le b$$

is very time-consuming in some applications, and many efforts have been made to reduce the computing time by using parallel computers. Throughout this paper we assume that only the number of evaluations of f determines the computing time.

[†] Supported by the Bavarian Consortium of High Performance Scientific Computing (FORTWIHR).
* Email: Kiehl@mathematik.tu-muenchen.de

1.1. Classical parallel methods

Several strategies have been developed to benefit from parallelism.

(a) *Parallelism in the problem:* The easiest way to parallelize the evaluations of f is to evaluate all components of

$$y'(x) = f(x, y) = (y_1', y_2', \dots, y_n')^{\mathrm{T}} = (f_1(x, y), f_2(x, y), \dots, f_n(x, y))^{\mathrm{T}}$$

in a parallel way. Each f_i can be evaluated independently from the others. If the dimension n of the problem is much larger than the number r of processors; a speedup of nearly a factor r is possible. It is the best strategy for large scale problems.

However, if the evaluation of f does not require many operations, the components of f should not be computed in a parallel way as communication would become dominant.

(b) *Parallel iteration methods:* By implicit Runge-Kutta methods a system of implicit equations

$$AK := A\begin{pmatrix} k_1 \\ \vdots \\ k_s \end{pmatrix} = hB\begin{pmatrix} f(k_1) \\ \vdots \\ f(k_s) \end{pmatrix} =: hBf(K) \quad A, B \in \mathbb{R}^{ns \times ns} \quad k_i, f(k_i) \in \mathbb{R}^n \tag{1.1}$$

is solved iteratively.

If A and B are lower triangle matrices with a constant diagonal we have SDIRK methods, which reduce the amount of work for solving the linear equation in each iteration. If A and B have no structure, then all k_i are iterated simultaneously. The linear algebra becomes more important, but the residuum of (1.1) can be computed in a parallel way.

Integration methods are constructed so that (1.1) is especially well suited to apply parallel iteration methods for non-linear equations. The methods are constructed for the use of only a few processors and can therefore only lead to limited speedup [10,11].

(c) *Block methods:* The theory of classical one-step methods can be generalised. Evaluations of f with different, independent arguments count for one evaluation if they are performed in parallel (parallel stages, block). This results, for example, in parallel Runge-Kutta methods [12]. However, the number of parallel stages cannot considerably be reduced compared to sequential stages [16] and therefore the speedup is limited.

Very good speedups are reported for extrapolation methods which can also be regarded as Runge-Kutta methods. They offer the additional advantage to perform tasks in parallel which consist of more than one evaluation of f and thereby reduce the communication overhead.

Until now the speedup factor obtained from other block methods is limited (< 5) even if many processors are available. But extrapolation methods already

allow an efficient use of up to 5 processors and more [16,20]. Block methods – especially extrapolation methods – can therefore be used if f does not require many operations to evaluate and if only a few processors are available.
(d) *Frontal methods:* A different approach is an extension of multi-step methods which use a prediction formula P and a correction formula C after each evaluation E of f (*PECE*).

If r processors are used, the approximation of the solution can be iterated simultaneously at r point. We define

$$\eta_{n,r} := \left(\eta_1^{(r)}, \dots, \eta_{n-r}^{(r)}, \eta_{n-r+1}^{(r-1)}, \dots, \eta_n^{(0)}\right)$$

where $\eta_k^{(j)}$ is the approximation of $y(t_k)$ after j corrections of $\eta_k^{(0)}$. $\eta_{n-r}^{(r)}$ is accurate enough (after r correction steps). The iteration is of the form

$$\begin{aligned}\eta_{n+1,r} &= \left(\eta_1^{(r)}, \dots, \eta_{n-r+1}^{(r)}, \dots, \eta_n^{(1)}, \eta_{n+1}^{(0)}\right) := \Phi(\eta_{n,r}, f(\eta_{n,r}))\\ &:= \left(\eta_1^{(r)}, \dots, \eta_{n-r}^{(r)}, C_r(\eta_{n,r}, f(\eta_{n,r})), \dots, C_1(\eta_{n,r}, f(\eta_{n,r})),\right.\\ &\quad \left. P(\eta_{n,r}, f(\eta_{n,r}))\right)\end{aligned}$$

where both the correctors C_i and the predictor P use

$$\eta_{n,r} \quad \text{and} \quad f(\eta_{n,r}) := \left(f(\eta_{n-r}^{(r)}), f(\eta_{n-r+1}^{(r-1)}), \dots, f(\eta_n^{(0)})\right).$$

The evaluation of $f(\eta_{n,r})$ and the computation of the C_i's and P can be done in a parallel way.

The advantage of these so-called waveform relaxation methods is that arbitrarily many processors can be used. For $r = 1$ we have an ordinary PECE-multi-step method. In this case the prediction $\eta_{n+1}^{(0)}$ is a good approximation of $y(t_{n+1})$ as it is based only on good approximations of the solution at the previous time points. For larger r the computation is less efficient.

It is a further extension to allow the prediction of more than one η_k in each iteration if $\eta_{n-r+1}^{(r)}$ but also $\eta_{n-r+2}^{(r-1)}$ etc. is accurate enough [3].

The speedup of frontal methods is not a priori limited, but the efficiency can become very low. These methods are recommended if many processors are available and if the evaluation of f requires only a few operations so that (a) does not apply.

1.2. Multiple shooting for initial value problems

The method proposed here belongs to the frontal methods. We consider IVPs where f is of low dimension and requires not many operations to evaluate. We further assume that the computing time is very large so that reducing it becomes very important.

The method presented shows a speedup factor much greater than 10, provided that sufficiently many processors are available.

The basic idea is to formulate the IVP as a boundary-value problem (BVP) where all the boundary conditions are placed at the left boundary, and to apply the multiple shooting method (MS). MS is an iterative method to solve BVPs by the repeated solution of IVPs. Therefore it is generally not very useful to solve IVPs by MS rather than by direct methods in sequential mode.

On the other hand MS has an inherent parallelism. This parallelism has already been widely discussed e.g. in [2,14,19,22 and 23]. But for the case of IVPs, it is still doubtfull, whether the speedup gained by parallelization is sufficient to qualify MS as a competitive method. Therefore parallel MS was only used to solve BVPs in the papers above.

In [15] MS was used for the solution of a restricted class of IVPs with numerical instability where forward integration is not possible. The stability was achieved by backward shooting and this was speeded up by parallel shooting.

Here parallel MS is used for problems where forward integration is possible and parallel MS has to compete with a large number of direct integration methods.

1.3. Contents

The paper is organized as follows:
In Section 2 parallelisation aspects of multiple shooting are discussed.
In Section 3 it is shown that multiple shooting is well suited for linear problems.
In Section 4 the convergence in case of non-linear IVPs is discussed.
In Section 5 it is shown how stiffness can affect the speedup.
In Section 6 an application is given where parallel MS leads to a considerable speedup.

2. Multiple-shooting

The multiple-shooting method (MS) is a very successful method for the numerical solution of boundary-value problems.

$$y'(t) = f(t, y) \tag{2.1}$$

subject to the boundary conditions

$$r(y(a), y(b)) = 0 \in \mathbb{R}^n. \tag{2.2}$$

The idea is to start with some approximations η_k of the solution $y()$ at points $t = t_k$ with

$$a = t_1 \le t_2 \le \cdots t_{m+1} = b.$$

Then a series of IVPs is solved,

$$y'(t) = f(t, y), \quad y(t_k) = \eta_k.$$

Let $y(t, t_a, y_a)$ denote the solution of the initial-value problem (IVP)

$$y' = f(t, y), \quad y(t_a) = y_a,$$

at the point t, and define

$$d_k := y(t_{k+1}, t_k, \eta_k) - \eta_{k+1} \quad \text{for} \quad k = 1, \dots, m, \tag{2.3}$$

Then the problem (2.1), (2.2) has a solution if and only if

$$\mathscr{F}(\eta_1,\dots,\eta_{m+1}) := (d_1,\dots,d_m, r) = 0 \in \mathbb{R}^{(n+1)m}. \tag{2.4}$$

In the Fortran subroutine BNDSCO [18], an implementation of the MS algorithm [21, 5], Eq. (2.4) is solved by a modified Newton method.

The evaluation of $d_1,\dots,d_m$ in (2.4) by (2.3) requires the solution of m IVPs. They are independent from each other and yield m parallel tasks with the complexity of one IVP over an interval of length $(b-a)/m$ on the average. This can be done naturally in a parallel way.

Using Newton's method the Jacobian $\mathscr{J}$ of $\mathscr{F}$ is needed. The Jacobian matrix is sparse and has a special structure. Therefore the $n(m+1)$ linear equations

$$\mathscr{J}\Delta\eta := \begin{pmatrix} G_1 & -I & & & \\ & G_2 & -I & & \\ & & \ddots & \ddots & \\ & & & G_m & -I \\ r_{ya} & & & & r_{yb} \end{pmatrix} \begin{pmatrix} \Delta\eta_1 \\ \Delta\eta_2 \\ \vdots \\ \Delta\eta_m \\ \Delta\eta_{m+1} \end{pmatrix} = -\begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \\ r \end{pmatrix} \tag{2.5}$$

for the Newton correction can be solved very efficiently. Moreover, Eq. (2.5) can be condensed by a blockwise Gauß elimination to $m+1$ systems of linear equations of order n [21]. This must be kept in mind when increasing m in order to increase the parallelism. It should be mentioned that the condition of the system limits the reduction of the dimension [7] but this is dependent on the problem and not on the number m of nodes. In any way the number of equations is always rather small, which is in contrast to collocation methods where the amount of computations for the linear algebra cannot be neglected.

Most of the computing time is spent on computing the matrices G_k which are defined by

$$G_k := \partial d_k/\partial\eta_k = \partial y(t_{k+1}, t_k, \eta_k)/\partial\eta_k. \tag{2.6}$$

These so-called transition matrices are the solutions of the matrix differential equations

$$G_k'(t) = f_y(t, y(t, t_k, \eta_k))G_k(t). \quad G_k(t_k) = I, \quad k = 1,\dots,m. \tag{2.7}$$

The columns of each matrix $G_k = \partial d_k/\partial\eta_k$ can be approximated by a difference approximation,

$$\frac{\partial d_k}{\partial\eta_{k,j}} \doteq \frac{y(t_{k+1}, t_k, \eta_k + \epsilon\cdot e_j) - y(t_{k+1}, t_k, \eta_k)}{\epsilon}, \quad j = 1,\dots,n,$$
$$k = 1,\dots,m, \tag{2.8}$$

where ϵ is sufficiently small and e_j is the jth unit vector.

Obviously these $n\cdot m$ additional IVPs in (2.8) can also be solved in a parallel way if enough processors are available. With $m\cdot(n+1)$ processors, (2.4) and (2.8) can be computed in the time needed to compute $y(t_2, t_1, y_1)$.

By choosing m large we obtain a parallelism of m in (2.4) respectively $n \cdot m$ in (2.8). The number m is limited only by the number $n_{\mathscr{F}}$ of basic integration steps needed for the evaluation of $\mathscr{F}$. If the problem requires small stepsizes or if the interval $[a, b]$ is large enough we can expect a high degree of parallelism which is only restricted by the number of available processors. Each parallel task is computing the solution of an IVP across a subinterval, which generally is still very time-consuming. For communication just the vectors $G_k \Delta\eta_k$ and $y(t_{k+1}, t_k, \eta_k)$ have to be sent from processor k to processor $k+1$ for all k after each iteration, independent of the length of the subintervals. Therefore communication overhead can be neglected even on distributed memory computers if the subintervals are long enough.

This justifies that all numerical tests were done on a sequential computer by simulating a multiprocessor system.

3. Linear problems

3.1. Super convergence for linear problems

In the case of a linear differential equation

$$y'(t) = A(t)y(t), \quad y(a) = y_a, \tag{3.1}$$

the exact solution at time t_k is a linear function of the initial value y_a and is given by

$$y(t_k, a, y_a) = G_k G_{k-1} \cdots G_2 G_1 y_a. \tag{3.2}$$

$f_y(t, y(t, t_k, \eta_k)) = A(t)$ is independent of η_k and so is G_k defined by (2.7). Eq. (2.5) becomes a system of linear equations with constant coefficients, and the Newton method converges after one iteration step independently of the initial guess $\eta_k^{(0)}$.

As the initial value uniquely defines $y(t_2) = y(t_2, t_1, y_a)$, G_1 is not needed to correct $\eta_2^{(1)}$ and the first interval requires only one processor. If we use (2.8) the following intervals require $n+1$ processors each to compute $y(t_{k+1}, t_k, \eta_k)$ and $G_k(t_{k+1})$ simultaneously. Therefore

$$r = 1 + (m-1)(n+1) \tag{3.3}$$

processors are necessary to achieve a speedup factor of

$$m = (r-1)/(n+1) + 1. \tag{3.4}$$

This speedup, however, can only be achieved under the following assumptions:

- The amount of computations for the matrix multiplications $G_m G_{m-1} \cdots G_2 G_1$ in (3.2) is negligible compared to the evaluations of f.
- The interval is divided into subintervals in such a way that the integration in each subinterval requires nearly the same computing time.

- The subintervals do not restrict the length of the stepsize.
- The stepsize does not vary a lot if the initial values are changed.

It is obvious that for most of the interesting applications at least one of the assumptions is violated. But nevertheless a good speedup can be expected.

If s_k steps are needed to compute $y(t_{k+1}, t_k, \eta_k)$ and if $s \approx \Sigma s_k$ steps are needed to compute $y(b, a, y(a))$ sequentially, then

$$B := s/(m \max\{s_k\}) \leq 1$$

measures the load balance if each $y(t_{k+1}, t_k, \eta_k)$ is computed on another processor. While sequential MS is not the alternative in this context, it should be mentioned however, that B also approximates the efficiency E of parallel MS with m processors compared to sequential MS.

If only $i = 1$ Newton iterations are necessary, the speedup S is restricted by

$$S \leq S_1 := m \frac{s}{m \max\{s_k\}} \leq \left(\frac{r-1}{n+1} + 1\right) B \leq \frac{r+n}{n+1} B.$$

This shows that (2.8) is not very useful in the interesting case of large n.

If we compute the G_k by solving the matrix differential Eq. (2.7) and if $[f]$ respectively $[f, G_k']$ denote the number of operations to evaluate f respectively f and the right side of (2.7), then we can choose $m = r$ and S is restricted by

$$S \leq S_2 := r \frac{[f]}{[f, G_k']} B.$$

3.2. Example

We consider a system of the form (3.1) with the coefficient matrix

$$A = (a_{i,j})_{i,j=1}^{n}, \quad a_{ij} = \frac{\cos(it)}{(2n+i-j)} + \sin^j(t)$$

with $n = 10$ and $[a, b] = [0, 10]$. The required accuracy was chosen to be 10^{-8}.

This non-stiff problem can be solved by the explicit integration routine DOPRI8 (see [8]). All the assumptions of Section 3.1 are fulfilled. An equidistant partitioning of the interval leads to a reasonable load balancing B.

Because of Eq. (3.4), only $m = 94$ intervals can be computed simultaneously with $r = 1024$ processors (c.f. (3.3)), if G_k is computed via (2.8).

The method converges after $i = 1$ iterations and we obtain

$$B = \frac{s}{m \max\{s_k\}} = \frac{170}{94 \cdot 3} \approx 0.6.$$

Speedup S and efficiency E for this low dimensional problem are

$$S = 170/(i \max\{s_k\}) = 170/3 \approx 56.6; \quad E \approx 0.055.$$

S_1 rapidly decreases if the dimension n increases. Therefore (2.8) is not useful.

In the example $f = Ay$ requires about $[f] = 10n^2 + 2n^2 = 1200$ operations, where $10n^2 = 1000$ operation are necessary to compute the n^2 elements of matrix $A = f_y$ and $2n^2 = 200$ operations for the matrix-vector multiplication.

G'_k and f require only $[f,\ G'_k] = 10n^2 + 2n^2 + 2n^3 = 3200$ operations. Thus computing G_k by (2.7) is preferable and the efficiency respectively the speedup with r processors become

$$E_2 \approx \frac{[f]}{[f,\ G'_k]} B \approx 0.225; \quad S_2 = rE_2,$$

as long as $r \ll s = 170$.

Until now we have neglected the computing time for the matrix multiplications in (3.2) $G_k G_{k-1} \cdots G_2 G_1$ for $k = 1, \ldots, m$. All the products can be computed in $\log_2(m)$ steps on $m/2$ processors where each step consists of one matrix multiplication per processor (This can be proved by induction).

In the example given above, we need $\lceil \log_2 170 \rceil = 8$ steps with 2000 operations on each processor. The overhead thus corresponds to 16000 operations which is equivalent to about 13.3 evaluations of f or about one basic integration step of DOPRI8, which requires 13 evaluations of f in each unit step. With $r = 94$ and G_k computed via (2.7), S and E become

$$S \approx s/(\max\, s_k + 1) \approx 170/(3+1) = 42.5; \quad E \approx 0.45.$$

Note that the example is not stiff. We will see in Section 5 that stiffness severely reduces the speedup.

4. Non-linear problems

4.1. *A modified relaxation strategy*

The modified Newton method which is used in MS leads to a special convergence behaviour if IVPs are solved. In this case the approximations at different nodes converge with different speed. The convergence can appropriately be described by using a criterion which distinguishes the values η_k. We define the residual vector

$$\mathscr{R} := (r_1, \ldots, r_m) := (\|d_1\|, \ldots, \|d_m\|).$$

With (2.5) and $d_1^{(0)} := y(t_2,\ t_1,\ \eta_1^{(0)}) - \eta_2^{(0)}$ one obtains $\Delta\eta_2^{(0)} := d_1^{(0)}$. A correction

$$\eta^{(1)} = \eta^{(0)} + \lambda \Delta\eta^{(0)} \tag{4.1}$$

with $\lambda = 1$ then yields $\eta_2^{(1)} = \eta_2^{(0)} + d_1^{(0)} = y(t_2,\ t_1,\ y_a) = y(t_2)$.

Hence the exact solution proceeds at least one subinterval by each iteration, so that $\eta_k^{(i)} = y(t_k,\ t_1,\ y_a)$ for all $i \geq k - 1$. Thus the whole problem is solved after at least m iterations. Obviously this does not hold for the modified Newton method with a relaxation factor $\lambda < 1$ in (4.1).

For a desired accuracy *acc* we find after i iterations that $r_k < acc$ for $k = 1,\dots,l(i)$. We then consider η_k for $k = 1,\dots,l(i)$ to be accurate. For $k > l(i)$, we compute the convergence factors

$$c_{k+1}^{(i)} := r_k^{(i)}/r_k^{(i-1)} = \| d_k^{(i)} \| / \| d_k^{(i-1)} \|$$

which may be larger than 1 for some indices k. Then the relaxation factor λ is chosen individually for each k. We define

$$\bar{\lambda}_k^{(i)} := \max\{0.5\lambda_k^{(i-1)},\ \lambda_{\min}\} \text{ if } c_k^{(i)} > 1$$

$$\bar{\lambda}_k^{(i)} := \min\{2\lambda_k^{(i-1)},\ 1\} \text{ if } c_k^{(i)} < 1.$$

which corresponds to the usual relaxation formulas. Then we define

$$\lambda_k^{(i)} := \min_{l(i)+1 \le l \le k} \{\bar{\lambda}_l^{(i)}\}$$

which leads to a decreasing sequence $\lambda_k^{(i)}$ and guarantees that poor convergence of $\eta_k^{(i)}$ does not have any influence on the convergence of $\eta_l^{(i)}$ for $l < k$.

The unrelaxed correction vector $(\Delta\eta_1^{(i)}, \Delta\eta_2^{(i)},\dots,\Delta\eta_m^{(i)})^{\mathrm{T}}$ thereby is not only changed in its size but also in its direction. However, the new correction still has a positive scalar product with the unrelaxed correction

$$\left(\lambda_1^{(i)}\Delta\eta_1^{(i)}, \lambda_2^{(i)}\Delta\eta_2^{(i)},\dots,\lambda_m^{(i)}\Delta\eta_m^{(i)}\right)\left(\Delta\eta_1^{(i)}, \Delta\eta_2^{(i)},\dots,\Delta\eta_m^{(i)}\right)^{\mathrm{T}}$$
$$= \sum_{k=1}^{m} \lambda_k^{(i)} \| \Delta\eta_k^{(i)} \|^2 > 0.$$

Note: If enough processors are available, we can perform corrections with different $\lambda_k^{(i)}$ in parallel. For example $\lambda_k^{(i)} \in \{\lambda_k^{(i-1)}, 2^{\pm 1}\lambda_k^{(i-1)}, 2^{\pm 2}\lambda_k^{(i-1)},\dots\}$.

In case of few processors we correct only those η_k with $\lambda_k^{(i)} = 1$. The other node values remain uncorrected or are changed by a different strategy. They can for example be extrapolated from the last node values that show good convergence. High order extrapolation thereby can easily lead to overflow in case of large subintervals. In the following algorithm order 0 extrapolation was used.

4.2. Algorithm

If the interval $[a, b]$ is large, we use the following algorithm

```
1. Choose $a = t_1 < t_2 < \cdots < t_{m+1} \ll b$
   and $\eta_k^{(0)} = y_a$ for $k = 1,\dots,m+1$.
   Set first index $l_1 = 1$; last index $l_2 := m + l_1$; iteration number $i := -1$
2. $i := i + 1$; For $k = l_1,\dots,l_2 - 1$ compute $y(t_{k+1}, t_k, \eta_k^{(i)})$, $r_k^{(i)}$;
3. Convergence monitor: Compute $l_{acc} := \max_c\{r_k^{(i)} < acc$ for $k = l_1,\dots,c-1\}$
   Then all node values $\eta_j^{(i)}$; $j = l_1 + 1,\dots,l_{acc}$ are accurate.
   $\eta_{lacc+1} := y(t_{lacc+1}, t_{lacc}, \eta_{lacc}^{(i)})$ is also accurate.
   Compute $l_{conv} := \max_c\{c_k^{(i)} \le 1$ for $k = l_1,\dots,c\}$.
   Then only node values $\eta_j^{(i)}$ with $j \le l_{conv}$ show good convergence.
   $l_2 := l_{conv}$; $l_1 := l_{acc}$; $l_{new} := m - (l_{conv} - l_{acc})$;
```

4. If $t_{l_1} = b$: Stop
   Else: Create l_{new} new node values.
   Choose c_1 and c_2 with $c_1 + c_2 = l_{new}$
   Split the longest intervals with large $s_k \gg \Sigma s_j/m$ by introducing c_1 intermediate nodes $t_{l_2+j} \in]t_{l_1}, t_{l_2}[j = 1, \ldots, c_1$.
   Approximate the node values by linear interpolation.
   Choose c_2 new nodes equidistant with interval length $h_{l_2-1} = t_{l_2} - t_{l_2-1}$.

$$t_{l_2+c_1+j} := \begin{cases} t_{l_2} + js_{l_2-1}h_{l_2-1} & \text{if } t_{l_2} + js_{l_2-1}h_{l_2-1} < b \\ t_{l_2} + j(b - t_{l_2})/c_2 & \text{if } t_{l_2} + js_{l_2-1}h_{l_2-1} \geq b \end{cases} \quad j = 1, \ldots, c_2$$

   Approximate η_{l_2+j} by constant extrapolation $\eta_{l_2+j} = \eta_{l_2}$ for $j = 1, \ldots, c_2$
5. Reorder the indices of the t_j so that $t_i < t_j$ if $i < j$; $l_2 = l_1 + m$; Goto 2.)

Thereby the only critical point is the choice of c_1 and c_2.

Large c_2 leads to long intervals and reduces communication overhead. However the convergence behaviour becomes poor as $\eta_{l_2}^{(i)}$ is a poor approximation of $y(t_{l_2+j})$ for large intervals.

Large c_1 leads to an optimal load balance and fast convergence but small intervals so that the overhead can become dominant.

The sequence $s_1, s_2, \ldots$ allows a very good approximation of the speed-up by splitting the subintervals with the largest s_k. And if we assume that $c_k^{(i)}$ is a restriction of a smooth function $c^{(i)}$: $\mathbb{R} \to \mathbb{R}$ on $\mathbb{N}$ with $c_k^{(i)} = c^{(i)}(k)$, then we can use extrapolation to estimate the convergence behaviour of η_{l+c_1+j}. This allows a dynamical choice of c_1 and c_2 dependent on the problem.

If we can neglect the communication we always split intervals with $s_k > 1$ (see Sections 4.3 and 5). Then the algorithm becomes quite similar to that one of Bellen et al. [3]. If the stepsizes do not vary a lot and convergence is not critical we choose $c_2 = l_{new}$ and $c_1 = 0$ (see Section 6).

4.3. Example

It is clear that Algorithm 4.2 leads to good results for linear problems and also for problems which have only small non-linearities. Therefore we now consider a problem with only non-linear terms. Besides that the example is non-stiff and not sensitive.

The regarded differential equation describes the solar system. The planet Mercury is assumed to be a part of the Sun, and the planet Pluto is neglected. Then we have an eight body system of Sun, Venus, Earth, Mars, Jupiter, Saturn, Uranus and Neptune. The position of each of the astronomical bodies has 3 coordinates x, y and z. The movement is governed by the second order differential equations

$$\ddot{x}_i = \sum_{j=1, j\neq i}^{8} \mu_j \frac{x_j - x_i}{r_{ij}^3}, \quad \ddot{y}_i = \sum_{j=1, j\neq i}^{8} \mu_j \frac{y_j - y_i}{r_{ij}^3}, \quad \ddot{z}_i = \sum_{j=1, j\neq i}^{8} \mu_j \frac{z_j - z_i}{r_{ij}^3},$$

where the distances r_{ij} are given by

$$r_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

and μ_j is the normalized mass of the j-th body. This system is transformed into a first order differential equation system of dimension $n = 48$. For 1024 processors an optimal number of subintervals by (3.4) therefore is $m = 22$.

We use an appropriate code like ODEX2 [8] especially designed to solve second order IVPs. If we choose any position of the planets of this century as initial data then the problem is not critical and the stepsizes are nearly constant (approximately 40 days). An equidistant interval partitioning need not be refined or corrected. But each subinterval should be a multiple of 40 days. The total interval thus should be at least $40m = 880$ days.

As no information is available except for the initial data y_a the only reasonable choice of the initial approximation is $\eta_k^{(0)} = y_a$ for all $k = 1,\dots, m+1$. This is a poor approximation of the exact position of the planets. Consequently the convergence is expected to be slow.

But in fact there is no speed-up at all. The relaxation factors become $\lambda_k^{(i)} < 1$ for all $i < k - 1$ and the convergence is as slow as possible ($l(i) = i + 1$).

The reason for this result is expressed in the following lemma.

4.4. Convergence for non-linear problems

Lemma. *Convergence of the unrelaxed Newton method for problem* (2.4) *with* ($n = 1$). *Let* $|\partial G(t_{k+1}, t_k, \eta_k)/\partial \eta_k| > c > 0$ $\forall k$, η_k, *and let* $|\eta_2^{(0)} - y(t_2)| > \max\{2/c, acc\}$. *Then* $|\eta_k^{(i)} - y(t_k)| > acc$ *for* $i < k - 1$. ($\Rightarrow \eta_k^{(i)}$ *converges exactly for* $i = k - 1$.)

Proof. The correction equation

$$\Delta\eta_{k+1}^{(i)} = G_k(t_{k+1}, t_k, \eta_k^{(i)})\Delta\eta_k^{(i)} + y(t_{k+1}, t_k, \eta_k^{(i)}) - \eta_{k+1}^{(i)} \tag{4.2}$$

is used for the correction $\eta_{k+1}^{(i+1)} := \eta_{k+1}^{(i)} + \Delta\eta_{k+1}^{(i)}$. On the other hand we have

$$\begin{aligned} y(t_{k+1}) &= y(t_{k+1}, t_k, y(t_k)) = y(t_{k+1}, t_k, \eta_k^{(i)}) \\ &\quad + G_k(t_{k+1}, t_k, \eta_k^{(i)})(y(t_k) - \eta_k^{(i)}) \\ &\quad + \frac{\partial G(t_{k+1}, t_k, \xi)}{\partial \eta_k} \frac{(y(t_k) - \eta_k^{(i)})^2}{2} \end{aligned} \tag{4.3}$$

with ξ between $y(t_k)$ and $\eta_k^{(i)}$. Because $\eta_k^{(i)} = y(t_k)$ after $i \geq k - 1$ iterations we have

$$\eta_{i+2}^{(i+1)} = y(t_{i+2}) \Rightarrow \Delta\eta_{i+2}^{(i)} = y(t_{i+2}) - \eta_{i+2}^{(i)}$$

and finally with $k = i + 2$ in (4.2) and (4.3)

$$\Delta\eta_{i+3}^{(i+1)} = y(t_{i+3}) - \eta_{i+3}^{(i+1)} = \frac{\partial G(t_{i+3}, t_{i+2}, \xi)}{\partial \eta_{i+2}} \frac{(\Delta\eta_{i+2}^{(i)})^2}{2}.$$

Thus $|\Delta\eta_{i+3}^{(i+1)}|$ is an increasing sequence if $|\Delta\eta_2^{(0)}| = |y(t_2) - \eta_2^{(0)}| > 2/c$. Therefore $\eta_k^{(i)}$ cannot converge before $i = k - 1$ iterations. □

If relaxed Newton corrections are used the situation is not significantly better as can be seen from the example $y' = ay^2 + by + c$ with a, b, $c > 0$ and $\eta_k^{(0)} = y_0$. Then $(\eta_k^{(i)})_{i \in \mathbb{N}}$ is an increasing sequence and the relaxed correction leads to slower growth and therefore slower convergence.

Note: (1) $\| \partial G_k(t_{k+1}, t_k, y(t_k))/\partial\eta_k \| = 0$ for linear problems. Thus the convergence problems directly reflect the non-linearity of the differential equation.

(2)

$$\left| \frac{\partial G(t_{k+1}, t_k, y(t_k))}{\partial \eta_k} \right| = \left| \int_{t_k}^{t_{k+1}} f_y(t, y(t)) \, \mathrm{d}t \right| \approx (t_{k+1} - t_k) \, | f_y(t, \xi) |.$$

Thus we can improve the convergence by choosing $(t_{k+1} - t_k)$ small. This may help in case of nearly linear problems. In the given example $(t_{k+1} - t_k)$ was chosen as small as reasonable (one unit step of ODEX2) neglecting the overhead, but still the convergence could not be improved.

The results reflect the evolutionary character of non-linear IVPs. The vector field of the differential equation near a bad approximation of the solution does not allow to get information about the vector field near the solution. We cannot hope to obtain a good approximation of the solution as long as such an information is missing, and we cannot get such an information before we have a good approximation of the solution. Therefore convergence can only be guaranteed in the neighborhood of the initial point, respectively a point where the approximation has already converged.

As we assume that f requires only few operations large intervals have to be computed in parallel. Then the problem must locally behave nearly like a linear problem, or a good approximation of the solution is needed.

5. Stiff problems

We now focus on an interesting class of problems where $|\partial G(t_{k+1}, t_k, \eta_k)/\partial\eta_k|$ is small. Then convergence is not critical as the solution is not critically dependent on y_0.

5.1. Example

We regard problem D3 of [6], a test example for reaction kinetics.

$$\begin{aligned}
y_1' &= y_3 - 100 y_1 y_2 & y_1(0) &= 1 \\
y_2' &= y_3 - 100 y_1 y_2 - 2 \cdot 10^4 y_2^2 + 2 y_4 & y_2(0) &= 1 \\
y_3' &= -y_3 + 100 y_1 y_2 & y_3(0) &= 0 \\
y_4' &= 10^4 y_2^2 - y_4 & y_4(0) &= 0.
\end{aligned}$$

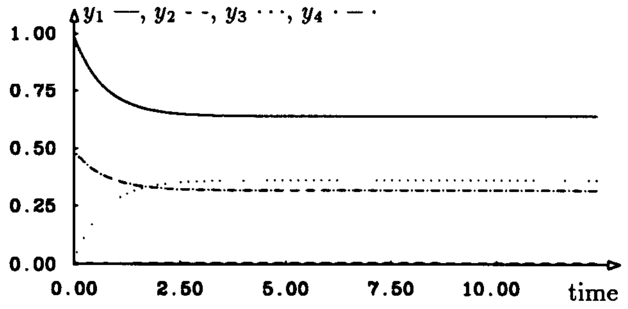

Fig. 1.

The concentrations y_i tend to a limit where $y' = 0$. This limit solution y_l is given by

$$y_l \approx (0.63976,\ 0.00563,\ 0.36024,\ 0.317065).$$

The problem is solved for $t \in [0,20]$ (see Fig. 1). y_l is very soon approached from the initial value in a transient phase, where the stepsizes are very small.

Then a long asymptotic phase follows, where implicit methods can use very large stepsizes.

Because of the damping character of the problem, a correction of η_k does not change η_{k+1} very much, so that η_{k+1} can converge nearly independent of previous η_k.

D3 was solved with $m = 205 \approx 1 + 1023/(n+1)$ equidistant subintervals. The initial approximations were all chosen $\eta_k = y_0$. The implicit method GRK4A [13] was used with $tol = 10^{-6}$. MS converged after $i = 3$ iterations with accuracy $acc < 10^{-4}$.

However, the speedup was only 1.12. The reason is that by sequential integration approximately 50 steps are needed for the first transient phase which lasts until about $t = 0.2$ and about 30 steps for the second transient phase. For the asymptotic phase only about 20 steps are necessary (see Fig. 2).

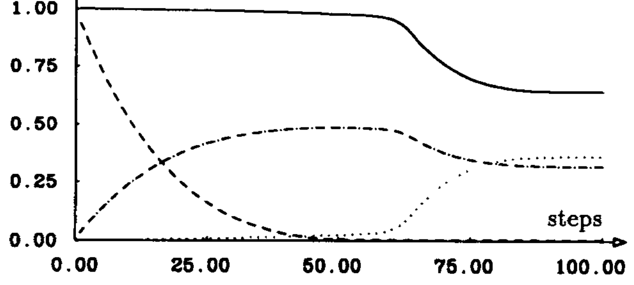

Fig. 2.

Using parallel MS a transient phase has to be computed in each subinterval, which requires small stepsizes so that more than 50 integration steps were necessary in each subinterval. These small stepsizes are needed as we start far away from the solution in each subinterval. The solution over the whole interval in one step therefore cannot be efficient as long as the stepsizes differ so much.

5.2. *Analysis of the stepsize growth*

The stepsize sequence of an implicit IVP solver is known to be increasing during the transient phase of stiff problems. Here we will investigate how the stepsize depends on the initial value. We regard the implicit Euler method and the problem $y' = -ay$. Then the exact solution is $y(t) = e^{-a(t-t_0)}y(t_0)$.

If we use a stepsize h the numerical approximation of the implicit Euler method becomes $\eta(t_0 + h) = 1/(1 + ah)y(t_0)$ and the error is

$$err = \eta(t_0 + h) - y(t_0 + h) = (1/(1 + ah) - e^{-ah})y(t_0).$$

Tolerance *tol* requires

$$tol > err > \left(\frac{1}{1+ah} - e^{-ah}\right)y(t_0) = (ah)^2 y(t_0) + \mathcal{O}(ah)^3,$$

according to the approximation order of the method. (For stiff problems the absolute error should be controlled rather than the relative error.)

If we use methods of order p we generally obtain

$$err = \eta(t_0 + h) - y(t_0 + h) = Kh^{p+1}y(t_0) + \mathcal{O}(ah)^{p+2}.$$

where K is related to $\partial^{p+1}f/\partial y^{p+1} = (-a)^{p+1}$, so that

$$tol > err > \text{const}(ah)^{p+1}y(t_0) + \mathcal{O}(ah)^{p+2}.$$

The tolerance therefore requires

$$ah_0 \approx \text{const}\,\sqrt[p+1]{tol/y_0}\,.$$

Thus we obtain $y_1 = y(t_0 + h_0) = e^{-ah_0}y_0$ and with $\alpha_i := h_{i+1}/h_i$ we find

$$1 < \alpha_0 = \frac{ah_1}{ah_0} \approx \sqrt[p+1]{\frac{tol/y_1}{tol/y_0}} = e^{ah_0/(p+1)} \approx 1$$

and

$$\alpha_1 \approx e^{ah_1/(p+1)} = e^{ah_0\alpha_0/(p+1)} = \alpha_0^{\alpha_0}.$$

If the damping of a stiff component dominates the stepsize control and the problem behaves nearly like a linear system, then α_k can locally be approximated by the recurrency

$$\alpha_{i+1} = \alpha_i^{\alpha_i} \tag{5.1}$$

independent of the order p of the method and the stiffness a of the problem.

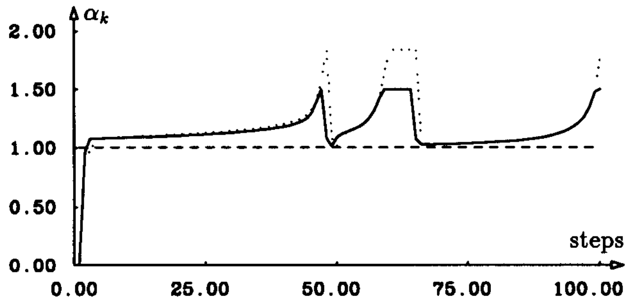

Fig. 3.

For the method GRK4A applied at the D3-problem from STIFF-DETEST the $\alpha_i = h_{i+1}/h_i$ are drawn versus the step number i in Fig. 3.

It shows a fast adaption of the initial stepsize (α small). Then we find a nearly constant $\alpha_3, \ldots, \alpha_{42}$ slightly growing, where the stiff component y_2 is damped. Then a fast adaption of the stepsize ($\alpha_{43}, \ldots, \alpha_{65} \approx \max\ \alpha := 1.5$), then again a nearly constant $\alpha_{66}, \ldots, \alpha_{95}$ slightly growing where the second stiff component is damped, and finally a fast growth of h at the beginning of the stationary phase.

The theoretical value of $\alpha_i^{\alpha_i}$ (dotted line) very well approximates the stepsize factors used by GRK4A (solid line) in the transient phase where the restriction $\alpha < 1.5$ is not active.

We see that the stepsize grow faster than a geometrical sequence. Therefore the transient phase has to be solved first, and the asymptotic phase can be solved afterwards as soon as good approximations for the asymptotic phase are computed. Thereby a clear distinction between transient and asymptotic phase is not possible. In the case $y' = -\lambda y$ each phase is transient compared to the following one and each phase is asymptotic compared to the preceding one. This implies that the interval $[t_{l_1}, t_{l_2}]$ always has to be small. It restricts the possible speedup of MS.

5.3. Numerical results

To achieve subintervals which require approximately the same number of integration steps, we solve D3 by MS with very small subintervals. If we use MS with m subintervals then each subinterval should have at least the length of the optimal initial stepsize ($h_1 = 3.3 \cdot 10^{-6}$ for problem D3). Table 1 shows the number of Newton-iteration steps i, the number of evaluations of $f\ n_{par}$ in each interval by

Table 1
Speedup regarding the transient phase $[0;\ h_1 m]$

m	10	20	30	100	200	300	1000
i	3	4	4	5	6	6	6
n_{par}	7	7	7	7	7	7	7
n_{seq}	63	91	119	196	240	259	322
S	3	3.3	4.3	5.6	5.7	6.2	7.7

parallel computation and in the whole interval by sequential computation n_{seq} and finally the speed-up $S = n_{seq}/(n_{par}i)$.

The table shows that the speedup increases very slowly with increasing m. If we use the algorithm of Section 4.2 with c_1 maximal and $m = 205$ then we find $\lambda_k^{(i)} = 1$ for all k and i. The algorithm needs 20 iterations to find the solution for the whole interval [0,20] and the resulting speedup was about 4.5.

As stiff problems become interesting for parallel computation only if the dimension of the problem becomes very large, it is a strong restriction that we need $m(n+1)$ processors with m large to compute G_k by (2.8) and achieve a remarkable speedup.

We can therefore try to compute the matrices G_k by solving the linear differential Eq. (2.7). In many cases the matrix $f_y(t, y)$ does not require much more operations than f. Then it is possible to compute $G_k(t_{k+1}, t_k, y(t_k))$ in nearly the same time as $y(t_{k+1}, t_k, y(t_k))$ if appropriate algorithms are used [4]. However, the IVP of G_k shows exactly the same stiffness as the original problem. Starting with I as initial value requires to compute the beginning of the transient phase also in the intervals which belong to the asymptotic phase.

Parallel MS therefore cannot be used to solve high dimensional stiff problems very efficiently. This holds also for linear stiff problems.

6. Applications for parallel multiple shooting

6.1. Simplified models

In this section special applications are introduced where parallel methods are necessary and parallel MS can be used efficiently.

Let us assume that the IVP describes a very complicated system and that it is very time-consuming to solve it. Let us further assume that there is a simplified model of the system which can be solved much faster. Typical examples where this is possible are the following:

(1) Problems where the simplified problem can be solved analytically.
(2) Problems which lose stiffness by setting stiff component equal to zero.
(3) Problems which can be decoupled into small problems by ignoring small dependencies.

Let $s(t, t_0, y_0)$ denote the solution of the simplified model at time t with initial condition $y(t_0) = y_0$ with

$$s(t, t_k, \eta_k) \approx y(t, t_k, \eta_k).$$

Then this approximation can serve as initial approximation of the parallel MS method.

After evaluating $\mathcal{F}$ (2.3) by parallel computation we have to compute the correction. Then we can profit from the fact that the simplified model sometimes also allows to approximate G_k by

$$\partial s(t_{k+1}, t_k, \eta_k)/\partial\eta_k \approx \partial y(t_{k+1}, t_k, \eta_k)/\partial\eta_k = G_k.$$

But we do not need the G_k. In fact the matrices G_k are only needed to correct η_{k+1}. In the case of an IVP this correction only depends on the correction of η_k and d_k

$$\Delta\eta_{k+1}^{(i)} = G_k \Delta\eta_k^{(i)} + d_k^{(i)}. \tag{6.1}$$

Instead of that we can also compute

$$\begin{aligned}\Delta\eta_{k+1}^{(i)} &\doteq y\left(t_{k+1}, t_k, \eta_k^{(i+1)}\right) - y\left(t_{k+1}, t_k, \eta_k^{(i)}\right) + d_k^{(i)} \\ &\approx s\left(t_{k+1}, t_k, \eta_k^{(i+1)}\right) - s\left(t_{k+1}, t_k, \eta_k^{(i)}\right) + d_k^{(i)},\end{aligned}$$

which essentially reduces the computational work.

6.2. Example

We return to our solar system of Section 4. It is well known that the motion of two isolated astronomical bodies as a result of the gravity can be computed analytically. The solutions first found by *Kepler* are the so-called *Kepler* ellipses. It is also known, that the astronomical n-body problem cannot be solved analytically for $n > 2$.

In our solar system the dominating body is the Sun. Therefore the motion of each planet can be approximated by regarding only the influence of the Sun, i.e. we approximate the motion of the planets relative to the Sun as solutions of two-body problems, neglecting all the others planets.

Let the masses, the position and the velocity of each planet and the sun at time t be given. These values uniquely define the form of the ellipses, the orientation in space and the exact position of the planet in *Kepler* coordinates relative to the Sun. In order to compute the *Kepler* coordinates for a new time $t_n \neq t$ we have to solve the *Kepler* equation

$$\psi(t_n) = \omega t_n + \epsilon \sin(\psi) = \omega t_n + \sum_{n=1}^{\infty} \frac{2}{n} J_n(n\epsilon) \sin(n\omega t_n),$$

for the eccentric anomaly ψ where ω is the frequency of revolution, ϵ is the eccentricity, and the J_n are the *Bessel* functions (see [1]).

The motion of the Sun itself cannot be approximated by only regarding one of the planets. But it can be computed regarding physical conservation laws, i.e. velocity and torque of the center of gravity of the whole system.

The result of this simplified model $s(t_{k+1}, t_k, \eta_k)$ allows to estimate the exact solution $y(t_{k+1}, t_k, \eta_k)$ with an accuracy of about 10^{-4} for long time intervals $[t_k, t_{k+1}]$ greater than 1000 years. This is a very good approximation and we can expect that parallel MS converges quadratically.

6.3. Parallel multiple shooting for problems with simplified models

The parallel MS algorithm then becomes the following form.

1. $i = 1$, $\Delta\eta_k^{(0)} := 0$ for $k = 1, \ldots, m$

2. Compute on processor k: for $k=1,\dots,m$ in parallel.
   $\eta_k^{(0)}=s(t_k, t_a, y_a)=s(t_k, t_{k-1}, y_a)$
3. Compute on processor k: for $k=1,\dots,m$ in parallel
   $\eta_k^{(i)}=\eta_k^{(i-1)}+\Delta\eta_k^{(i-1)}$ and $d_k^{(i)}=y(t_{k+1}, t_k, \eta_k^{(i)})-\eta_{k+1}^{(i)}$.
4. While $\|\Delta\eta_k\| > acc$ do: For $k=1,\dots,m$ compute on processor k in parallel:
   $\Delta\eta_{k+1}^{(i)}=d_k^{(i)}+s(t_{k+1}, t_k, \eta_k^{(i)}+\Delta\eta_k^{(i)})-s(t_{k+1}, t_k, \eta_k^{(i)})$
   Send $\Delta\eta_{k+1}^{(i)}$ to processor $k+1$.
   $i=i+1$ Compute $\eta_k^{(i)}$ and $d_k^{(i)}$

If r processors are available we choose $m=r$. $\Delta\eta_{k+1}^{(i)}$ has to be computed by processor number k evaluating s (in time T_s) before $d_{k+1}^{(i+1)}$ can be computed by processor number $k+1$. Therefore the parallel integrations of the exact model in Step 4 are started with a time lag kT_s each in interval k. This results in a total start-up time of mT_s. The size of T_s depends on the complexity of s.

The speedup S with r processors can then be estimated by

$$S \approx \frac{T_{ys}}{i(T_{yp}+T_s)+rT_s}, \tag{6.2}$$

where T_{ys} is the time for sequential solution of the IVP and T_{yp} is the maximal time to compute $y(t_{k+1}, t_k, \eta_k^{(i)})$. As $T_{yp} \approx T_{ys}/(Br)$ the highest speedup is obtained for

$$r \approx \sqrt{\frac{iT_{ys}}{BT_s}} \tag{6.3}$$

In the given example the complexity of s is about $[s] \approx 6300$ floating point operations. The complexity of f in this example is about $[f] \approx 1000$ floating point operations. Thus mT_s is negligable only if many integration steps are performed in each subinterval.

Parallel MS was simulated on a workstation to integrate the solar system over intervals of different length. The optimal number $m=r$ of equidistant subintervals and simulated processors cannot be estimated a priori by (6.3). The first three rows of Table 2 show results for short intervals which a posteriori nearly fit the relation (6.3). The computing times of T_s, T_{ys} and T_{yp} in Table 2 are given in CPU seconds of a SPARC-slc. The extrapolation code ODEX2 ([9]) was used for the numerical integration. To achieve the required accuracy $acc=10^{-9}$ the local error of the integration method had to be chosen as $TOL=10^{-13}$. The last row of Table 2 shows that the speed-up in fact decreases if too many processors are used. In this case the overhead rT_s increases (c.f. (6.2), (6.3)).

Table 2
Speedup for short intervals with r nearly optimal

Interval	T_s	T_{ys}	$r=m$	i	T_{yp}	E	S
1 year	0.0156	3.70	30	3	0.22	0.56	3.15
10 years	0.0156	33.74	100	4	0.515	0.655	9.16
100 years	0.0156	331.17	500	8	0.90	0.74	21.9
100 years	0.0156	331.17	1024	8	0.518	0.624	16.35

Table 3
Speedup for a long interval (1000 years) depending on r

r	10	30	100	200	300	400	500	700	1024
S	1.68	5.00	15.53	25.41	29.28	29.49	28.05	23.99	18.43

For very long intervals (j centuries), however, the convergence becomes so slow that the algorithm 6.3 has to be combined with the Algorithm of 4.2. Thereby we divide the first 100 years into r subintervals and start the iteration. For each converging approximation at the beginning of the interval a new subinterval is appended at the end. Thus we iterate the approximations on a moving interval of about 100 years and the total number of created subinterval is approximately $m \approx jr$. It turned out that after an initial phase of about 4 iterations the interval moves about 25 years in each iteration and each η_k is corrected i_k times with a mean value of $\bar{i} = i * r/m \approx 4$.

Table 3 shows results obtained for an interval of 1000 years ($j = 10$).

Again S decreases for large r because of the overhead. But note that the complexity of s and T_s increase linearly with the number n of bodies, wheras f, T_{ys} and T_{yp} increase quadratically. Therefore T_s can be neglected in (6.2) if a system of many astronomical bodies and a long interval is regarded. As the relation T_{ys}/T_{yp} is nearly independent of n we obtain for large n

$$S \approx \frac{Br}{\bar{i}},$$

which can easily exceed 100 for $B \approx 0.6$ and $\bar{i} = 4$ and $r \approx 1000$.

Then the following type of questions can now be answered by parallel MS:

(1) Is a given astronomical multi-body system stable or chaotic?

(2) What perturbations can lead to a loss of stability within a given time?

(3) What is the reason that not all planets move in the same ecliptic plane?

For all these questions f is very small so that parallelism in the problem does not apply. The intervals of interest are very large ($> 10^9$ years) and require many floating point operations ($\gg 10^{15}$) to solve. So that also the fastest scalar computer would need many years to solve one of the problems.

If it turns out that a question can only be answered if the IVP is solved with higher accuracy then the low accuracy can be increased by additional iteration steps.

This is not possible if usual integration methods have been used.

Furthermore the matrices G_k automatically yield the information about the stability of the problem.

7. Conclusion

We have seen that parallel multiple shooting can be used to solve non-stiff linear problems very fast if f_y is given in analytical form. It can also be applied to special non-linear problems if a good approximation can be obtained very fast.

The given example is one of the very few reported applications where parallelism in the problem is not possible, computing time must essentially be reduced and parallelism in the method leads to the necessary speed-up.

It has a small right-hand side so that communication overhead does not allow an efficient parallelisation of f. Even parallel extrapolation methods would not be useful. Parallel MS, however, allows to answer questions that cannot be solved by sequential methods in a reasonable time.

For stiff problems of low dimension only a moderate speedup is possible.

8. References

[1] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions* (Dover, 1964).

[2] U.M. Ascher and S.Y.P. Chan, On parallel methods for boundary value ODEs, *Computing* 46 (1991) 1–17.

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

[4] O. Buchauer, P. Hiltmann and M. Kiehl, Sensitivity analysis of initial-value problems with application to shooting techniques, Schwerpunktprogramm der Deutschen Forschungsgemeinschaft, Technische Universität München, Report No. 403, 1992.

[5] P. Deuflhard, A relaxation strategy for the modified Newton method, in: R. Bulirsch, W. Oettli and J. Stoer, eds., *Proc. Conf. on Optimization and Optimal Control*, Oberwolfach, Germany (Nov. 17–23, 1974) *Lecture Notes in Mathematics* 477 (Springer, Heidelberg, 1974) 59–73.

[6] W.H. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of ordinary differential equations, *BIT* 15 (1975) 10–48.

[7] M. Goldmann, Vectorization of the multiple shooting method for the non linear boundary value problem in ordinary differential equations, *Parallel Comput.* 7 (1988) 97–110.

[8] E. Hairer, S.P. Nørsett and G. Wanner, *Solving ODE I* (Springer, Berlin, 1987).

[9] E. Hairer and G. Wanner, *Solving ODE II* (Springer, Berlin, 1991).

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

[11] P.J. van der Houwen and B.P. Sommeijer, Iterated Runge-Kutta methods on parallel computers, *SIAM J. Sci. Stat. Comp.* 12 (5) (1991) 1000–1028.

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

[13] P. Kaps and P. Rentrop, Generalized Runge-Kutta methods of order four with stepsize control for stiff ordinary differential equations, *Numer. Math.* 33 (1979) 55–68.

[14] H.B. Keller and P. Nelson, A hypercube Implementation of parallel shooting, *Appl. Math. Comput.* 31 (1989) 574–603.

[15] B.M.S. Khalaf and D. Hutchinson, Parallel algorithms for initial-value problems: Parallel shooting, *Parallel Comput.* 18 (1992) 661–673.

[16] M. Kiehl, Parallel one-step methods with minimal parallel stages, Technische Universität München, Report No. M-9210, 1992.

[17] S.P. Nørsett and H.H. Simonsen, Aspects of parallel Runge-Kutta methods, *Springer Lecture Notes in Mathematics* 1386 (1989) 103–117.

[18] H.J. Oberle and W. Grimm, BNDSCO - A program for the numerical solution of optimal control problems user guide, DLR IB 515-89/22, Germany, Oberpfaffenhofen, 1989.

[19] T. Ojika and W. Welsh, A numerical method for multipoint boundary value problems with application to a restricted three body problem, *Internat. J. Comput. Math.* 8 (1980) 329–344.

[20] S. Plowman, A parallel initial value ODE-solver, contributed talk at the Internat. Conf. on Parallel Methods for Ordinary Differential Equations, Grado (10–13. Sep. 1991).

[21] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis* (Springer, New York, 1980).

[22] K. Wright, Parallel treatment of block-bidiagonal matrices in the solution of ordinary differential boundary value problems, *J. Comput. Appl. Math.* 45 (1993) 191–200.

[23] S.J. Wright and V. Pereyra, Adaption of a two-point boundary value problem solver to a vector-multiprocessor environment, *SIAM J. Sci. Stat. Comput.* 11 (3) (1990) 425–449.