Comparing Task and Data Parallel Execution Schemes for the DIIRK Method

THOMAS RAUBER  GUDULA RÜNGER *

Computer Science Dep., Universität des Saarlandes, 66041 Saarbrücken, Germany

Abstract. We investigate the parallel implementation of the diagonal-implicitly iterated Runge-Kutta method, an iteration method which is appropriate for the solution of stiff systems of ordinary differential equations. We discuss different strategies for the implementation of the method on distributed memory multiprocessors, which mainly differ in the data distribution and the order of independent computations. In particular, we consider a consecutive implementation that executes the steps of each corrector iteration in sequential order and distributes the resulting equation systems among all available processors, and a group implementation that executes the steps in parallel by independent groups of processors.

1 Introduction

We consider numerical methods for the solution of initial value problems (IVPs) associated with systems of first order ordinary differential equations (ODEs)

\[
\frac{dy(t)}{dt} = f(t, y(t)), \quad y(t_0) = y_0, \quad t_0 \leq t \leq t_{\text{end}}
\]

and the numerical approximation of their solution \( y : \mathbb{R} \rightarrow \mathbb{R}^n \) on distributed memory multiprocessors (DMMs). The right hand side of System (1) is a non-linear function \( f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n \).

A class of solution methods called iterated Runge-Kutta methods have been proposed for a parallel solution of IVPs [6, 3, 8]. Iterated Runge-Kutta methods are predictor-corrector (PC) methods based on implicit Runge-Kutta (RK) correctors, i.e., the corrector steps represent an iteration of the (implicit) basic RK-method. These methods have a large degree of inherent parallelism and, therefore, they are very attractive for a parallel implementation. The stability properties of iterated RK methods depend on the way the corrector is iterated. A functional iteration (fixed point iteration) of an implicit RK corrector results in the IRK method. In [7, 8], IRK methods were proposed for a parallel implementation on shared memory machines with a small number \( s \) of processors (\( s \) is the number of stages of the corrector RK-method). In [4], IRK methods have been parallelized for DMMs. But because of their relatively limited region of stability those methods are only suitable for nonstiff ODEs. In this paper, we consider the diagonal-implicitly iterated Runge-Kutta (DIIRK) method which is appropriate for the integration of stiff systems [2, 8]. We investigate parallel implementations of the DIIRK method on DMMs with an arbitrary number

* both authors are supported by DFG
of processors. We present strategies for the parallel implementation of the DII-
RK method that differ in the data distributions and the order of computations.
The algorithms take into account special properties of the DIIRK method, e.g.,
the stepsize control with embedded solutions and a reduction of the number of
function evaluations by precomputations in the preceding corrector iteration.
In particular, we consider a consecutive implementation and a group implementa-
tion. The consecutive implementation breaks down each corrector step into
independent pieces and computes them in sequential order by distributing the
resulting equation systems among all available processors. The group implementa-
tion executes the pieces in parallel by independent groups of processors.
We have implemented the different parallel variants of the DIIRK method
on an Intel iPSC/860. The experiments take into account different numbers of
processors, different dimensions of the systems and different computational effort
for the right hand side $f$ of the ODE system. The experiments show that the
performance of the implementations depends strongly on the function $f$: For
sparse functions, the group implementation is much better and reaches medium
range speedup values. For dense functions, the consecutive implementation is
superior and reaches good speedup values. The remaining part of this article is
organized as follows: Section 2 describes the diagonal-implicitly iterated Runge-
Kutta method and some characteristic properties of the DIIRK method. Section
3 develops different parallel implementations. Section 4 presents the numerical
experiments on an Intel iPSC/860.

2 Diagonal–Implicitly Iterated Runge–Kutta Method

One time step of the DIIRK method to compute the next approximation vector
$y_{\kappa+1}$ consists of a fixed number $m$ of iteration steps, each computing $s$ stage
vectors $v_i^{(j)}$ for $l = 1, \ldots, s$ and $j = 1, \ldots, m$. The initial iteration vector is
provided by the predictor method. Choosing a simple one-step predictor method
yields the following standard algorithm $\text{Std}$ for the DIIRK method:

$$v_i^{(0)} = y_{\kappa}, \quad l = 1, \ldots, s$$  \hspace{1cm} (2)

$$v_i^{(j)} = y_{\kappa} + h \sum_{i=1}^{s} (a_{ii} - d_{ii}) f(v_i^{(j-1)}) + h d_{ii} f(v_i^{(j)}) \quad l = 1, \ldots s \quad j = 1, \ldots m$$  \hspace{1cm} (3)

$$y_{\kappa+1} = y_{\kappa} + h \sum_{i=1}^{s} b_l f(v_i^{(m)})$$  \hspace{1cm} (4)

One time step $\kappa \rightarrow \kappa+1$ according to system (2), (3), (4) is called a macrostep.
The execution of one iteration step $j \rightarrow j+1$ of (3) is called a corrector step. The
number $m$ of corrector steps determines the convergence order of the method. For
each corrector step $j$, an implicit nonlinear system of equations has to be solved
in order to get the vectors $v_1^{(j)}, \ldots v_l^{(j)}$. This is done by the Newton method
as described in [5]. Each step of the Newton method includes the computation
of a Jacobi matrix by forward difference approximations and the solution of a
linear equation system by the Gaussian elimination. The number of function
evvaluations in the corrector step $j+1$ for the computation of $v_l^{(j+1)}, l = 1, \ldots s,$