SCALABLE COMPUTING
Practice and Experience

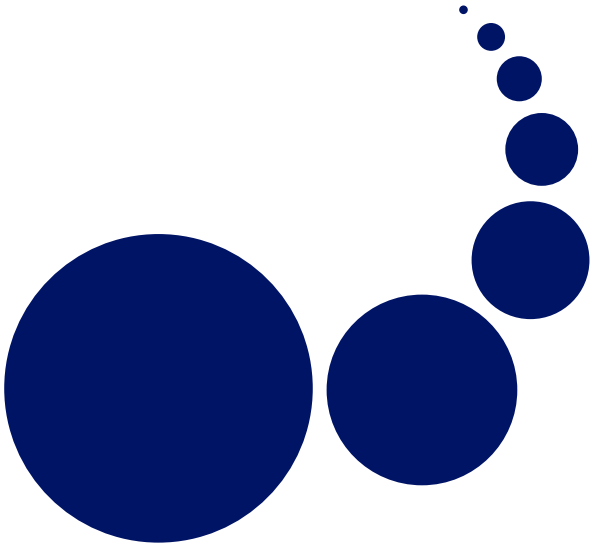
Special Issues:

Special Issue Devoted to Professor Ian Gladwell

Editors: Pierluigi Amodio and Luigi Brugnano

Selected Papers From Grid&SEA 2008 and From LaSCoG 2008

Editor: Dana Petcu



Volume 10, Number 4, December 2009

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 10, Number 4, December 2009

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. i–ii. http://www.scpe.org
ISSN 1895-1767

INTRODUCTION TO THE SPECIAL ISSUE DEVOTED TO PROFESSOR IAN GLADWELL

Dear SCPE readers,

This issue is devoted to Professor Ian Gladwell on the occasion of his retirement from the Mathematics Department of *Southern Methodist University* (SMU) in Dallas, Texas, USA. In his long career, Ian Gladwell was a member of the faculty of the *University of Manchester* (England) from 1967 to 1987 and a member of the faculty of SMU from 1987 to date. Whilst at SMU he was Chair of the Mathematics Department for two terms and served multiple terms as Director of Undergraduate Studies and as Director of Graduate Studies. Between 1975 and 2006 he supervised more than 20 Ph.D. students (see [1]) who are now working in the USA, Europe, and Asia.

Professor Gladwell has been very influential as an editor. This activity includes serving as Editor-in-Chief of the ACM Transactions on Mathematical Software from 2005 to date. He was also Associate Editor of the *IMA Journal on Numerical Analysis*; *Scalable Computing: Practice and Experience*; and the *SIAM Journal on Numerical Analysis*. He has served as editor for several books and special issues of journals. At present he serves as editor for the chapter in Scholarpedia devoted to Boundary Value Problems [2].

The many publications listed in [3] show that his research activity has focussed on the numerical integration of ordinary differential equations, quadrature, parallel computing, and mathematical software. His recent research has been concerned with the numerical solution of almost block diagonal (ABD) systems and applications. His most recent book is *Solving ODEs with MATLAB*, which he wrote with L.F. Shampine and S. Thompson. It was published by Cambridge University Press in 2003 [4].

Professor Gladwell was a pioneer in the development of mathematical software, especially software for the numerical solution of ordinary differential equations. Three software packages were published by the ACM and several of his programs were included in the NAG Fortran 77 library [5]. His association with NAG began in 1975 with his numerical ODE programs for the first NAG Library. He is a founder member of both the NAG Ltd. Technical Policy Committee and the NAG Inc. Advisory Panel. He has also been a long-term consultant for Texas Instruments.

The Special Issue contains five papers dealing with subjects related to the research activity of Ian Gladwell. Most of the authors had a fruitful collaboration with Ian in the past. We thank all of them to have agreed to our call.

The first paper *Vectorized Solution of ODEs in Matlab* is by L. F. Shampine, from the Southern Methodist University (USA). The author investigates a class of Runge-Kutta methods, able to efficiently exploit vectorization in the popular problem-solving environment Matlab. Local error estimates and continuous extensions that require no additional function evaluations are also derived. As a result, a (7,8) pair is derived and implemented in the program BV78 that well compares with the well-known Matlab ODE solver ode45, based on a (4,5) pair.

The second paper *Conditioning and Hybrid Mesh Selection Algorithms for Two-Point Boundary Value Problems* is by J. R. Cash, from the Imperial College, London (England), and F. Mazzia, from the University of Bari (Italy). The authors deal with the use of conditioning of the problem in the stepsize variation strategy. This allows to obtain very reliable algorithms, which are implemented in "state of the art" numerical codes for boundary value problems for ordinary differential equations. In particular, they speak about different choices of monitor functions that are used in the BVP codes and analyze the setting of the parameters in order to optimize the stepsize variation strategy.

The third paper *Preconditioning of Implicit Runge-Kutta Methods* is by L. O. Jay, from the University of Iowa, Iowa City (USA). A major problem in obtaining an efficient implementation of fully implicit Runge-Kutta (IRK) methods applied to systems of differential equations is to solve the underlying systems of nonlinear equations, usually obtained by application of modified Newton iterations with an approximate Jacobian matrix. In this article the author presents a cheap, and parallelizable, preconditioner for solving the linear systems with the approximate Jacobian matrix.

The fourth paper *Parallel Numerical Solution of ABD and BABD Linear Systems Arising from BVPs* is by P. Amodio, from the University of Bari (Italy), and G. Romanazzi, from the University of Coimbra (Portugal). The authors describe a parallel algorithm (based on the cyclic reduction) for the solution of linear systems with coefficient matrices having the ABD or the Bordered ABD (BABD) structures. They also report numerical tests involving parallel OpenMP versions of the Fortran 90 codes *BABDCR* and *GBABDCR* and compare them with *COLROW*. Finally, they discuss about the use of *GBABDCR* inside paralell version of BVP codes.

The fifth paper *Parallel Factorizations in Numerical Analysis* is by P. Amodio, from the University of Bari (Italy), and L. Brugnano, from the University of Florence (Italy). The authors review a number of parallel solvers for large, sparse, and structured linear systems through the use of the so called *parallel factorizations*, which provide parallel extensions of usual matrix factorizations. In particular, the paper is focused on the use of parallel factorizations for solving linear systems deriving from the numerical solution of ODEs. Moreover, the so called *Parareal* algorithm is derived within the framework of parallel factorizations.

Pierluigi Amodio,
Dipartimento di Matematica,
Università di Bari,
Bari, Italy

Luigi Brugnano,
Dipartimento di Matematica,
Università di Firenze,
Firenze, Italy

REFERENCES

[1] Ian Gladwell: Graduate Student Supervision, http://faculty.smu.edu/igladwel/phd.html
[2] Scholarpedia Boundary Value Problem, http://www.scholarpedia.org/article/Boundary_value_problem
[3] MathSciNet: Publication results for "Items authored by or related to Gladwell, Ian", http://www.ams.org/mathscinet/search/publications.html?pg1=IID&s1=74115
[4] L. F. Shampine, I. Gladwell, S. Thompson: Solving ODEs with MATLAB – Cambridge University Press, http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521530941
[5] Nag Fortran library, http://www.nag.com/numeric/FL/FLdescription.asp

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 337–345. http://www.scpe.org
ISSN 1895-1767

VECTORIZED SOLUTION OF ODES IN MATLAB

L. F. SHAMPINE*

Abstract. Vectorization is very important to the efficiency of computation in the popular problem-solving environment MATLAB. It is shown that a class of Runge-Kutta methods investigated by Milne and Rosser that compute a block of new values at each step are well-suited to vectorization. Local error estimates and continuous extensions that require no additional function evaluations are derived. A (7,8) pair is derived and implemented in a program `BV78` that is shown to perform quite well when compared to the well-known MATLAB ODE solver `ode45` which is based on a (4,5) pair.

Key words: Matlab, vectorization, ordinary differential equations, initial value problems

1. Introduction. MATLAB is a problem-solving environment (PSE) that is in very wide use. A suite of programs for solving the initial value problem (IVP) for ordinary differential equations (ODEs) in this PSE is developed in [11]. As illustrated there, both algorithms and coding practices must take into account that the costs of certain computations in MATLAB are quite different from the costs in general scientific computation. Indeed, all the programs in MATLAB for approximating $\int_a^b f(x)\,dx$ require that evaluation of the integrand be coded to accept a vector $[x^1, x^2, \ldots, x^k]$ and return a vector $[f(x^1), f(x^2), \ldots, f(x^k)]$. That is because the cost of evaluating the integrand for all these arguments in one call with an array argument is generally not much greater than the cost of a call with a single argument. There are two reasons for this. One is that the overhead of a call is relatively expensive and the other is that skillful use of array operations and vectorized built-in functions generally allows an integrand to be evaluated very efficiently for k arguments simultaneously. When evaluation of the integrand is coded in this way, the function is said to be "vectorized". In this paper we consider how vectorization might be exploited when solving numerically a system of n first-order ODEs

$$y' = f(t, y) \tag{1.1}$$

on an interval $[t_0, t_f]$ with initial value $y_0 = y(t_0)$. There are already some examples of this in MATLAB. When solving stiff IVPs, it is necessary to form Jacobian matrices $J(t, y)$ from time to time. By default these Jacobians are approximated by finite differences. If the Jacobian is dense, $J(t^0, y^0)$ is approximated using values of $f(t^0, y^m)$ for n vectors $\{y^1, y^2, \ldots, y^n\}$ that are "close" to y^0. Users are encouraged to vectorize the computation so that when t^0 and a matrix with columns $\{y^1, \ldots, y^n\}$ are input, the function will compute and return the corresponding $\{f(t^0, y^1), \ldots, f(t^0, y^n)\}$ as columns in a matrix. If full advantage is taken of array operations, the Jacobian can often be approximated at a cost not much greater than the cost of evaluating f for a single y^m. Kierzenka [5, 10] recognized that when solving boundary value problems (BVPs) with collocation methods, the cost of solving the associated nonlinear algebraic equations could be reduced very substantially if evaluation of $f(t, y)$ was vectorized so that for $\{t^1, t^2, \ldots, t^k\}$ given as entries in a vector and $\{y^1, y^2, \ldots, y^k\}$ given as columns in a matrix, the function returns $\{f(t^1, y^1), \ldots, f(t^k, y^k)\}$ as columns in a matrix. Because the argument y is generally a vector when solving ODEs, we use in this paper the term "array evaluation" of f to distinguish the evaluation of $f(t, y)$ with vector t and matrix y from an evaluation with a scalar t and vector y.

To compute starting values for multistep methods, Milne [8] suggested a scheme that is described nowadays as a block Runge-Kutta (RK) method. The scheme starts with an approximation y_0 to $y(t_0)$ and then computes simultaneously a block of approximations $y_1, y_2, \ldots, y_r$ at points $t_0 + h, t_0 + 2h, \ldots, t_0 + rh$. Later Rosser [9] proposed an explicit variant of these formulas. He uses simple (functional) iteration to evaluate the implicit formulas, but does a fixed number of iterations so as to obtain an explicit RK method. Rosser argues that these explicit block RK methods require many evaluations of f at each step, but when it is appreciated that r new values are formed, the cost per new value is lower than for conventional schemes of moderate to high order. The methods never caught on, but we observe here that they are quite attractive in MATLAB because they are so well suited to vectorization. The key observation is that each iteration requires only one array evaluation of the function. With this a high order can be obtained with a remarkably small number of (array) function evaluations. In this paper we show that it is easy to estimate the local error without additional function evaluations. Using this result, we implemented the scheme studied in detail by Rosser as a (5,6) pair in a program `R56`. We also

*Mathematics Department, Southern Methodist University, Dallas, TX 75275. E-mail: shampine@smu.edu

show that a continuous extension is available without additonal function evaluations. Appreciating that the advantages of vectorization are greater at higher order, we derived formulas analogous to R(5,6) for a (7,8) pair and a corresponding continuous extension. The `BV78` program that implements these formulas was substantially more efficient than `R56`, so we give our attention primarily to the (7,8) pair. We compare `BV78` to the MATLAB `ode45` solver because the documentation of the PSE recommends `ode45` for non-stiff problems that are to be solved for moderate to stringent accuracies. In §4 we see that the (7,8) pair has stability comparable to that of the (4,5) pair implemented in `ode45`. In §5 we find that `BV78` solves a wide range of standard test problems for a wide range of tolerances in about one third the time required by `ode45`.

2. Explicit RK Pairs in Matlab. When the ODE Suite [11] was originally conceived, the authors were very interested in implementing one of the effective (7,8) pairs derived by Fehlberg, Dormand and Prince, and Verner. The main reason this was not done is the cost of a continuous extension for the pair. To be concrete, we discuss in this article a particularly efficient pair of 13 stages due to Verner [13]. He derived continuous extensions of orders 7 and 8 that require 3 and 4 extra stages, respectively. In MATLAB it is typical that a user plot the solution. The (7,8) pairs take rather long steps, long in the sense that a solution can change substantially over the span of the step. The same is true of the (4,5) pair derived by Dormand and Prince [3] that is implemented in `ode45`, though to a lesser degree. This pair, which is known as DOPRI5, is a 7 stage pair, but it is FSAL, meaning that if the step is a success, the last stage of the step can be used as the first stage of the next step. Most steps are successful, so this pair generally costs little more than 6 stages per step. Because solution values at mesh points alone often do not provide a satisfactory plot, the `ode45` solver evaluates a continuous extension to obtain approximate solutions at some additional points equally spaced in the span of each step. Although we might have preferred a continuous extension of order 5, it would require extra function evaluations. These function evaluations are made at every step in typical use of the solver, which increases the cost of the pair by a relatively large amount. An interpolant of order 4 is implemented in `ode45` because it is an adequate approximation that costs no extra function evaluations. The extra stages for interpolation with the popular (7,8) pairs increase the cost so much at every step in typical use of a MATLAB ODE solver that we decided not to provide an `ode78`. The fact that the typical problem is solved to only moderate accuracy in MATLAB also influenced this decision. For the sake of completeness, we note that for the relatively low order BS(2,3) pair [1] implemented in `ode23`, values at mesh points alone generally do provide a satisfactory plot. Accordingly, this solver does not evaluate the continuous extension to obtain additional approximations in the span of each step. Even if it did, this would be inexpensive because the continuous extension requires no extra function evaluations.

3. Block RK Methods. A difficulty with multistep methods is to compute somehow starting values. Milne [8] suggested a scheme that is described nowadays as a block Runge-Kutta (RK) method. The method starts with an approximation y_0 to the solution at t_0 and then computes simultaneously a block of approximations $y_1, y_2, \ldots, y_r$ at points $t_0 + h, t_0 + 2h, \ldots, t_0 + rh$. We prefer to rescale so as to view the method as an implicit RK formula that steps from t_0 to $t_0 + h$ and in the course of the step forms accurate approximations at points equally spaced in the span of the step, namely $t_0 + h/r, t_0 + 2h/r, \ldots, t_0 + h$. The formulas are commonly derived by using an integrated form of the ODEs and replacing the integrals with quadrature formulas. With equally spaced points the quadrature schemes are of Newton-Cotes type, so the block RK method is also said to be of Newton-Cotes type. For our purposes it will be more convenient to derive the formulas by collocation. In this approach a polynomial $P(t)$ with $P(t_0) = y_0$ is to collocate at equally spaced points $t_j = t_0 + jh/r$ for $j = 0, 1, \ldots, r$. With the notation $y_j = P(t_j)$ and $f_j = f(t_j, y_j)$, the resulting formulas have the form

$$\begin{aligned}
y_1 &= (y_0 + h\,A_{1,0}\,f_0) + h\,[A_{1,1}\,f_1 + \ldots + A_{1,r}\,f_r] \\
y_2 &= (y_0 + h\,A_{2,0}\,f_0) + h\,[A_{2,1}\,f_1 + \ldots + A_{2,r}\,f_r] \\
&\vdots = \vdots \\
y_r &= (y_0 + h\,A_{r,0}\,f_0) + h\,[A_{r,1}\,f_1 + \ldots + A_{r,r}\,f_r]
\end{aligned}$$

We remark that if we denote these equations as $Eq_1, Eq_2, \ldots, Eq_r$, then combining and replacing them with $Eq_1, (Eq_2 - Eq_1), \ldots, (Eq_r - Eq_{r-1})$ results in a block GAM with r steps and minimum block size r as studied by Brugnano and Trigiante [2]. A theorem in [14] says that for a method of this kind, all the y_j are of local order $r + 2$, that is, they agree with $y(t_j)$ to $O(h^{r+2})$. Correspondingly, the implicit RK method is of global

order $r+1$. A familiar example is the trapezoidal rule which has $r=1$ and global order 2. It is also shown in [14] that if r is even, the value at the end of the step has a higher local order than at intermediate points, namely $r+3$, hence the method is of global order $r+2$. Clippinger and Dimsdale developed the formulas for the case $r=2$, see [12] for details. When written in the form of an implicit RK method, it is found that this is a familiar method known as the three point Lobatto or Simpson method. An important reason for choosing it as the basic formula of the BVP solver `bvp4c` [5] is that it has global order 4, the one intermediate value is also of order 4, and a natural continuous extension is uniformly of order 4. Milne [8] provides an example known as Method IV which has $r=4$. For that example the intermediate values are all of local order 6 and the RK method is of global order 6.

Implicit RK methods have not been widely used to solve non-stiff IVPs because of the cost of evaluating the formulas. Rosser [9] takes a different approach. He starts an iteration for evaluating the implicit formulas, but does a fixed number of iterations so as to obtain an explicit RK method. The case $r=1$ is a familiar example. If Euler's method is used to form the first approximation to y_1 and a single iteration of simple (functional) iteration is done, the resulting formula is an explicit RK formula of global order two known as Heun's method. In the same way Rosser proposes starting with the locally second order approximations $y_j^{[1]} = y_0 + (jh/r)\, f_0$ for $j = 1, \ldots, r$. Simple iteration consists of first forming the $f_j^{[m]} = f(t_j, y_j^{[m]})$ and then computing the $y_j^{[m+1]}$ from

$$
\begin{aligned}
y_1^{[m+1]} &= (y_0 + h\, A_{1,0}\, f_0) + h \left[A_{1,1}\, f_1^{[m]} + \ldots + A_{1,r}\, f_r^{[m]} \right] \\
y_2^{[m+1]} &= (y_0 + h\, A_{2,0}\, f_0) + h \left[A_{2,1}\, f_1^{[m]} + \ldots + A_{2,r}\, f_r^{[m]} \right] \\
&\vdots = \vdots \\
y_r^{[m+1]} &= (y_0 + h\, A_{r,0}\, f_0) + h \left[A_{r,1}\, f_1^{[m]} + \ldots + A_{r,r}\, f_r^{[m]} \right]
\end{aligned}
$$

In connection with Milne's Method IV, Rosser points out that each iteration raises the local order of the approximations $y_j^{[m]}$ by one (up to a maximum order determined by the order of the underlying implicit formula), so that after 5 iterations, they all have local order 6. When another iteration is done, the local order remains 6 at all the interior points, but the local error at the end of the step is increased to 7, resulting in a formula of global order 6. It is easily seen that this observation about the order of accuracy of the iterates is general—each iteration raises the local order of the approximations by one.

Rosser argues that these explicit block methods require many evaluations of f at each step, but when it is appreciated that r new values are formed, the cost per new value is lower than for conventional schemes of moderate to high order. Vectorization changes completely our view of the cost of these explicit block RK methods. The key observation is that when the evaluation of f is vectorized, we can form $f_j^{[m]}$ for $j = 1, 2, \ldots, r$ in a single call. Because of this a high order can be obtained with a remarkably small number of (array) function evaluations. Although Rosser did not take up any general ways of estimating the error, we noticed that this is easily done for block methods. Because each iterate raises the order by one, we not only have a result of global order 6 at the end of a step with Rosser's method, we also have a result of global order 5. With it we have a (5,6) pair with a conventional local error estimate.

We wrote a research code called `R56` that implements Rosser's explicit variation of Milne's Method IV. This code differs from Rosser's in that the formula is implemented as a (5,6) pair to control the local error at each step and vectorization of f is used to reduce the run time dramatically. As mentioned earlier, we originally considered a (7,8) pair for the ODE Suite. That was one reason we decided to investigate a block, vectorized (7,8) pair. Another was that the higher order scheme takes more advantage of vectorization. Accordingly, we used Maple [7] to derive the formulas for a block of size $r=6$ and implemented them in a modification of `R56` that we called `BV78`. The coefficients of the formulas and the continuous extension are rational, but there are a good many coefficients, so rather than list them here, we refer the reader to `BV78` itself. In our experiments this code proved to be more effective than `R56`, so in what follows we discuss only this BV(7,8) pair. At each step the differential equations are evaluated with a single argument to obtain f_0 and then each iteration requires an evaluation of f with 6 arguments. Seven iterations are needed to reach the maximum order, so each step costs 8 (array) evaluations. This is much better than the 13 evaluations of the conventional (7,8) pairs. Moreover, as we explain shortly, a continuous extension of order 7 costs no extra evaluations, whereas Verner's (7,8) pair requires 3 extra evaluations for a continuous extension of this order.

The block, vectorized Newton-Cotes formulas are very attractive as regards a continuous extension because the method is defined in terms of interpolating polynomials. Specifically, the values $y_j^{[m+1]}$ are just $P^{[m]}(t_j)$ for the polynomial $P^{[m]}(t)$ that interpolates the value y_0 and the slopes $f_j^{[m]}$ for $j = 0, 1 \ldots, r$. This means that when we reach the m that provides the desired order, we already have available a continuous extension. We might remark that Rosser [9] does not consider this natural continuous extension, preferring instead to do Hermite interpolation on a subset of the new values. The `BV78` solver we implemented shares the basic design of the ODE Suite. In particular, the default mode is for a user to supply the interval of integration. The solver then returns approximate solutions on a mesh that spans the interval. The BV(7,8) pair forms six new approximations equally spaced within each step. They are all returned by the solver. In our experiments these values have always provided a smooth graph, so this matter is handled rather more easily than in `ode45`. Another mode is for a user to supply not only the interval, but also specific points where output is desired. The solvers then return approximate solutions at these points only. For efficiency the solvers choose the same step sizes for both modes of output and compute approximations at specific points in the span of a step by evaluating the continuous extension. Evaluation of the continuous extension to get output at specific points is coded as a nested function that is vectorized with respect to output points in the span of a step.

4. Stability. It is shown in [14] that the implicit block one-step methods defined by interpolatory formulas of Newton-Cotes type are A-stable for block sizes $r = 1, 2, \ldots, 8$. Of course, the stability regions of the explicit Runge-Kutta methods defined by evaluating such a formula with Euler's method followed by a fixed number of simple iterations are all finite, but we might hope that the methods will at least have good stability. The lowest order case is the trapezoidal rule with $r = 1$. A single iteration leads to Heun's method, which is a familiar two-stage, second-order explicit RK method. All such methods have the same region of absolute stability which is displayed as Figure 7.2 in Dormand [3]. It is reasonably large, which gives us hope that the BV(7,8) pair will inherit good stability from the underlying A-stable implicit RK formula. Figure 4.1 shows the stability regions of the two formulas of the BV(7,8) pair. Both regions are quite satisfactory as regards both size and shape. And, as these things go, the stability regions of the pair match pretty well. Figure 7.4 of [3] shows the stability regions of the DOPRI5 pair used in `ode45`. The stability regions of the two pairs are comparable, but those of the BV(7,8) pair are uniformly somewhat bigger, especially in the direction of increasing imaginary part. On the other hand, the (7,8) pair costs 8 evaluations per step and the (4,5) pair only 6 for a successful step. However, this somewhat overstates the efficiency of the DOPRI5 pair because it is FSAL. This is less advantageous when stability is an issue because there are then many step failures in which case the pair costs 7 evaluations. All things considered, the stability of the BV(7,8) pair and the DOPRI5 pair may be fairly described as comparable.

Both formulas of Verner's (7,8) pair have excellent stability and their regions are almost perfectly matched. Verner states that the average radius of the eighth order formula is 5.69. The typical solution of an IVP in MATLAB displays the solution graphically and for high order formulas it is necessary to produce some additional solution values in the span of each step so as to get a smooth graph. When used in this way, Verner's pair costs 16 or 17 stages per step. The BV(7,8) pair costs 8 (array) evaluations per step and the interpolant costs no extra evaluations. Even if we assume that no interpolation is done, scaling the radius of the stability regions of Verner's pair to the same number of evaluations as the BV(7,8) pair results in an equivalent radius of $5.69 \times 8/13 = 3.50$. This is quite a bit smaller than the average radius 4.66 of the eighth order formula of BV(7,8). It appears, then, that the BV(7,8) pair is rather more stable than Verner's (7,8) pair provided that the cost of evaluating f is a weak function of the number of arguments.

In §5 we describe some numerical experiments with `BV78` and `ode45` applied to a test set of Krogh [6]. Details of the computations are provided in that section, but the problem K7 is pertinent here. The IVP

$$y' = t(1-y) + (1-t)e^{-t}, \qquad y(0) = 1$$

is to be solved on $[0, 50]$ with a pure absolute error control. The eigenvalue $-t$ of the local Jacobian increases in magnitude as the integration proceeds, making the problem increasingly stiff. Figure 4.2 shows how much accuracy is achieved for a given number of function evaluations. In this no distinction is made between calling the function with one or six arguments. As we might have expected from the stability regions and costs per step, the two codes perform much the same. Table 5.1 shows that the same is true with respect to run times.

5. Illustrative Computations. Krogh [6] proposed a small set of test problems carefully chosen to illuminate the behavior of programs for solving non-stiff IVPs for ODEs. We have used them to compare

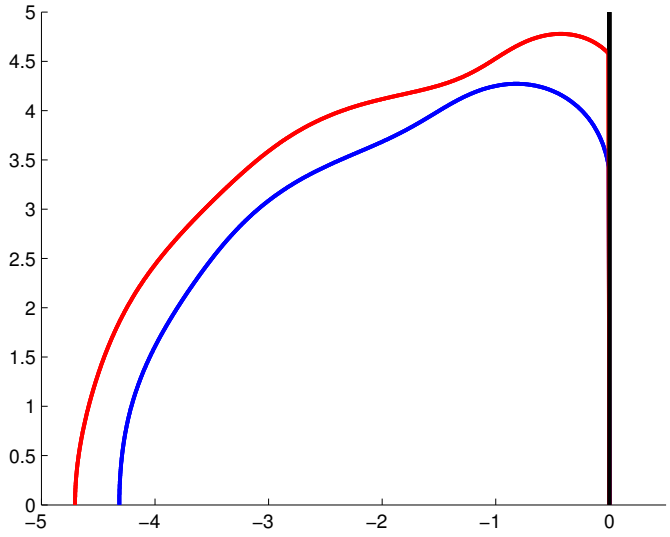


FIG. 4.1. *Stability regions for BV(7,8). Order 8 is the larger region.*

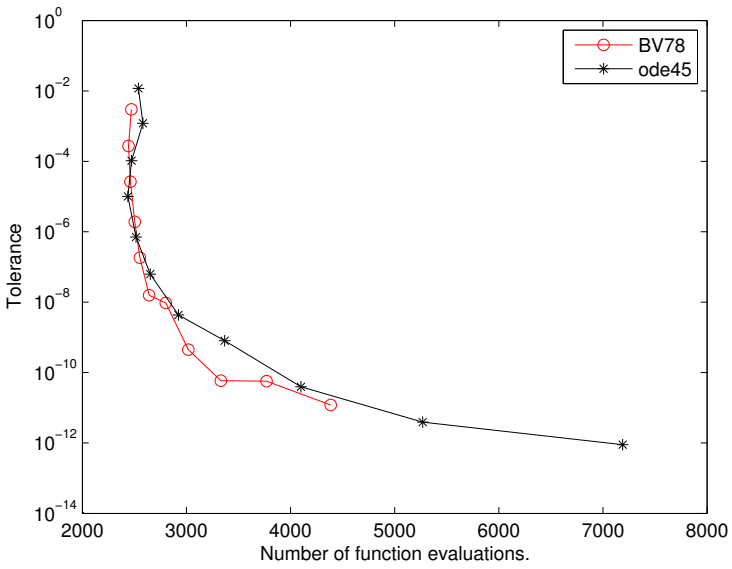


FIG. 4.2. *K7 tests stability along the negative real axis.*

numerically `BV78` and `ode45`. The test programs and `BV78` itself are available at `http://faculty.smu.edu/shampine/current.html`. Problem 5 is a single second order ODE and problem 4 is the same equation written as a first order system. The solvers we consider require the ODEs to be written as first order systems, so we had to exclude problem 5 from the comparison. Each problem in the test set is to be solved with either a pure relative or pure absolute error tolerance. The design of the MATLAB ODE Suite [11] permits neither kind of control. Its mixed error control requires that the relative error tolerance be at least 100 units of roundoff. Problems that are to be solved with a pure absolute error control were solved in these tests with the specified absolute error tolerance and this minimum relative error. The design also requires that the absolute error tolerance be greater than zero, though there is no minimum value. Problems that are to be solved with a pure relative error control were solved with the specified relative error tolerance and an absolute error tolerance of

10^{-100}. Analytical solutions are available for all the problems except K9 and K10. Reference values for the two body problem K9 were obtained by solving accurately Kepler's (algebraic) equation. Reference values for the restricted three body problem K10 were computed using `ode113` with the minimum relative error tolerance of 2.2×10^{-16} and absolute error tolerance of 10^{-100}. This should provide a solution accurate enough to compare the solvers for tolerances $10^{-2}, 10^{-3}, \ldots, 10^{-12}$. For some problems one or more of these tolerances was excluded from the comparison because one of the solvers had an error bigger than 1.

An important issue in comparing solvers is where they are to be compared. An approach that is typical of methods without continuous extensions is to measure the accuracy only at mesh points. Krogh specifies the points where the accuracy is to be measured. However, he specifies only a few points and as a consequence, quality solvers without continuous extensions are not much affected by output at these points. Our approach is more revealing and closer to the way solvers are used in MATLAB. We compared the accuracy at 200 points equally spaced in the interval of integration. More precisely, we measured the maximum over the 200 points of the maximum relative or absolute error, as the case may be, in the components of the solution at each point. This tests not just the accuracy of the solver, but also the accuracy of the continuous extension. Furthermore, it tests the overhead of evaluating the continuous extension along with the overhead of the solver itself.

It is difficult to obtain consistent run times in MATLAB. This is especially true when the run times are small in absolute terms, *as they are for these test problems*. We summed the run times over the range of tolerances and report only the leading digits. The total times reported in Table 5.1 were obtained in a single run of each program preceded by a "`clear all`" command. We have done this repeatedly and found the times to be consistent, though the last digit displayed might vary slightly. Still, *the run times displayed in the table should be considered only a rough guide as to relative costs*. The results might be summarized by adding up the run times over the nine problems and dividing to find that `BV78` solves a wide range of problems over a wide range of tolerances in about one third the time taken by `ode45`. Each program also plots efficiency in terms of the accuracy achieved versus the number of array evaluations. Details vary from problem to problem, but Figure 5.1 shows what might be described as a typical plot that was produced when solving K6. By virtue of vectorization `BV78` is competitive even at modest tolerances and at the more stringent tolerances its higher order gives it a substantial advantage.

TABLE 5.1
Run times for test set of Krogh [6].

Problem	$K1$	$K2$	$K3$	$K4\_5$	$K6$	$K7$	$K8$	$K9$	$K10$
`BV78`	0.2	0.6	0.6	0.7	0.7	1.2	1.0	1.2	0.5
`ode45`	0.6	2.3	2.2	2.7	2.5	1.6	2.7	3.7	2.0

A larger test set of Hull et alia [4] has five categories, each having five problems. We have compared `BV78` to `ode45` on all these problems exactly as we did with Krogh's test set. All the problems of this set are to be solved with pure absolute error control. It is worth remark that group D consists of a two body problem with initial conditions that lead to orbits of different eccentricity. A pure absolute error tolerance does not provide a good control of the error over the whole interval for the more eccentric orbits, so more tolerances were excluded in the comparison for this group than others. To make the data easier to assimilate, we report in Table 5.2 the sum of the run times for each category. When the run times are summed over the categories, we find just as with Krogh's test set that `BV78` is faster than `ode45` by about a factor of three over a wide range of problems and a wide range of tolerances.

TABLE 5.2
Run times for test set of Hull et alia [4].

Category	A	B	C	D	E
`BV78`	0.7	2.1	4.4	5.2	1.6
`ode45`	2.9	6.9	10.2	17.2	6.7

The C5 problem is a reduced model of the solar system that we use to make some points about vectorization. It is tedious to program the 30 first order ODEs of this model, though the array functions of MATLAB not only make this easier, but also speed up evaluation of the ODEs. This is important when solving IVPs with *any*

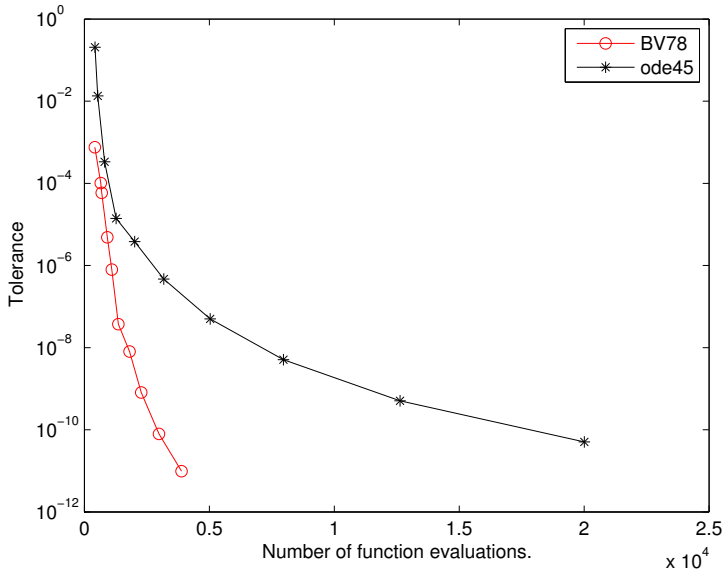


FIG. 5.1. *K6 is Euler's equations of motion of a rigid body.*

of the MATLAB ODE solvers. Indeed, we found that different ways of coding the ODEs might affect the run time with `ode45` by as much as an order of magnitude! In this paper we further assume that the ODEs are evaluated efficiently for an array of arguments. When coded in this way, `BV78` solved the problem for a wide range of tolerances in 2.2s and `ode45` did this in 6.1s. Figure 5.2 shows the efficiency in terms of the number of array evaluations. There seem to be too few data points for `BV78` in this figure. That is because for absolute error tolerances $10^{-2}, \ldots, 10^{-6}$, the results are all the same—the default maximum step size of one tenth of the length of the interval of integration already provides an error of about 5×10^{-8}. For testing purposes we provide an option in `BV78` of integrating with only one argument in each call to evaluate the ODEs. The run time of 10.2s when this is done shows the importance of vectorizing. Nonetheless, Figure 5.3 shows that even without vectorization, `BV78` is reasonably efficient at all tolerances and at stringent tolerances is comparable to `ode45` because the higher order compensates for more expensive steps.

6. Conclusions. Our analysis and numerical experiments support the conclusion that if it is convenient to vectorize the function that evaluates the differential equations, `BV78` will be comparable to `ode45` at all tolerances and considerably more efficient at stringent tolerances. In fact, for a wide range of standard test problems solved for a wide range of tolerances, `BV78` required only one third the run time of `ode45`. This conclusion assumes that the cost of evaluating the vectorized function is not much greater than the cost of evaluating it with a single argument. If that is not the case, `BV78` will solve the problem with acceptable efficiency, but it will be slower than `ode45` at all but stringent tolerances.

Though the scheme we have implemented seems to be a good one, it suggests other possibilities that merit future investigation: The scheme described for evaluating the BV(7,8) pair forms approximate solutions of all local orders from 2 to 8. Clearly it would be possible to develop a variable order code. We have discussed block methods of Newton-Cotes type, but there are other kinds of block methods that might have advantages. For example, a family of implicit block methods with superior stability is studied in [14].

7. Acknowledgement. This paper was improved as a result of valuable comments from W. Enright of the University of Toronto and P. Muir of Saint Mary's University.

REFERENCES

[1] P. BOGACKI AND L. F. SHAMPINE, *A 3(2) Pair of Runge-Kutta Formulas*, Appl. Math. Letters, 2 (1989), pp. 1–9.
[2] L. BRUGNANO AND D. TRIGIANTE, *Solving Differential Problems by Multistep Initial and Boundary Value Methods*, Gordon and Breach, Amsterdam, 1998.

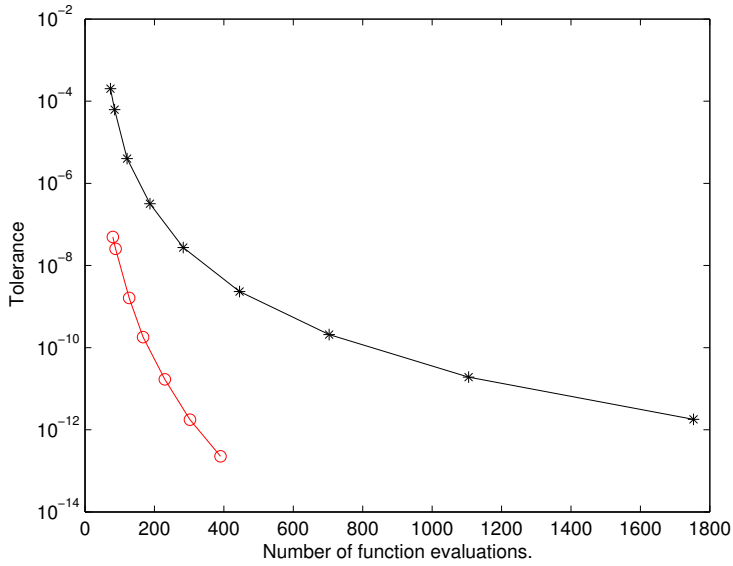


FIG. 5.2. *C5 is a five body model of the solar system.*

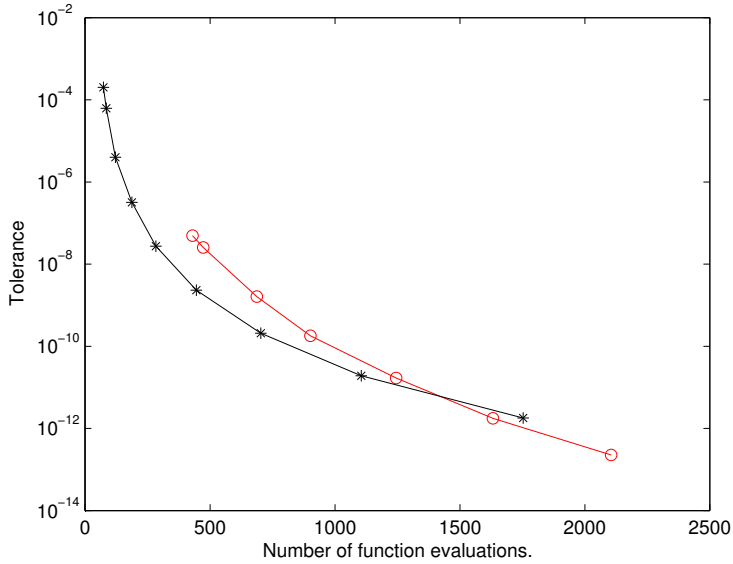


FIG. 5.3. *C5 when evaluation of the ODEs is* not *vectorized.*

[3] J. R. DORMAND, *Numerical Methods for Differential Equations a Computational Approach*, CRC Press, Boca Raton, FL, 1996.
[4] 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), pp. 603–637.
[5] J. KIERZENKA AND L.F. SHAMPINE, *A BVP Solver Based on Residual Control and the* MATLAB *PSE*, ACM Trans. Math. Softw., 27 (2001), pp. 299–316.
[6] F. T. KROGH, *On Testing a Subroutine for the Numerical Integration of Ordinary Differential Equations*, J. ACM, 20 (1973), pp. 545–562.
[7] MAPLE 11, Maplesoft, 615 Kumpf Dr., Waterloo, CA.
[8] W. E. MILNE, *Numerical Solution of Differential Equations*, Dover, Mineola, NY, 1970.
[9] J.B. ROSSER, *A Runge-Kutta for All Seasons*, SIAM Rev., 9 (1967), pp. 417–452.
[10] L. F. SHAMPINE AND J. KIERZENKA, *A BVP Solver that Controls Residual and Error*, JNAIAM, 3 (2008), pp. 27–41.

[11] L. F. Shampine and M. W. Reichelt, *The* Matlab *ODE Suite*, SIAM J. Sci. Comp., 10 (1997), pp. 1–22.
[12] L. F. Shampine and H. A. Watts, *Block Implicit One-step Methods*, Math. Comp., 23 (1969), pp. 731–740.
[13] J. H. Verner, A 'Most Efficient' Runge-Kutta (13:7,8) Pair, available at `http://www.math.sfu.ca/~jverner/`.
[14] H. A. Watts and L. F. Shampine, *A-Stable Block Implicit One Step Methods*, BIT, 12 (1972), pp. 252–256.

Edited by: Pierluigi Amodio and Luigi Brugnano
Received: January 23, 2009
Accepted: May 15, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 347–361. http://www.scpe.org
ISSN 1895-1767

CONDITIONING AND HYBRID MESH SELECTION ALGORITHMS FOR TWO-POINT BOUNDARY VALUE PROBLEMS*

JEFF R. CASH† AND FRANCESCA MAZZIA‡

Abstract. Boundary value problems for ordinary differential equations (BVODES) occur in a great many practical situations and they are generally much harder to solve than initial value problems. Traditionally codes for BVODES did not take into account the conditioning of the problem and it was generally assumed that the problem being solved was well conditioned so that small local errors gave rise to correspondingly small global errors. Recently a new generation of codes which take account of conditioning has been developed. However most of these codes are based on a rather ad hoc approach with the need to choose several heuristics without any real guidance on how these choices can be made. In this paper we identify clearly which heuristics need to be chosen and we discuss different choices of monitor functions that are used in our codes. This has the important effect of unifying the various approaches that have recently been proposed. This in turn allows us, in the present paper, to introduce a new technique for computing the conditioning which is ideally suited to BVODES.

1. Introduction. The task of solving systems of nonlinear two-point boundary value problems numerically has, for a long time, received a great deal of attention. Boundary value problems for ordinary differential equations (BVODES) occur in a great many practical situations and they are generally much harder to solve than initial value problems. In particular the numerical solution of singular perturbation problems can be especially difficult because such equations can have solutions with very narrow regions of rapid variation typified by boundary layers, shocks and interior layers.

Traditionally codes for BVODES did not take into account the conditioning of the problem and it was generally assumed that the problem being solved was well conditioned so that small local errors gave rise to correspondingly small global errors. However, in an important paper by Shampine and Muir [28], the need to consider the conditioning of the problem being solved was clearly demonstrated by means of a numerical example and this brought into focus some important earlier work by Brugnano and Trigiante [5, 6] and by Mazzia and Trigiante [27] who derived codes with a monitor function depending on both conditioning and error. Subsequently a new generation of codes which take account of conditioning has been developed. However most of these codes are based on a rather ad hoc approach with the need to choose several heuristics without any real guidance on how these choices can be made. In this paper we identify clearly which heuristics need to be chosen and we discuss different choices of monitor functions that are used in our codes. This has the important effect of unifying the various approaches that have recently been proposed. This in turn allows us, in the present paper, to introduce a new technique for computing the conditioning parameters which is ideally suited to BVODES. Finally we describe some codes which implement these new ideas and we give some numerical results obtained from using these codes.

We will only consider in this paper singular perturbation boundary value problems, although the algorithms developed are designed for general first order boundary value problems. We will not be concerned in this paper with shooting methods but will instead confine our attention to so called finite difference or boundary value methods. However a lot has been done on estimating the conditioning of shooting methods and the interested reader is referred to [1, 20, 21, 19, 13].

In section 2 we give a review of the algorithms based on conditioning for singularly perturbed boundary value problems. In section 3 we present the conditioning parameters for the continuous problem and in section 4 the corresponding parameters for the discrete one. In section 5 we analyze the hybrid mesh selection strategies based on conditioning and the way they have been unified and updated. In section 6 we present the results of some numerical experiments to analyze the behavior of the codes with the new condition estimator.

2. Singular Perturbation Problems.

2.1. Second Order Scalar Problems.

The first attempt to use conditioning in algorithms for the solution of singularly perturbed BVODES was made in [24, 8] by Mazzia and Trigiante. The algorithm presented in [24, 8] was designed for the following general class of scalar linear problems:

*Work developed within the project "Numerical methods and software for differential equations"
†Department of Mathematics, Imperial College, South Kensington, London SW7, England
‡Dipartimento di Matematica, Università di Bari, Via Orabona 4, I-70125 Bari, Italy

$$\begin{aligned} &\epsilon y'' + p(x)y' + q(x)y = f(x) \\ &y(a) = y_a, \;\; y(b) = y_b, \;\; a \le x \le b, \end{aligned} \tag{2.1}$$

where ϵ is a positive parameter which is small compared with $b-a$, $q(x)$ and $f(x)$ are continuous functions and $p(x)$ is differentiable.

If we discretize (2.1) using simple three point finite difference formulae, the discrete problem is a tri-diagonal system of algebraic equations that can be solved to give an approximation to the solution of (2.1). In matrix form this system is

$$\mathbf{T}\boldsymbol{y} = f \tag{2.2}$$

where $y_0, y_1, \ldots, y_n$ is the numerical solution computed on the mesh $\pi = \{x_0, x_1, \ldots, x_n\}$, $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$ and

$$\mathbf{T} = \begin{pmatrix} 1 & \tau_1 & & & \\ \sigma_1 & 1 & \tau_2 & & \\ & \sigma_2 & 1 & \ddots & \\ & & \ddots & \ddots & \tau_{n-1} \\ & & & \sigma_{n-1} & 1 \end{pmatrix}. \tag{2.3}$$

The matrix T depends on π and the algorithm presented in [24, 8] uses sufficient conditions for the well conditioning of tridiagonal matrices derived in [3] to compute the mesh. This allows us to derive, for example, the conditions that a constant stepsize, h, should satisfy in order to have T well conditioned. In general we require $h \approx O(\sqrt{\epsilon})$. However in the much more important case where we solve (2.1) using a variable meshsize the mesh must be choosen in order to have the $n \times n$ system (2.2) such that $n \ll \dfrac{1}{\epsilon}$.

In [24, 8] Mazzia and Trigiante, using only information related to the well conditioning of the tridiagonal matrix T, derive second order methods for the solution of (2.1) where the grid is chosen so that

1. The user requested accuracy is achieved.
2. The inverse coefficient matrix T^{-1} exists and its condition number is either independent of n (well conditioned) or grows as n or n^2 (weakly well conditioned).

The way that these two conditions are satisfied is to choose h_i in order to have (σ_i, τ_i) that satisfy the conditions for the (weakly) well-conditioning of T. The algorithm simplifies considerably if σ_i and τ_i have constant sign. We note that in a classical approach, where ϵ is not small, it would be normal to choose σ_i and τ_i so that the matrix T is diagonally dominant. However in this case h is of the order ϵ and n is of order $\frac{1}{\epsilon}$ and this is unacceptable when ϵ is very small. An algorithm that chooses τ_i and σ_i so that 1) and 2) are satisfied is described in [8] and some numerical results presented for singular perturbation problems show the power of the method.

2.2. First Order Systems of Singular Perturbation Problems. Although the algorithm described in the previous section represented a major step forward in the solution of singular perturbation problems it has the disadvantage that it is only applicable for linear, scalar, second order systems, it is of low order and it is not very easy to apply for variable stepsizes. However the strategy of using conditioning in choosing the mesh size is an important one and in the following sections we will discuss how to deal with nonlinear systems of BVODES.

In what follows we will be concerned with the nonlinear system of singular perturbation problems

$$D(\epsilon)y'(x) = f(x, y), \;\; a \le x \le b, \quad g(y(a), y(b)) = 0, \tag{2.4}$$

where $D(\epsilon)$ is a diagonal matrix whose elements depend linearly on ϵ. Usually $D(\epsilon) = diag(1, \ldots, 1, \epsilon)$, and $y, f,\ g \in R^m$. As in the previous section we will be particularly interested in the case where $0 < \epsilon \ll 1$. An efficient numerical method for the solution of (2.4) needs to be able to use non-uniform grids so that many

grid points are clustered in regions of rapid variation of the solution and relatively few grid points are placed in regions where the solution is smooth. Most codes attempt to control the error in the solution either by estimating the discrete error at the mesh points [7] or else by controlling a residual [28]. The important point to realise is that in both cases the codes attempt to control the local error on the assumption that the problem is well conditioned so that if the local error is small then the global error will also be small. However in the case where the problem is ill-conditioned a standard backward error analysis shows that it is possible for an accepted solution to have a small local error but to have an unacceptably large global error. There is the additional problem that if a monitor function ([1], p. 363) takes no account of conditioning of a problem then the grid choosing algorithm may become very inefficient. This is manifested by a sort of cycling where points are added into the grid on conditioning considerations and are then removed in the next remeshing due to accuracy considerations. To illustrate these ideas we present a test problem which we solve by using the code TWPBVPC [11] both with the standard mesh selection strategy based on the estimation of the local error and with a mesh selection strategy which considers both conditioning and local error estimation. We note that to change from the code TWPBVPC, which takes account of conditioning, to TWPBVP, which uses a conventional mesh choosing strategy, we need to change just one input parameter of the code TWPBVPC.

Example 1. We examine the following problem [11] :

$$\begin{aligned} &\epsilon y'' + xy' = -\epsilon\pi^2\cos(\pi x) - \pi x\sin(\pi x), \\ &y(-1) = -2, y(1) = 0, \end{aligned} \tag{2.5}$$

whose exact solution is $\cos(\pi x) + \mathrm{erf}(x/\sqrt{2\epsilon})/\mathrm{erf}(1/\sqrt{2\epsilon})$, solved using the code TWPBVP when conditioning is not taken into account. The problem is rewritten as a first order system with two components $(y, y')^T$ and the input parameters are $\epsilon = 10^{-7}$ and $tol(y) = 10^{-8}$. This means that we are seeking an approximation to the solution with an error $0.5 \cdot 10^{-8}$ or less in the y component. The code gives a solution using a final mesh of 4192 points with a maximum error of $0.17 \cdot 10^{-8}$. The mesh sizes used in the intermediate steps of the computation are: 16, 31, 61, 121, 241, 481, 530, 1059, 1108, 2215, 2242, 4483, 8965, 19912, 4192. It seems that the code finds the problem very difficult to solve and it needs 19912 mesh points to get some worthwhile information about the solution. This is mainly due to the fact that the mesh selection algorithm adds extra points in the wrong place. If we use the same code with the mesh selection based on conditioning, and we call this TWPBVPC, the solution is obtained using 368 mesh points with maximum error of $0.85 \cdot 10^{-8}$. The meshes used are of sizes: 16, 31, 58, 85, 169, 368. The information given by the conditioning parameters allows the code to put the mesh points in the correct place and makes it considerably more efficient for the solution of this problem.

The first papers to consider this problem of estimating the conditioning constants in a serious way were by Brugnano and Trigiante [4, 5] and by Mazzia and Trigiante [27]. Their basic approach is to identify two constants which characterise the conditioning of the continuous problem. Having done this the monitor function used in the mesh choosing algorithm is based on the relative size of these two parameters as well as on a local error estimate. The basic aim of the algorithm described in [27] is to choose the mesh so that the discrete and continuous problems have conditioning parameters which have the same order of magnitude. Perhaps the first really powerful code that used a monitor function based on accuracy and conditioning was the code TOM [27]. Numerical results presented in [27] show the excellent performance of this code compared with the case where conditioning is not considered. Using ideas explained in [27] the two deferred correction codes TWPBVP and TWPBVPL were modified to include conditioning in their monitor function and the much improved performance of the modified codes TWPBVPC and TWPBVPLC can be seen from the results on the authors' web page [11].

3. Conditioning parameters. To introduce the conditioning parameters which will be used in the mesh selection strategy of our codes, let us consider for simplicity the following linear boundary value problem:

$$\frac{dy}{dx} = A(x)y(x) + q(x), \quad a \le x \le b, \quad B_a y(a) + B_b y(b) = \beta,\ \beta \in R^m \tag{3.1}$$

whose solution is given by

$$y(x) = Y(x)Q^{-1}\beta + \int_a^b G(x,t)q(t)dt. \tag{3.2}$$

Here $Y(x)$ is a fundamental solution, $Q = B_a Y(a) + B_b Y(b)$ is non singular and $G(x,t)$ is the Greens' function. Using the ∞-norm we can compute the conditioning parameter by considering a perturbed equation:

$$\frac{du}{dx} = A(x)u + q(x) + \delta(x),\ a \le x \le b, \quad B_a u(a) + B_b u(b) = \beta + \delta\beta.$$

Here $\delta(x)$ and $\delta\beta$ are small perturbations of the data. The difference between the two solutions satisfies:

$$||u(x) - y(x)|| \le ||Y(x)Q^{-1}\delta\beta|| + ||\int_a^b G(x,t)\delta(t)dt||. \tag{3.3}$$

After some algebraic manipulation we obtain:

$$\max_{a\le x\le b} ||u(x) - y(x)|| \le \kappa_1 \|\delta\beta\| + \kappa_2 \max_{a\le x\le b} ||\delta(x)||,$$

and

$$\max_{a\le x\le b} ||u(x) - y(x)|| \le \kappa \max(\|\delta\beta\|, \max_{a\le x\le b} ||\delta(x)||),$$

where

$$\kappa_1 = \max_{a\le x\le b} ||Y(x)Q^{-1}||, \qquad \kappa_2 = \sup_x \int_a^b ||G(x,t)||dt,$$

and

$$\kappa = \max_{a\le x\le b} (||Y(x)Q^{-1}|| + \int_a^b ||G(x,t)||dt).$$

Following the same procedure as above and using the 1-norm we obtain

$$\frac{1}{b-a}\int_a^b ||u(x) - y(x)||dx \le \gamma_1 \|\delta\beta\| + \gamma_2 \max_{a\le x\le b} ||\delta(x)||,$$

and

$$\frac{1}{b-a}\int_a^b ||u(x) - y(x)||dx \le \gamma \max(\|\delta\beta\|, \max_{a\le x\le b} ||\delta(x)||),$$

where

$$\gamma_1 = \frac{1}{b-a}\int_a^b ||Y(x)Q^{-1}||dx, \qquad \gamma_2 = \frac{1}{b-a}\int_a^b\int_a^b ||G(x,t)||dtdx,$$

and

$$\gamma = \frac{1}{b-a}\int_a^b (||Y(x)Q^{-1}|| + \int_a^b ||G(x,t)||dt)dx.$$

For many problems of interest the relative sizes of the two parameters κ and γ tell us about the conditioning of the continuous problem. However in the discrete case the parameter γ is very difficult to compute. In general, in a problem where the correct dichotomy is present only κ_1 and γ_1 are of interest. In [4] a problem with κ_1 large and γ_1 small was called stiff, the stiffness ratio being $\dfrac{\kappa_1}{\gamma_1}$. This definition, although very often giving the correct information about the stiffness, may fail for non scalar problems, as pointed out in [6]. In [18] Iavernaro, Mazzia and Trigiante give a slightly different definition of stiffness. The conditioning parameters presented in [18], called $\kappa_{1,c}([a,b])$ and $\gamma_{1,c}([a,b])$, only depend on the perturbations of the boundary conditions, and are defined by supposing that $\delta(x) \equiv 0$ in (3.3). The way in which these parameters are defined is as follows:

$$\begin{aligned}
\kappa_{1,c}([a,b],\delta\beta) &= \frac{\max_{a\le x\le b}\|u(x)-y(x)\|}{\|\delta\beta\|}, \qquad & \kappa_{1,c}([a,b]) &= \max_{\delta\beta}\kappa_{1,c}([a,b],\delta\beta),\\
\gamma_{1,c}([a,b],\delta\beta) &= \frac{\int_a^b\|u(x)-y(x)\|dx}{(b-a)\|\delta\beta\|}, & \gamma_{1,c}([a,b]) &= \max_{\delta\beta}\gamma_{1,c}([a,b],\delta\beta).
\end{aligned} \tag{3.4}$$

Upper bounds on $\kappa_{1,c}([a,b]$ and $\gamma_{1,c}([a,b])$ can be obtained in terms of the parameters previously introduced, and it can be shown that

$$\kappa_{1,c}([a,b]) \le \kappa_1, \qquad \gamma_{1,c}([a,b]) \le \gamma_1. \tag{3.5}$$

This definition of $\kappa_{1,c}([a,b],\delta\beta)$ and $\gamma_{1,c}([a,b],\delta\beta)$ allows us to define the stiffness ratio which is defined as

$$\sigma_c([a,b]) = \max_{\delta\beta}\frac{\kappa_{1,c}([a,b],\delta\beta)}{\gamma_{1,c}([a,b],\delta\beta)}.$$

With these parameters it is possible to give the definitions of well conditioned, stiff and ill conditioned problems, see [18, 23] for details:

Well conditioned: κ, κ_1, γ_1 and $\sigma_c([a,b])$ are of moderate size;
Stiff: $\sigma_c([a,b]) \gg 1$;
Ill conditioned: $\kappa \gg 1$ and $\gamma \gg 1$.

If $\sigma_c([a,b])$ is large we are dealing with problems possessing different time scales for which the growth or decay rates of some fundamental solution modes are very rapid compared to others. Many singularly perturbed BVODES have $\sigma_c([a,b])$ large. Our aim is to choose the mesh so that the continuous and discrete problems have similar conditioning parameters. This leads us to investigate the conditioning of the discrete problem and this we do in the next section.

4. Conditioning parameters for the discrete problems. In order to use the conditioning parameters in a mesh selection strategy we first need to compute a discrete approximation to them which can be used in our numerical method. If in our algorithm we use a Newton iteration scheme to solve the nonlinear algebraic equations we need to solve a linear system of algebraic equations of the form $My = b$ for each iteration. The matrix M depends on the numerical scheme and on the stepsize used. We use a grid $\pi = \{x_0, x_1, \ldots, x_n\}$ with grid spacing $h_i = x_i - x_{i-1}$, $i = 1, \ldots, n$ on which to solve the problem. The block matrix M, of size $(n+1)m \times (n+1)m$, is set up so that the boundary conditions appear only in the first row block of b. Ascher, Mattheij and Russell [1] prove that for one-step schemes $\|M^{-1}\|_\infty \approx \kappa$ and this is a fundamentally important result. For the computation of $\|M^{-1}\|$ we have used up to now the algorithm presented by Higham in [15], which computes an approximation to the 1-norm of a matrix. This algorithm has now been optimized in order to use the information that we already know to compute the estimation of κ_1 and γ_1. Using the definition given in (3.4) it is possible to define the values of $\kappa_{1,d}(\pi,\delta\beta)$, $\gamma_{1,d}(\pi,\delta\beta)$ as:

$$\begin{aligned}
\kappa_{1,d}(\pi,\delta\beta) &= \tfrac{1}{\|\delta\beta\|}\max_{0\le i\le n}\|y_i\|, & \kappa_{1,d}(\pi) &= \max_{\delta\beta}\kappa_{1,d}(\pi,\delta\beta),\\
\gamma_{1,d}(\pi,\delta\beta) &= \tfrac{1}{(b-a)\|\delta\beta\|}\sum_{i=1}^n h_i\max(\|y_i\|,\|y_{i-1}\|), & \gamma_{1,d}(\pi) &= \max_{\delta\beta}\gamma_d(\pi,\delta\beta),
\end{aligned} \tag{4.1}$$

where $y_i, i = 0, 1, \ldots, n$ is the solution of the discrete problem having $\delta\eta$ as boundary conditions and $\kappa_{1,d}(\pi)$ and $\gamma_{1,d}(\pi)$ are the discrete approximations of κ_1 and γ_1 respectively. The *discrete stiffness ratio* is:

$$\sigma_d(\pi) = \max_{\delta\beta}\frac{\kappa_{1,d}(\pi,\delta\beta)}{\gamma_{1,d}(\pi,\delta\beta)}.$$

Since in the numerical codes we need to use easy to compute parameters, we define upper and lower bounds for them using the information given by the matrix M^{-1}. We define the matrices $G = M^{-1}$ and Ω having elements $\Omega_{ij} = ||G_{ij}||_\infty$ of size $(n+1)\times(n+1)$. Here G_{ij} is the i,jth block element of size m appearing in G.

The discrete approximations to κ_1 and γ_1 satisfy the following upper bounds that correspond to the discrete parameters computed in [27, 5, 6, 8, 9]:

$$\kappa_{1,d}(\pi) \leq \max_i \Omega_{i0}, \qquad \gamma_{1,d}(\pi) \leq (\sum_{i=1}^{N} h_i \max(\Omega_{i-1,0}, \Omega_{i0}))/(b-a). \tag{4.2}$$

In the following we approximate the discrete conditioning parameters by their upper bounds. Moreover, taking into account the relation between the matrix G and the Green's function, a discrete approximation of κ_2 is given by $\|g^r_{m+1,(n+1)m}\|_1$ (we consider only the components from $m+1$ to $(n+1)m$), where g^r is the row of the matrix G such that $\|g^r\|_1 = \|G\|_\infty$. Since the first block column of the matrix G_{*0} is a discrete approximation of $Y(t)Q^{-1}$ we have that the *i-th* column of G_{*0} is an approximation of the solution of the continuous problem (3.3) when $\delta(x) = 0$ and $\delta\beta = e_i$, with e_i being the column i of the identity matrix of size m. We are able now to define, denoting by $g^{(i)}$ the *i-th* column of G_{*0}, and by $g^{(i)}_j, 0 \leq j \leq N$ its blocks of size m,

$$\kappa_{1,d}(\pi, e_i) = \|g^{(i)}\|_\infty,$$

and

$$\gamma_{1,d}(\pi, e_i) = \frac{1}{b-a} \sum_{j=1}^{n} (x_j - x_{j-1}) \max(\|g^{(i)}_{j-1}\|_\infty, \|g^{(i)}_j\|_\infty).$$

A lower bound for $\sigma_d(\pi)$ is computed as

$$\sigma_d(\pi) \geq \max_{e_i, 1 \leq i \leq m} \frac{\kappa_{1,d}(\pi, e_i)}{\gamma_{1,d}(\pi, e_i)}. \tag{4.3}$$

We note that the new value of $\sigma_d(\pi)$ differs from the one computed in [6] because, instead of considering the ratio $\frac{\kappa_{1,d}(\pi)}{\gamma_{1,d}(\pi)}$ involving the norm of G_{i0}, we consider the columns of G_{*0} separately. This allows us to retain information that would be lost by using directly the norm of each block.

By construction $\sigma_d(\pi)$ is a discrete approximation of the following continuous lower bound of $\sigma_c([a,b])$:

$$\sigma_c([a,b]) \geq \max_{e_i, 1 \leq i \leq m} \frac{\kappa_{1,c}([a,b], e_i)}{\gamma_{1,c}([a,b], e_i)}.$$

In the following we approximate the discrete value of $\sigma_d(\pi)$ by the lower bound given in (4.3). One of the important advantages of computing this approximation of the discrete conditioning parameters is that it is very inexpensive to implement since it requires only one block column of G to be computed. In what follows we describe how to use this information to compute an estimate of $\|G\|_\infty$ which allows us to obtain a discrete approximation of κ. In this paper we describe the algorithm only for the one step formulae (implemented in TWPBVPLC and TWPBVPC). In what follows we give in detail the algorithm implemented by Higham in [15] for the estimation of the one norm of a matrix. This algorithm is a variant of the original algorithm derived by Hager in [14]. We apply it to the transpose matrix, to compute an approximation of the infinity norm. This algorithm, given $G \in \mathbb{R}^{N \times N}$, computes $\kappa_d(\pi)) \leq \|G\|_\infty$ and $\|Gx\|_\infty = \kappa_d(\pi)\|x\|_\infty$

The flow chart (matlab-like) for doing this is given below (see [16]):

function $\kappa_d(\pi)$ = *KappaHigham(G,N)*
  $\zeta = N^{-1} ones(N,1)$
  $g^r = G^T \zeta$
  $\kappa_d = \|g^r\|_1$
  $\xi = sign(g^r)$
  $z = G\xi$
  $nstep = 2$
  while nstep <= 5
    $\zeta = e_j$, *where* $|z_j| = \|z\|_\infty$ *(smallest j)*

$g^r = G^T\zeta$
$\kappa_d^n = \|g^r\|_1$
if $sign(g^r) = \xi$ *or* $\kappa_d^n(\pi) < \kappa_d(\pi)$, *goto (*), end*
$\kappa_d = \kappa_d^n$
$\xi = sign(g^r)$
$z = G\xi$
$nstep = nstep + 1$
if $|z_j| = \|z\|_\infty$, *break, end*
end
(*) $z_i = (-1)^{i+1}(1 + (i-1)/(N-1)), i = 1 : N$
$z = G^T z$
if $2\|z\|_1/(3N) > \kappa_d(\pi)$ *then*
$g^r = z$
$\kappa_d(\pi) = 2\|z\|_1/(3N)$
end

In the following we present a variant of the Higham algorithm that uses the information given by the conditioning parameters. Since to compute $\kappa_{1,d}(\pi)$ we need the infinity norm of the vector Ω_{*0}, which depends on the first m columns of G, we could easily compute the index j_k in which Ω_{*0} reaches its maximum, that is $\Omega_{j_k 0} = \kappa_{1,d}(\pi) = \max_i \Omega_{i0}$, and the index k such that $\|(G_{j_k,0})_{k*}\|_\infty = \Omega_{j_k 0}$. We call g^r the row of index $i_k = (j_k - 1)m + k$ of the matrix M^{-1}, ($g^r = G^T e_{i_k}$). In the case of separated boundary conditions, and it is these boundary conditions that are allowed in the codes, if the problem has an exponential dichotomy and it is well conditioned, we have that κ_1 gives all the required information about the conditioning, and the Green's function in (3.2) decays exponentially with $|x - t|$ (see [1] p.126). Since for stiff problems Ω_{*0} reaches its maximum in one isolated element we conclude that the 1-norm of g^r is a good approximation to $\|G\|_\infty$. In this case $\kappa_{1,d}(\pi) = \|g^r_{1:m}\|_1$ by construction and $\kappa_{2,d}(\pi) = \|g^r_{m+1:(n+1)m}\|_1$ is an approximation to κ_2. However for non stiff problems Ω_{*0} could have many elements equal to the maximum, and if $\kappa_{2,d}(\pi)$ is of the same order as $\kappa_{1,d}$ its contribution to $\kappa_d(\pi)$ is important. We have modified the Higham algorithm in order to use this information given by the stiffness parameters. The new algorithm is:

function $[\kappa_d(\pi), \kappa_{2,d}(\pi)] =$ *KappaStiffBvp(G,N,m,*i_k*,*$\kappa_{1,d}(\pi)$*,* $\sigma_d(\pi)$*)*
$nstep = 1$
if $\sigma_d(\pi) < 10$
$\zeta = e_{i_k}$
else
$\zeta = ones(N, 1)/(N + 11)$, $\zeta_{i_k} = 11/(N + 11)$
end
$g^r = G^T\zeta$
$\kappa_d(\pi) = \|g^r\|_1$
$\kappa_{2,d}(\pi) = \|g^r_{m+1:N}\|_1$
while $(\sigma_d(\pi) < 10 \mid \kappa_{2,d}(\pi) > \kappa_{1,d}/10)$ & $nstep \le 3$
$\xi = sign(g^r)$
$z = G\xi$
if $\|z\|_\infty \le z^T\zeta$
break
end
$\zeta = e_j$, *where* $|z_j| = \|z\|_\infty$ *(smallest j)*
$g^r = G^T\zeta$
$\kappa_d^n(\pi) = \|g^r\|_1$
if $\kappa_d^n(\pi) < \kappa_d(\pi)$, *break, end*
$\kappa_d(\pi) = \kappa_d^n(\pi)$
$\kappa_{2,d}(\pi) = \|g^r_{m+1:N}\|_1$
$nstep = nstep + 1$
end

We note that a similar modification could be applied to the block algorithm to estimate the one norm presented in [17], we have however explained only the algorithm which is used in the current version of the codes.

Since the matrix that has already been factorized in the codes is of the form $\tilde{M} = DM$, where D is a diagonal matrix with blocks $D_0 = I$, $D_i = h_i I, i = 1, \ldots, N$, where h_i are the gridsizes used (for simplicity we suppose that the boundary conditions are in the first block row), we need to compute an approximation of $\|M^{-1}\|_\infty$, knowing the factorization of $\tilde{M}$. To do this we apply the algorithm previously described for the computation of $\|(\tilde{M}^{-1}D)\|_\infty$.

We have updated the codes TWPBVPC and TWPBVPLC in order to compute $\sigma_d(\pi)$ using the approximation in (4.3), and $\kappa_d(\pi)$ and $\kappa_{2,d}(\pi)$ using the new algorithm *KappaStiffBvp* and these are available on the authors' web page.

5. Hybrid mesh selection strategies based on conditioning. At present there exist just a few codes that implement hybrid mesh selection strategies, and these include TOM, TWPBVPC and TWPBVPLC. We have structured our approach to mesh selection so that the strategies described in this paper are common to all three codes being considered with the only things changing being the values of several variables which need to be chosen heuristically. The common approach is to choose the mesh in order to have a discrete problem with conditioning parameters similar to those of the continuous problem. That is, we need to choose the mesh so that the discrete monitor function used by all three codes, when only the conditioning is taken into account, is:

$$\psi(x_i) = |\Omega_{i0} - \Omega_{i-1,0}| + \alpha \tag{5.1}$$

where $\alpha = \frac{p}{(1-p)(b-a)} \sum_{i=1}^{N} |\Omega_{i0} - \Omega_{i-1,0}|$.

5.1. Mesh selection for Deferred Correction Formulae. The code TWPBVPL is a deferred correction code based on Lobatto IIIa formulae of order 4,6 and 8 [12]. This code is an extension of the one presented in [2], and this is considerably more robust than TWPBVPC, especially for singularly perturbed boundary value problems. For this reason these Lobatto deferred correction algorithms have also been implemented in the code ACDC [10] in a continuation framework, in order to deal with extremely stiff problems. The deferred correction scheme implemented in TWPBVPL has many similarities with that used in TWPBVPC, and as a result the hybrid mesh selection strategy derived in TWPBVPC has been implemented for TWPBVPLC using a similar procedure. The two hybrid strategies have been described in detail in [8, 12]. However since the underlying numerical schemes used in TWPBVPC and TWPBVPLC are very different it is important to set up some empirical parameters in order to have an efficient mesh selection for each code. In what follows we will describe some of these parameters which have been chosen as a result of extensive numerical experimentation. In particular, we report the empirical parameters that have been set up when the codes have been updated using $\sigma_d(\pi)$ in (4.3) for the definition of stiffness and computing $\kappa_d(\pi)$ using the algorithm *KappaStiffBvp* presented in the previous section.

In particular the monitor function is as defined by (5.1), but the parameter $p/(1-p) = 10^{-5}$ is used for Lobatto and $p/(1-p) = 0.08$ is used for TWPBVPC. The three parameters $\kappa_d(\pi)$ $\kappa_{1,d}(\pi)$ and $\gamma_{1,d}(\pi)$ are considered as having become stabilised using the same criterion as was defined in [8] that is if they change by less than 5 % from one mesh to another. A problem is considered stiff if

$$\sigma_d(\pi) > 10. \tag{5.2}$$

This criterion is different from the one presented in the previous version of the code TWPBVPLC, where a problem was considered stiff if $\kappa_1(\pi)/\gamma_1(\pi) > 100$. However the new definition of the stiffness parameter $\sigma_d(\pi)$ in (4.3) allows us to use the same criterion (5.2) for both the codes.

The algorithm for adding and removing points, for stiff problems, is based on the following two quantities associated with the monitor function ψ:

$$r_1 = \max_{i=1,\ldots,n} (\psi(x_i)h_i),$$

and

$$r_2 = \sum_{i=1}^{n} (\psi(x_i)h_i)/N.$$

Here h_i refers to the mesh spacing on the current mesh and n refers to the number of points in the mesh. We decided to add additional mesh points when $\psi(x_i)h_i$ is sufficiently large and in our Lobatto code we have taken this as being when it is greater than $\max(0.65r_1, r_2)$. For TWPBVPC we add additional mesh points when $\psi(x_i)h_i$ is greater than $max(0.5r_1, r_2)$. Note that these two parameters differ because usually the Lobatto schemes give us more reliable information concerning the regions where it is appropriate for points to be added. The number of mesh points to be added depends on the number of intervals in which this relation is satisfied. If the problem is stiff and the conditioning parameters are not stabilized the mesh control procedure puts points in the region of rapid variation of the monitor function and also makes sure that not too many points are added at any stage. If the conditioning parameters are stabilized, we are usually working with a good mesh and, if the error is small, we add points using the estimated local error. The technique for removing points also takes into account the monitor function and we remove points only when $\psi(x_i)h_i$ is less than $10^{-5}r_2$. This parameter has been changed for TWPBVPLC in order to make it more robust for non linear problems.

For nonlinear problems the strategy used in TWPBVPLC and in TWPBVPC remain the same that is, if the Newton iteration scheme does not converge, the partially converged solution is considered stiff if $\sigma_d(\pi) > 10$ in TWPBVPLC whereas in TWPBVPC 10 is replaced by 5.

5.2. Mesh selection for boundary value methods. The code TOM, based on boundary value methods of even order from 2 to 10, is different from the deferred correction codes in two main respects. The first is in the solution of non linear problems, where it uses a quasi linearization procedure that allows the use of the conditioning parameters for each linear problem arising during the quasi linearization. The second is the use of the monitor function that allows us to both move the mesh points and add and remove them. Nevertheless, we tried to keep most of the empirical parameters similar to the codes based on deferred corrections. In particular, the three parameters $\kappa_d(\pi)$ $\kappa_{1,d}(\pi)$ and $\gamma_{1,d}(\pi)$ are considered as having become stabilised using the same criterion as for the deferred correction codes, that is if they change by less than 5% from one mesh to another. A problem is considered stiff when $\sigma_d(\pi) > 100$ and this is different from the criterion (5.2) used in the deferred correction codes and is mainly due to the different numerical schemes used. We set $p/(1-p) = 0.08$ which is exactly the same as was used for TWPBVPC. We decided to add additional mesh points when $\psi(x_i)h_i$ is sufficiently large and in TOM we take this as being when it is greater than $\max(0.65r_1, r_2)$, which again is exactly the same as in the Lobatto code. In TOM we remove points, in the first quasilinearization step, when $\psi(x_i)h_i$ is less than $10^{-3}r_2$ or $10^{-8}r_2$, depending on the order of the method used and on the estimated error in the boundary points. For the subsequent quasilinearization steps we use $10^{-3}r_2$. We note that the quasilinearization algorithm implemented in TOM makes the behavior of this code different from the codes using a damped Newton technique. The reader is referred to [23] for more details concerning the solution of nonlinear problems.

5.3. Estimation of the error. We note that the hybrid mesh selection strategy asks for the conditioning parameters to be stabilized before we are able to use the error or the defect in the mesh selection. This is equivalent to asking that the fundamental matrix $Y(x)$ in (3.2) is well approximated by the numerical method before selecting the mesh. In this case the error estimate would be of interest and the size of the stability constant $\kappa_d(\pi)$ gives us information about the reliability of this estimate. In codes based on deferred correction the error for the order 4 method is estimated using the local truncation error, while for the order 6, and the order 8 methods, the error is estimated using the difference between the computed solution of order 4 and 6, or 6 and 8, respectively. Since the error is usually computed using the relative difference between the solution computed by methods of different orders, it is a good estimate of the global error. The same is true for the code TOM, where the error is estimated by computing an approximate solution of a higher order method using one step of a deferred correction procedure. However, the conditioning parameters give us more information about the error estimation. In fact, if the codes do not give a solution because the input error tolerances are too stringent, the user could change the input parameters accordingly. The codes only give a warning about this, because the input error tolerances are usually different for each component of the solution and cannot be changed automatically.

Additional important information related to the conditioning parameters is whether they are stabilized or not. In particular if $\kappa_d(\pi)$ is not stabilized the exit flag of the codes is -1 and not 0, because we can not assume that the numerical solution is reliable and we may be basing our strategy on non-converged solutions.

6. Numerical Results. In this section we present some numerical results to justify the changes made in the codes TWPBVPLC and TWPBVPC. We do not report on numerical experience with the code TOM

because the Fortran version is not yet available. Numerical results using TOM can be found in the following papers [23, 22, 25, 26]. In the following, to simplify the notation, we denote the discrete conditioning parameters by $\kappa(\pi), \kappa_2(\pi), \kappa_1(\pi), \gamma_1(\pi), \sigma(\pi)$. We have run the codes on a set of 33 singular perturbation boundary value problems (see [11] for a description of the test problems), the first set of numerical results is intended to show the reliability of the infinity norm estimator used in the codes. In Figure 6.1 we report, for each test problem, the mean value of the ratio $\kappa(\pi)/\kappa_H(\pi)$ where $\kappa_H(\pi)$ is the value computed using the Higham algorithm in [15]. The mean value has been computed running the problems for different values of ϵ and for different values of the input tolerances. A value smaller than unity means that the Higham algorithm is on average more accurate; a value higher than unity means that the new estimator is better. The number of computed matrices is 19195, having size ranging from 9 to 34976. The mean value over all the experiments for both codes is 1.139. This value means that the new estimator gives, on average, better results. In Figure 6.1 we give a diagram showing the mean value of the ratio $\kappa(\pi)/\kappa_H(\pi)$ for the 33 problems and for the two codes. We note that for problems 9 and 16 the new estimator computes a better lower bound. For the other problems the results are similar. However it is interesting to see that the minimum value of the mean ratio is 0.8 (computed when solving problem 27 with $\epsilon = 10^{-5}$ and $tol = 10^{-6}$ using TWPBVPC), the maximum value of the mean ratio is 84.3 (computed when solving problem 9 with $\epsilon = 10^{-6}$ and $tol = 10^{-4}$ using TWPBVPC).

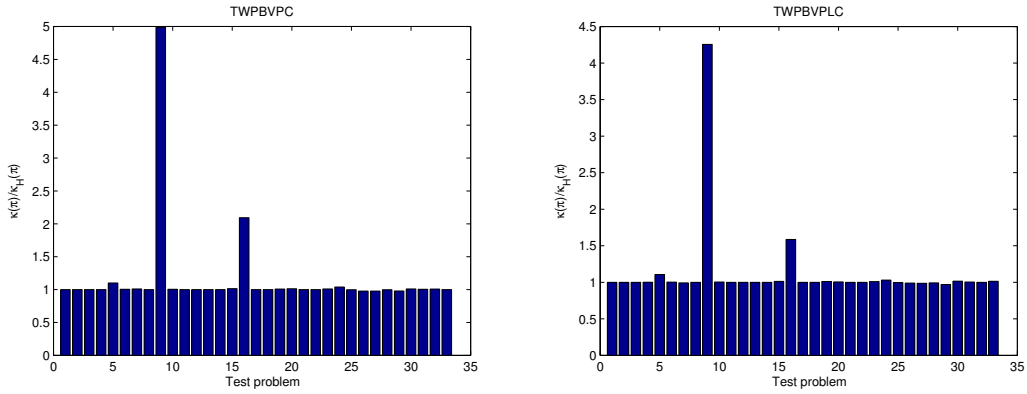


FIG. 6.1. *Mean value of the ratio $\kappa(\pi)/\kappa_h(\pi)$ for each test problem solved with different values of ϵ and different values of tol.*

In Figure 6.2 we report, for the problems 1 and 15, the ratio $\kappa_1(\pi)/\gamma_1(\pi)$ compared with the new value of $\sigma(\pi)$. We note that for these problems $\sigma(\pi)$ is a more reliable estimate of the stiffness of the problem. In general, however, the difference between the two stiffness estimators is minimal.

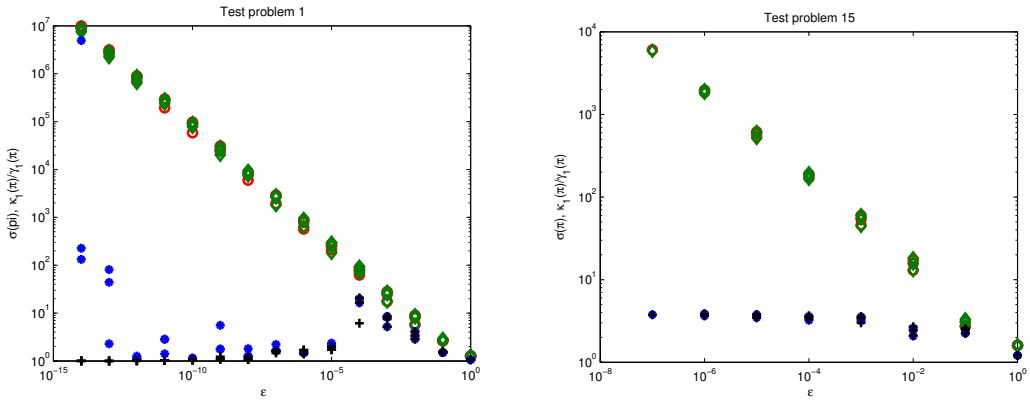


FIG. 6.2. *Circle (TWPBVPC) and diamond (TWPBVPLC) are the values of $\sigma(\pi)$ computed with different values of the input tolerances, asterisks (TWPBVPC) and plus (TWPBVPLC) are the corresponding values of $\kappa_1(\pi)/\gamma_1(\pi)$.*

The behavior of the codes using the conditioning parameters is more or less the same as for the previous versions. We refer to [8, 12], where many experiments are reported, for a comparison to the standard mesh

selection strategy with the one using conditioning. Nevertheless there are some improvements on certain difficult problems that are explained below.

We note that when $\kappa_2(\pi)$ is very small the standard mesh selection strategy and the one based on conditioning gives very similar results, and in many cases the standard mesh selection works better. This is explained by the fact that, since $\kappa_2(\pi)$ is small, the local truncation error is a good approximation of the global error. In the following we report only the results for three problems, the first two are problems for which $\kappa_2(\pi)$ is of the same size as $\kappa_1(\pi)$, the third problem is one that has 2, 1 or 0 solutions depending on the value of a certain parameter appearing in the differential equation. For all the problems we set $tol(ncomp_1) = tol(ncomp_2) = tol$ as input. The number of the problem is the same as reported in [11].

Problem 6

$$\epsilon y'' + xy' = -\epsilon\pi^2\cos(\pi x) - \pi x\sin(\pi x), \qquad y(-1) = -2,\ y(1) = 0.$$

This problem is a linear singularly perturbed problem. It has been chosen because, for $0 < \epsilon \ll 1$, the solution has a turning point at $x = 0$. The exact solution is $y(x) = \cos(\pi x) + \exp((x-1)/\sqrt{\epsilon}) + \exp(-(x+1)/\sqrt{\epsilon})$.

Problem 19

$$\epsilon y'' + \exp(y)y' - \frac{\pi}{2}\sin(\frac{\pi x}{2})\exp(2y) = 0, \quad y(0) = 0, y(1) = 0.$$

This problem has a boundary layer at $x = 0$. We use as initial guess 0 for y and y'.

Problem 34

$$y'' + \lambda exp(y) = 0, \quad y(0) = y(1) = 0.$$

This example is Bratu's problem, which is considered by Shampine and Muir [28]. Setting $\lambda^* = 3.51383\ldots$ it is known that if $0 \le \lambda < \lambda^*$ the problem has two solutions, for $\lambda = \lambda^*$ it has one solution and for $\lambda > \lambda^*$ there is no solution. We use as initial guess 0 for y and y'.

6.1. Description of the results. In Figure 6.3 we report, for problems 6 and 19, the maximum mesh used by the two codes with and without using the conditioning parameters in the mesh selection. For simplicity we add in brackets to the name of the code the term ON or OFF that indicates if the conditioning parameters have been used or not in the mesh selection (TWPBVPC(ON) means that the conditioning has been used) . The problems were solved for different values of ϵ and $tol = 10^{-4}, 10^{-6}, 10^{-8}$. In the diagrams, circles and diamonds are related to TWPBVPC(ON) and TWPBVPLC(ON) respectively, asterisks and plus are related to TWPBVPC(OFF) and TWPBVPLC(OFF) respectively (if the code was not able to find a solution the corresponding symbol is not reported). We note that for these problems the conditioning parameters usually allow us to obtain the solution with a smaller number of mesh points than when conditioning is not used. For both problems TWPBVPC is able to obtain the solution when the other algorithm fails; TWPBVPLC has no problem with the linear example, but for the nonlinear example the use of the conditioning allows us to compute the solution for smaller values of ϵ.

Figure 6.4 plots the condition numbers $\kappa(\pi)$ and $\kappa_1(\pi)$ for a range of values of ϵ, and Figure 6.5 plots the value of $\sigma(\pi)$ and the ratio $\kappa_1(\pi)/\gamma_1(\pi)$. For these problems we note that $\kappa_1(\pi)/\gamma_1(\pi) \approx \sigma(\pi)$, and the condition number $\kappa(\pi) \approx 2\kappa_1(\pi)$ grows like $\sqrt{(}1/\epsilon)$ for problem 6 and like $1/\epsilon$ for problem 19. The stiffness ratio $\sigma(\pi)$ has the same behavior (see 6.5).

The third example is taken from [28] where the authors solved the problem by using the MATLAB code BVP4C with $\lambda = 3.45$ for which there are two solutions. For this problem a solution was computed in a perfectly satisfactory way and the condition number was estimated to be 3400. The same procedure was carried out again with $\lambda = 3.55$ (for which there is no solution) and the MATLAB code produced what looked like a perfectly satisfactory solution with a conditioning constant estimated to be 10^6. This very big conditioning constant warns that the 'solution' may have no correct digits and this is indeed the case. Further experiments are reported in [28] where it is emphasised that we need to look at the conditioning of a problem before accepting a solution. The approach taken by Shampine and Muir is to warn the user of a very large condition number if it exists. We note that the condition number computed in [28] is the infinity norm of a scaled matrix W_1GW_2 where the diagonal scaling matrices are related to the scaling factors used in the code to compute the residual. We compute, instead, a condition number related to the problem and some important information computed by the code is whether the estimates of the conditioning parameters stabilised or not. In Table 6.1 we report

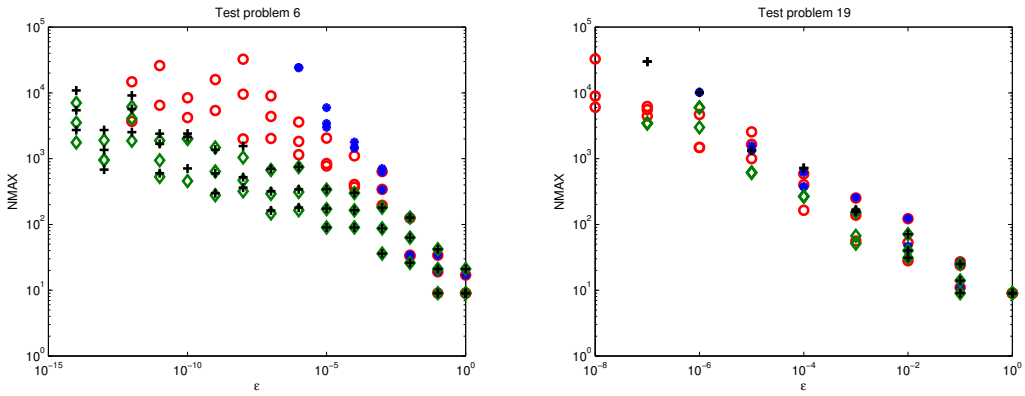


FIG. 6.3. *Circle (TWPBVPC(ON)), asterisks (TWPBVPC(OFF)), diamond (TWPBVPLC(ON)) and plus (TWPBVPLC(OFF))) are the values of maximum mesh used by the codes with different values of tol and of* ϵ.

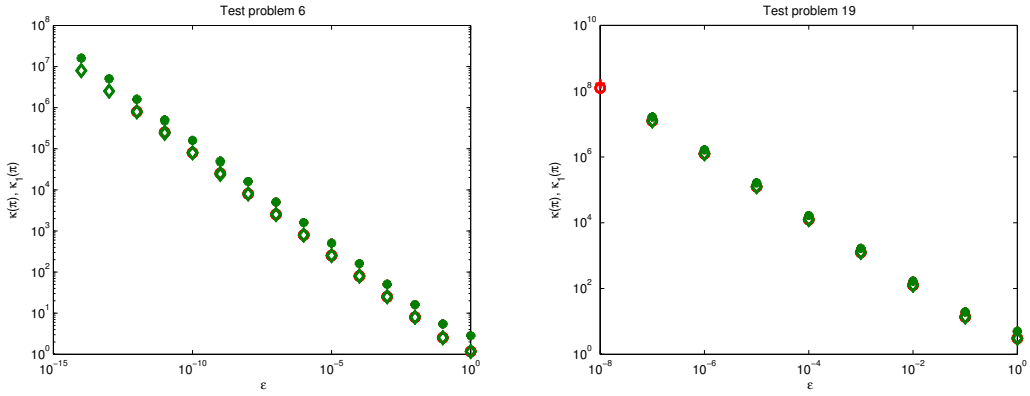


FIG. 6.4. *Circle (TWPBVPC) and diamond (TWPBVPLC) are the values of* $\kappa_1(\pi)$, *asterisks (TWPBVPC) and plus (TWPBVPLC) are the values of* $\kappa(\pi)$.

the results obtained by the codes TWPBVPC and TWPBVPLC solving the problem with different values of λ and $tol = 10^{-3}$, $tol = 10^{-6}$ and $tol = 10^{-9}$, using an initial mesh with $N_S + 1$ mesh points. To be sure that the conditioning parameters obtained are stabilized we checked the value by running the code using a mesh doubled with respect to the final one and where we do not perform any grid refinement (the final mesh has $N_F + 1$ points). In Table 6.1 we set the value of $STAB_C$ equal to T if for the doubled mesh the value of $\kappa(\pi)$ differs by less than 5 % from the value computed with the mesh $N_F + 1$, $STAB_C$ is set to F otherwise.

Only for $\lambda = 3.5, 3.51, 3.513$ do the codes give an output flag with a value of 0 and $STAB_C$= T, for all the other values we have that the flag is -1. But in no cases do we accept a computed 'solution' when none exist. If we start the code with a starting mesh with a larger value of N_S, we see that for both the codes the value of $STAB_C$ is T when $\lambda <= 3.51383$. For $\lambda > 3.51383$, however, the number of mesh points in the final mesh can be very high compared with the one used for $\lambda \leq 3.51383$. In particular TWPBVPC fails to give a solution for $\lambda \geq 3.513832$, TWPBVPLC fails for $\lambda \geq 3.513833$, and for the solution given with $\lambda = 3.513832$ the $STAB_C$ flag is always F.

7. Conclusion. In this paper we have been concerned with the numerical solution of singularly perturbed boundary value problems using variable grid integration methods. We have established a fundamental approach to choosing our meshes, defining sequences of meshes so that the continuous and discrete problems have the same conditioning. We do this by developing a monitor function which depends both on local accuracy and conditioning. We have presented new techniques to compute the conditioning parameters and we have emphasised that the approach adopted is very similar for all three codes: TOM, TWPBVPC and TWPBVPLC. The main difference is in how we choose the different values of a few heuristic parameters and we have explained in

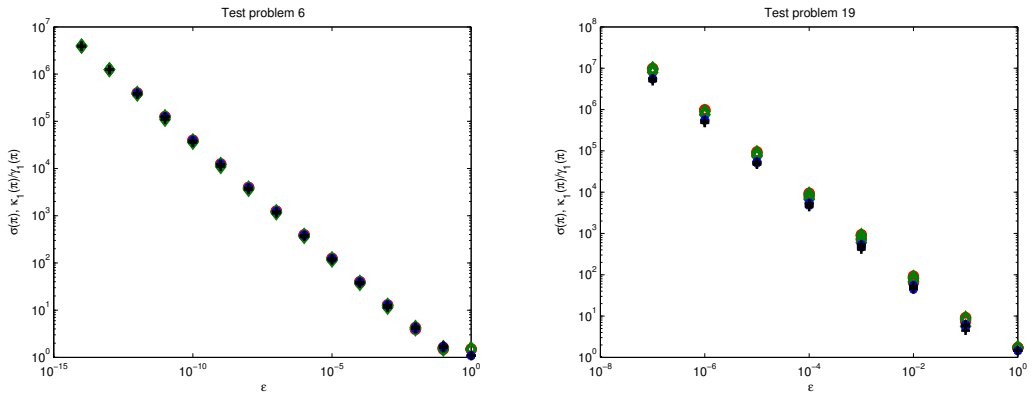


FIG. 6.5. *Circle (TWPBVPC) and diamond (TWPBVPLC) are the values of $\sigma(\pi)$, asterisks (TWPBVPC) and plus (TWPBVPLC) are the corresponding values of $\kappa_1(\pi)/\gamma_1(\pi)$.*

some detail how this is done. Our codes can be considerably more efficient than other codes, due to the fact that we pay attention to the conditioning of the problem, and we have never accepted a 'solution' when none exists. However, for some problems, our conditioning parameters may converge rather slowly and we intend to examine this in detail in a future paper.

REFERENCES

[1] U. M. Ascher, R. M. M. Mattheij and R. D. Russell. *Numerical solution of boundary value problems for ordinary differential equations*, Classics in Applied Mathematics **13**. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1995. Corrected reprint of the 1988 original.

[2] Z. Bashir-Ali, J. R. Cash and H. H. M. Silva. Lobatto deferred correction for stiff two-point boundary value problems. *Comput. Math. Appl.*, **36**, no. 10-12, 59–69, 1998. Advances in difference equations, II.

[3] L. Brugnano and D. Trigiante. Tridiagonal matrices: invertibility and conditioning. *Linear Algebra Appl.*, **166**, 131–150, 1992.

[4] L. Brugnano and D. Trigiante. On the characterization of stiffness for ODEs. *Dynam. Contin. Discrete Impuls. Systems*, **2**, no. 3, 317–335, 1996.

[5] L. Brugnano and D. Trigiante. A new mesh selection strategy for ODEs. *Appl. Numer. Math.*, **24**, no. 1, 1–21, 1997.

[6] L. Brugnano and D. Trigiante. *Solving Differential Problems by Multistep Initial and Boundary Value Methods*. Gordon & Breach,Amsterdam, 1998.

[7] J. R. Cash. On the numerical integration of nonlinear two-point boundary value problems using iterated deferred corrections. II. The development and analysis of highly stable deferred correction formulae. *SIAM J. Numer. Anal.*, **25**, no. 4, 862–882, 1988.

[8] J. R. Cash and F. Mazzia. A new mesh selection algorithm, based on conditioning, for two-point boundary value codes. *J. Comput. Appl. Math.*, **184**, no. 2, 362–381, 2005.

[9] J. R. Cash, F. Mazzia, N. Sumarti and D. Trigiante. The role of conditioning in mesh selection algorithms for first order systems of linear two point boundary value problems. *J. Comput. Appl. Math.*, **185**, no. 2, 212–224, 2006.

[10] J. R. Cash, G. Moore and R. Wright. An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems. *ACM Trans. Math. Software*, **27**, no. 2, 245–266, 2001.

[11] J. R. Cash and F. Mazzia. Algorithms for the solution of two-point boundary value problems. `http://www.ma.ic.ac.uk/~jcash/BVP_software/twpbvp.php`.

[12] J. R. Cash and F. Mazzia. Hybrid mesh selection algorithms based on conditioning for two-point boundary value problems. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, **1**, no. 1, 81–90, 2006.

[13] C. de Boor and H.-O. Kreiss. On the condition of the linear systems associated with discretized BVPs of ODEs. *SIAM J. Numer. Anal.*, **23**, no. 5, 936–939, 1986.

[14] W. W. Hager. Condition estimates. *SIAM J. Sci. Statist. Comput.*, **5**, no. 2, 311–316, 1984.

[15] N. J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Software*, **14**, no. 4, 381–396 (1989), 1988.

[16] N. J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.

[17] N. J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, **21**, no. 4, 1185–1201 (electronic), 2000.

[18] F. Iavernaro, F. Mazzia and D. Trigiante. Stability and conditioning in numerical analysis. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, **1**, no. 1, 91–112, 2006.

[19] P. Kunkel and V. Mehrmann. *Differential-algebraic equations,Analysis and numerical solution*. EMS Textbooks in Mathematics. European Mathematical Society (EMS), Zürich, 2006.

TABLE 6.1

Conditioning parameters, flags, number of mesh points in the initial constant mesh and number of mesh points in the final mesh for Problem 34

				TWPBVPC(ON)					
λ	tol	$\kappa(\pi)$	$\kappa_1(\pi)$	$\gamma_1(\pi)$	$\sigma(\pi)$	FL	N_S+1	$STAB_C$	N+1
3.5	10^{-3}	0.534E+02	0.366E+02	0.289E+02	0.130E+01	0	10	T	10
	10^{-6}	0.537E+02	0.368E+02	0.277E+02	0.136E+01	0	10	T	18
	10^{-9}	0.538E+02	0.368E+02	0.275E+02	0.137E+01	0	10	T	21
3.51	10^{-3}	0.102E+03	0.701E+02	0.556E+02	0.128E+01	0	10	T	10
	10^{-6}	0.104E+03	0.712E+02	0.539E+02	0.134E+01	0	10	T	17
	10^{-9}	0.104E+03	0.712E+02	0.535E+02	0.135E+01	0	10	T	21
3.513	10^{-3}	0.209E+03	0.144E+03	0.114E+03	0.126E+01	0	10	F	10
	10^{-6}	0.224E+03	0.154E+03	0.117E+03	0.132E+01	-1	10	T	19
	10^{-9}	0.224E+03	0.154E+03	0.115E+03	0.134E+01	-1	10	T	24
3.5138	10^{-3}	0.105E+04	0.720E+03	0.549E+03	0.131E+01	-1	10	F	19
	10^{-6}	0.105E+04	0.720E+03	0.549E+03	0.131E+01	-1	10	F	19
	10^{-9}	0.105E+04	0.720E+03	0.549E+03	0.131E+01	-1	10	F	19
	10^{-9}	0.117E+04	0.801E+03	0.593E+03	0.135E+01	0	40	T	40
3.51383	10^{-3}	0.214E+04	0.147E+04	0.112E+04	0.131E+01	-1	10	F	19
	10^{-6}	0.537E+04	0.369E+04	0.274E+04	0.135E+01	-1	10	F	37
	10^{-9}	0.537E+04	0.369E+04	0.274E+04	0.135E+01	-1	10	F	37
	10^{-9}	0.721E+04	0.495E+04	0.363E+04	0.136E+01	0	80	T	80
3.513831	10^{-3}	0.225E+04	0.155E+04	0.118E+04	0.131E+01	-1	10	F	19
	10^{-6}	0.120E+05	0.823E+04	0.612E+04	0.135E+01	-1	10	F	37
	10^{-9}	0.120E+05	0.823E+04	0.612E+04	0.135E+01	-1	10	F	37
	10^{-9}	0.773E+04	0.531E+04	0.385E+04	0.138E+01	-1	80	T	10113
3.513832	10^{-3}	*							
	10^{-6}	*							
	10^{-9}	*							
				TWPBVPLC(ON)					
3.5	10^{-3}	0.534D+02	0.366D+02	0.289D+02	0.130D+01	0	10	T	10
	10^{-6}	0.534D+02	0.366D+02	0.289D+02	0.130D+01	0	10	T	10
	10^{-9}	0.538D+02	0.368D+02	0.276D+02	0.137D+01	0	10	T	21
3.51	10^{-3}	0.102D+03	0.701D+02	0.556D+02	0.128D+01	0	10	T	10
	10^{-6}	0.102D+03	0.701D+02	0.556D+02	0.128D+01	0	10	T	10
	10^{-9}	0.104D+03	0.712D+02	0.535D+02	0.135D+01	0	10	T	22
3.513	10^{-3}	0.209D+03	0.144D+03	0.114D+03	0.126D+01	0	10	F	10
	10^{-6}	0.209D+03	0.144D+03	0.114D+03	0.126D+01	0	10	F	10
	10^{-9}	0.154D+03	0.116D+03	0.224D+03	0.133D+01	-1	10	T	21
3.5138	10^{-3}	0.102D+04	0.700D+03	0.533D+03	0.131D+01	-1	10	F	19
	10^{-6}	0.102D+04	0.700D+03	0.533D+03	0.131D+01	-1	10	F	19
	10^{-9}	0.102D+04	0.700D+03	0.533D+03	0.131D+01	-1	10	F	19
	10^{-9}	0.117E+04	0.801E+03	0.593E+03	0.135E+01	0	40	T	40
3.51383	10^{-3}	0.214D+04	0.147D+04	0.112D+04	0.131D+01	-1	10	F	19
	10^{-6}	0.537D+04	0.369D+04	0.274D+04	0.135D+01	-1	10	T	37
	10^{-9}	0.537D+04	0.369D+04	0.274D+04	0.135D+01	-1	10	T	37
3.513831	10^{-3}	0.225D+04	0.155D+04	0.118D+04	0.131D+01	-1	10	F	19
	10^{-6}	0.354D+04	0.244D+04	0.176D+04	0.138D+01	-1	10	T	9217
	10^{-9}	0.354D+04	0.244D+04	0.176D+04	0.138D+01	-1	10	T	9217
3.513832	10^{-3}	0.238D+04	0.164D+04	0.125D+04	0.131D+01	-1	10	F	19
	10^{-6}	0.146D+05	0.101D+05	0.729D+04	0.138D+01	-1	10	F	9217
	10^{-9}	0.146D+05	0.101D+05	0.729D+04	0.138D+01	-1	10	F	9217
	10^{-9}	0.146D+05	0.101D+05	0.729D+04	0.138D+01	-1	20	F	9729
	10^{-9}	*	*	*	*	-1	40	*	*
3.513833	10^{-3}	*							
	10^{-6}	*							
	10^{-9}	*							

[20] R. M. M. Mattheij and G. W. M. Staarink. An efficient algorithm for solving general linear two-point BVP. *SIAM J. Sci. Statist. Comput.*, **5**, no. 4, 745–763, 1984.

[21] R. M. M. Mattheij and G. W. M. Staarink. On optimal shooting intervals. *Math. Comp.*, **42**, no. 165, 25–40, 1984.

[22] F. Mazzia, A. Sestini and D. Trigiante. The continous extension of the B-spline linear multistep metods for BVPs on non-uniform meshes. *Appl. Numer. Math.*, **59**, no. 3-4, 723–738, 2009.

[23] F. Mazzia and D. Trigiante. Efficient strategies for solving nonlinear problems in bvps codes. *Nonlinear Studies*. In press.

[24] F. Mazzia and D. Trigiante. Numerical solution of singular perturbation problems. *Calcolo*, **30**, no. 4, 355–369, 1993.

[25] F. Mazzia, A. Sestini and D. Trigiante. B-spline linear multistep methods and their continuous extensions. *SIAM J. Numer. Anal.*, **44**, no. 5, 1954–1973 (electronic), 2006.
[26] F. Mazzia, A. Sestini and D. Trigiante. BS linear multistep methods on non-uniform meshes. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, **1**, no. 1, 131–144, 2006.
[27] F. Mazzia and D. Trigiante. A hybrid mesh selection strategy based on conditioning for boundary value ODE problems. *Numer. Algorithms*, **36**, no. 2, 169–187, 2004.
[28] L. F. Shampine and P. H. Muir. Estimating conditioning of BVPs for ODEs. *Math. Comput. Modelling*, **40**, no. 11-12, 1309–1321, 2004.

Edited by: Pierluigi Amodio and Luigi Brugnano
Received: April 9, 2009
Accepted: October 7, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 363–372. http://www.scpe.org
ISSN 1895-1767

PRECONDITIONING OF IMPLICIT RUNGE-KUTTA METHODS*

LAURENT O. JAY†

Abstract. A major problem in obtaining an efficient implementation of fully implicit Runge-Kutta (IRK) methods applied to systems of differential equations is to solve the underlying systems of nonlinear equations. Their solution is usually obtained by application of modified Newton iterations with an approximate Jacobian matrix. The systems of linear equations of the modified Newton method can actually be solved approximately with a preconditioned linear iterative method. In this article we present a truly parallelizable preconditioner to the approximate Jacobian matrix. Its decomposition cost for a sequential or parallel implementation can be made equivalent to the cost corresponding to the implicit Euler method. The application of the preconditioner to a vector consists of three steps: two steps involve the solution of a linear system with the same block-diagonal matrix and one step involves a matrix-vector product. The preconditioner is asymptotically correct for the Dahlquist test equation. Some free parameters of the preconditioner can be determined in order to optimize certain properties of the preconditioned approximate Jacobian matrix.

Key words: GMRES, implicit Runge-Kutta methods, inexact modified Newton iterations, linear iterative methods, nonlinear equations, ordinary differential equations, parallelism, preconditioning, stiffness

1. Introduction. We consider the numerical solution of implicit systems of differential equations by fully implicit Runge-Kutta (IRK) methods. A major problem in obtaining an efficient implementation of IRK methods lies in the numerical solution of the underlying systems of nonlinear equations. These nonlinear equations are usually solved by modified Newton iterations. This requires at each iteration the solution of a system of linear equations with an approximate Jacobian matrix. Instead of being solved exactly, these systems of linear equations can actually be solved approximately and iteratively with the help of a preconditioner to the approximate Jacobian matrix. In this paper we present a new preconditioner whose decomposition cost for a sequential or parallel implementation can be made equivalent to the cost corresponding to the implicit Euler method. Therefore, this preconditioner is of interest for both sequential and parallel computers. The application of the preconditioner to a vector consists of three steps: two involving the solution of a linear system with the same block-diagonal matrix and one involving a matrix-vector product. Each of these steps can be executed in parallel with some communication in-between. Not only the decomposition of the block-diagonal matrix can be executed in parallel, but of course also the solution of the linear systems corresponding to each block. This makes this preconditioner truly parallel compared to the one proposed in [23, 25]. The preconditioner is asymptotically correct for the Dahlquist test equation. Moreover, some free parameters of the preconditioner can be determined in order to optimize certain properties of the preconditioned approximate Jacobian matrix in relation, for example, with the convergence properties of the preconditioned linear iterative method. In this paper we consider exclusively initial value problems for ordinary differential equations (ODEs). Parallel algorithms for boundary value problems of ODEs can be found for example in [13, 27].

In section 2, the class of implicit systems of ODEs considered in this article is presented together with a definition of the application of IRK methods to these equations. In section 3, we motivate the use of inexact modified Newton iterations to solve the systems of nonlinear equations of IRK methods. A detailed presentation of the new preconditioner is given in section 4. In section 5, the free parameters of the preconditioned Jacobian matrix corresponding to the Dahlquist test equation are optimized with respect to various criteria. In section 6, we present a numerical experiment for a reaction-diffusion problem, showing that the new preconditioner is effective. A short conclusion is given in section 7.

2. The implicit system of ODEs and IRK methods. We consider an implicit system of ordinary differential equations (ODEs)

$$\frac{d}{dt}a(t,y) = f(t,y), \tag{2.1}$$

*This material is based upon work supported by the National Science Foundation under Grant No. 0654044.
†Department of Mathematics, 14 MacLean Hall, The University of Iowa, Iowa City, IA 52242-1419, USA. E-mail: ljay@math.uiowa.edu and na.ljay@na-net.ornl.gov.

where $y = (y^1, \ldots, y^n)^T \in \mathbb{R}^n$. When $a(t,y) \equiv y$ we obtain a standard system of ODEs $\frac{d}{dt}y = f(t,y)$. We suppose that an initial value y_0 at t_0 is given and we assume that

$$a_y(t,y) := \frac{\partial}{\partial y} a(t,y) \quad \text{is invertible} \tag{2.2}$$

in a neighborhood of the solution of the initial value problem. Applying the chain rule to the left-hand side of (2.1) and then inverting $a_y(t,y)$, we obtain an explicit system of ODEs

$$\frac{d}{dt}y = a_y^{-1}(t,y)\left(f(t,y) - a_t(t,y)\right), \tag{2.3}$$

where $a_t(t,y) := \frac{\partial}{\partial t} a(t,y)$. In this paper we consider IRK methods applied directly to (2.1), not to (2.3). This has the advantage of not requiring the exact computation of $a_t(t,y)$ and $a_y^{-1}(t,y)$. It is assumed that the system of ODEs presents some stiffness, thus justifying the application of implicit methods.

The direct application of IRK methods to the implicit system of ODEs (2.1) can be defined as follows [23]:

DEFINITION 2.1. *One step $y_0 \mapsto y_1$ from t_0 to $t_0 + h$ of an s-stage* implicit Runge-Kutta (IRK) method *applied to (2.1) with initial value y_0 at t_0 and stepsize h is defined implicitly by*

$$a(t_0 + c_i h, Y_i) - \Big(a(t_0, y_0) + h \sum_{j=1}^{s} a_{ij} f(t_0 + c_j h, Y_j)\Big) = 0 \quad \text{for } i = 1, \ldots, s, \tag{2.4a}$$

$$a(t_0 + h, y_1) - \Big(a(t_0, y_0) + h \sum_{j=1}^{s} b_j f(t_0 + c_j h, Y_j)\Big) = 0. \tag{2.4b}$$

The RK coefficients are given by $(b_j)_{j=1,\ldots,s}$, $(c_j)_{j=1,\ldots,s}$, and $A := (a_{ij})_{i,j=1,\ldots,s}$. The equations (2.4a) define a nonlinear system of dimension $s \cdot n$ to be solved for the s *internal stages* Y_i for $i = 1, \ldots, s$. The numerical approximation y_1 at $t_0 + h$ is then given by the solution of the n-dimensional implicit system (2.4b). In this paper we concentrate the discussion on the solution of the system of nonlinear equations (2.4a). The value y_1 can then be obtained by standard application of modified Newton iterations to (2.4b) when $a(t,y)$ is nonlinear. It can also be obtained explicitly in certain situations. For example when the IRK method is *stiffly accurate*, i. e., when $a_{sj} = b_j$ for $j = 1, \ldots, s$, this value is directly given by $y_1 = Y_s$.

3. Modified Newton iterations for the internal stages. The system of nonlinear equations (2.4a) for the s internal stages is usually solved by modified Newton iterations with approximate/modified Jacobian matrix

$$L := I_s \otimes M - hA \otimes J \quad \text{where} \quad M \approx a_y(t_0, y_0), \quad J \approx f_y(t_0, y_0). \tag{3.1}$$

Here, the symbol $\otimes$ denotes the matrix tensor product and I_s is the identity matrix in $\mathbb{R}^s$. Modified Newton iterations read

$$L\Delta Y^k = -F(Y^k), \qquad Y^{k+1} = Y^k + \Delta Y^k, \qquad k = 0, 1, 2, \ldots, \tag{3.2}$$

where $Y := (Y_1^T, \ldots, Y_s^T)^T$ is a vector collecting the s internal stages and $F(Y)$ corresponds to the left-hand side of (2.4a). Hence, each modified Newton iteration requires the solution of an $(s \cdot n)$-dimensional system of linear equations. A direct decomposition of the approximate Jacobian matrix L is generally inefficient when $s \geq 2$. However, the computational cost of its decomposition can be greatly reduced by exploiting its special structure. For example by diagonalizing the RK coefficient matrix A

$$SAS^{-1} = \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_s),$$

the approximate Jacobian matrix L can be transformed into a block-diagonal matrix

$$(S \otimes I_n) L (S^{-1} \otimes I_n) = I_s \otimes M - h\Lambda \otimes J = \begin{pmatrix} M - h\lambda_1 J & & O \\ & \ddots & \\ O & & M - h\lambda_s J \end{pmatrix}. \tag{3.3}$$

This kind of transformation can dramatically reduce the number of arithmetic operations when decomposing L and it allows for parallelism. Unfortunately, all eigenvalues (except one when s is odd) of most standard IRK methods, such as Radau IIA, Gauss, and Lobatto IIIC methods, arise as conjugate complex eigenpairs with nonzero imaginary parts. This impairs parallelism and significantly increases the decomposition cost of the transformed approximate Jacobian matrix (3.3) compared to the situation where all eigenvalues are real [20, Section IV.8]. Moreover, if several distinct IRK methods are used in a partitioned/additive way, such as for SPARK methods [22, 24, 32], this diagonalization technique cannot be applied since distinct RK matrices have in general distinct eigenvectors. Ideally, the decomposition cost of the approximate Jacobian matrix for s-stage IRK methods should be reduced to at most s independent decompositions of matrices of dimension n.

Various iteration schemes to solve the nonlinear equations of IRK methods have been suggested [9, 10, 15, 16, 17, 18, 21, 33]. The schemes of [9, 21, 33] can be interpreted as the direct application of modified Newton iterations with a modified Jacobian. Actually it could certainly be valuable to use such modified Jacobians as preconditioners to the system of linear equations of the modified Newton method (3.2) with approximate Jacobian matrix (3.1). The schemes of [10, 15, 16, 17, 18] do not have the same interpretation. They have an inherent sequential structure and can be interpreted as nonlinear Gauss-Seidel iterations. All aforementioned schemes require the solution of a sequence of s linear systems of dimension n and are therefore not truly fully parallelizable compared to the iterations developed in this paper. The various coefficients introduced in these *ad hoc* iterations are generally tuned for the numerical IRK solution y_1 (2.4) of the Dahlquist test equation

$$y' = \lambda y, \qquad \mathrm{R}e(\lambda) \leq 0. \tag{3.4}$$

Unfortunately, when $\mathrm{R}e(\lambda) \to -\infty$, none of the aforementioned methods is asymptotically correct for the internal stages Y_i. In contrast, the preconditioner introduced in this paper is asymptotically correct by construction, as is the one presented in [23, 25].

To reduce the amount of computations, instead of solving at each modified Newton iteration (3.2) the linear system exactly, we can solve it approximately by application of a preconditioned linear iterative method, as was already proposed in [23, 25]. Hence, we obtain a sequence of iterates $\widetilde{Y}^k$ with a residual error $r_k := L\Delta\widetilde{Y}^k + F(\widetilde{Y}^k)$ after each iteration. Theoretical and practical conditions to ensure convergence of such *inexact modified Newton iterations* to the solution of the nonlinear system of equations (2.4a) are given in [23] in a general framework. The use of linear iterative methods for the solution of implicit integration methods was also considered in [2, 8, 12], with an emphasis on preconditioning in [3]. Inexact Newton-type methods are generally considered to be amongst the most efficient ways to solve nonlinear system of equations [11, 29].

In this paper we present a preconditioner to the approximate Jacobian matrix (3.1) requiring s independent decompositions of matrices of dimension n. Each matrix to be decomposed depends on a free and distinct parameter. If these parameters are all chosen to be equal, only one matrix of dimension n needs to be decomposed. This is of high interest when using serial computers since the decomposition cost is thus really minimal and equivalent to the decomposition cost corresponding to the implicit Euler method. In fact a major interest of this new preconditioner is that not only matrix decompositions can be executed in parallel, but also the solution of the systems of linear equations. This makes this preconditioner truly parallel compared to the one developed in [23, 25] which entails the sequential solution of systems of linear equations and which is based on the W-transformation of the RK coefficients and an approximate block-LU decomposition.

4. Preconditioning the linear systems. For the sake of generality we consider a linear transformation $T \otimes I$ of the approximate Jacobian matrix L in (3.1). At each modified Newton iteration we obtain a system of linear equations

$$Kx = b \tag{4.1a}$$

with matrix

$$K = (T \otimes I)L(T^{-1} \otimes I) = I \otimes M - hTAT^{-1} \otimes J. \tag{4.1b}$$

The matrix T adds some potential additional freedom. To solve the linear system (4.1) we consider the application of linear iterative methods, such as GMRES [14, 19, 26, 30, 31], with a preconditioner $Q \approx K^{-1}$. We take Q of the form

$$Q := H^{-1}GH^{-1},$$

where

$$H := I_s \otimes M - h\Gamma \otimes J, \qquad G := I_s \otimes M - h\Omega \otimes J, \tag{4.2}$$

with coefficients matrices Γ and Ω still to be determined. We choose the coefficients matrix Γ to be diagonal

$$\Gamma := \operatorname{diag}(\gamma_1, \gamma_2, \ldots, \gamma_s)$$

so that H is block-diagonal

$$H = \begin{pmatrix} H_1 & & & O \\ & H_2 & & \\ & & \ddots & \\ O & & & H_s \end{pmatrix} \tag{4.3}$$

with blocks given by

$$H_i := M - h\gamma_i J \quad \text{for } i = 1, \ldots, s.$$

The matrices H_i are independent, hence they can be decomposed in parallel. Solving a linear system with matrix H can also be done in parallel since it is block-diagonal. This is the main advantage of this preconditioner compared to the one presented in [23, 25]. Assuming at least s processors on a parallel computer, the local cost on the ith processor of computing the matrix-vector product QKv consists essentially of:

- one decomposition of matrix H_i;
- solving two linear systems with the decomposed matrix H_i;
- two matrix-vector products with matrix M;
- two matrix-vector products with matrix J;
- some communication with other processors according to the nonzero coefficients of the ith row of matrices TAT^{-1} and Ω.

The coefficients of matrices Γ and Ω remain to be fixed to some values. Assuming the coefficients γ_i for $i = 1, \ldots, s$ to be given, it is natural to determine the coefficients Ω_{ij} for $i, j = 1, \ldots, s$ such that the preconditioner Q is asymptotically correct when considering the Dahlquist test equation (3.4). Denoting $z := h\lambda$ we obtain

$$Q(z) = H^{-1}(z)G(z)H^{-1}(z), \qquad K(z) = T(I_s - zA)T^{-1} \tag{4.4a}$$

where

$$H(z) = I_s - z\Gamma, \qquad G(z) = I_s - z\Omega. \tag{4.4b}$$

Defining $B(z) := Q(z)K(z)$ we have

$$B(z) = H^{-1}(z)G(z)H^{-1}(z)T(I_s - zA)T^{-1} \tag{4.5}$$

At $z = 0$ we have $B(0) = I_s$. We determine Ω such that

$$B(z) \longrightarrow I_s \quad \text{for} \quad |z| \longrightarrow \infty \tag{4.6}$$

to obtain an asymptotically correct result in one iteration in this case. From (4.5) we easily obtain the following result.

THEOREM 4.1. *If the RK matrix A is invertible and the coefficients γ_i for $i = 1, \ldots, s$ satisfy $\gamma_i > 0$ then condition (4.6) holds if and only if*

$$\Omega = \Gamma T A^{-1} T^{-1} \Gamma. \tag{4.7}$$

When the RK matrix A is not invertible, an expression similar to (4.7) can be obtained for certain classes of IRK methods such as the Lobatto IIIA and Lobatto IIIB methods.

From now on we assume for simplicity that the matrix A of RK coefficients is invertible. The coefficients γ_i for $i = 1, \ldots, s$ remain free, they are only required to satisfy $\gamma_i > 0$ which is a natural assumption to ensure the invertibility of the matrices H_i. When all these coefficients are equal, only one matrix decomposition of dimension n is needed. This is quite advantageous on a serial computer compared to some other implementations of implicit Runge-Kutta methods [20, 21]. This new approach can also be extended to SPARK methods [22, 32] applied to differential-algebraic equations [24].

EXAMPLE 4.2. *Consider the 2-stage Radau IIA method whose RK matrix is given by*

$$A = \begin{pmatrix} 5/12 & -1/12 \\ 3/4 & 1/4 \end{pmatrix}. \tag{4.8}$$

Assuming $T = I$ and $\Gamma = \gamma I$, we obtain $\Omega = \gamma^2 A^{-1}$ and

$$B(z) = \frac{1}{(1-\gamma z)^2} \begin{pmatrix} 1 - (5/12 + 3\gamma^2/2)z + \gamma^2 z^2 & (1/12 - \gamma^2/2)z \\ (9\gamma^2/2 - 3/4)z & 1 - (1/4 + 5\gamma^2/2)z + \gamma^2 z^2 \end{pmatrix}.$$

Ideally we would like $B(z)$ to be as close as possible to the identity matrix. By taking $\gamma := 1/\sqrt{6} \approx 0.408$ the matrix $B(z)$ becomes diagonal with double eigenvalue

$$\lambda(z) = 1 + z\frac{\sqrt{2/3} - 2/3}{(1 - z/\sqrt{6})^2}.$$

Hence, the condition number of $B(z)$ satisfies $\kappa(B(z)) = 1$ and is therefore minimal. This choice of γ is not only natural, but also optimal in other ways which will be stated precisely in the next section. Interestingly the value $1/\sqrt{6}$ also appears in [28] for the same 2-stage Radau IIA method, but for a different type of iterations.

5. Optimal choices of coefficients γ_i. For simplicity we assume the matrix A of RK coefficients to be invertible. The coefficients Ω_{ij} for $i, j = 1, \ldots, s$ are given by (4.7). The coefficients $\gamma_i > 0$ for $i = 1, \ldots, s$ remain to be determined. We consider the preconditioned matrix $B(z) := Q(z)K(z)$ given in (4.5) corresponding to the Dahlquist test equation (3.4). This matrix $B(z)$ depends on the coefficients γ_i of Γ. Ideally we would like to have $B(z) = I_s$ and this is of course generally not possible. Hence, we must define some criterion to determine the remaining coefficients γ_i. Here we present a few criteria based on the solution of optimization problems

$$\min_{\gamma_1 > 0, \ldots, \gamma_s > 0} \phi(\gamma_1, \ldots, \gamma_s) \tag{5.1}$$

for diverse functions ϕ in relation with the convergence properties of preconditioned linear iterative methods.

A first choice for ϕ is given by

$$\phi_\infty(\gamma_1, \ldots, \gamma_s) := \max_{\mathrm{Re}(z) \le 0} \left(\max_{i=1,\ldots,s} |\lambda_i(B(z)) - 1| \right) \tag{5.2}$$

where $\lambda_i(B(z))$ for $i = 1, \ldots, s$ are the eigenvalues of matrix $B(z)$. The goal is to obtain a good clusterization of the eigenvalues of the preconditioned approximate Jacobian to the value 1 to ensure a rapid convergence of the preconditioned linear iterative method. This is justified since the convergence of most linear iterative methods, such as GMRES [30, 31], is dictated by the eigenspectrum of the preconditioned matrix [7, 19, 26, 30]. We have

$$\phi_\infty(\gamma_1, \ldots, \gamma_s) = \max_{i=1,\ldots,s} \left(\max_{\mathrm{Re}(z) \le 0} |\lambda_i(B(z)) - 1| \right) = \max_{i=1,\ldots,s} \left(\max_{\mathrm{Re}(z) = 0} |\lambda_i(B(z)) - 1| \right)$$

from the maximum principle. Even for T fixed, finding a solution to the global optimization problem (5.1)-(5.2) is certainly difficult and remains an open question. Nevertheless, for example for the 2-stage Radau IIA (4.8) and $T = I$ we have obtained numerically $\gamma_{1,\min} \approx 0.25, \gamma_{2,\min} \approx 0.66$ and $\phi_{\infty,\min} \approx 0.04$ in (5.2). In the situation where we assume that all parameters γ_i are equal to a unique value γ, i. e., $\Gamma = \gamma \cdot I_s$, the eigenvalues $\lambda_i(B(z))$ are independent of T since, see again (4.7),

$$B(z) = \frac{\gamma^2}{(1-\gamma z)^2} T(I_s - zA^{-1})(I_s - zA)T^{-1},$$

and we obtain

$$|\lambda_i(B(z)) - 1| = \left| \frac{\gamma z}{(1-\gamma z)^2} \right| \left| \frac{\mu_i}{\gamma} \left(1 - \frac{\gamma}{\mu_i} \right)^2 \right|$$

where μ_i for $i = 1, \ldots, s$ are the eigenvalues of A. The maximum value of the term $\left|\gamma z/(1-\gamma z)^2\right|$ for $\mathrm{Re}(z) \leq 0$ is independent of γ and is equal to $1/2$ (value taken at $z = \pm \mathrm{i}/\gamma$). Hence, we need to determine γ as the optimal solution of

$$\min_{\gamma > 0} \left(\max_{i=1,\ldots,s} f_i(\gamma) \right) \tag{5.3a}$$

where

$$f_i(\gamma) := \left| \frac{\mu_i}{\gamma} \left(1 - \frac{\gamma}{\mu_i} \right)^2 \right| = \frac{|\mu_i|}{\gamma} + \frac{\gamma}{|\mu_i|} - 2\cos(\theta_i) \tag{5.3b}$$

with $\theta_i := \arg(\mu_i)$. The minimum value of $f_i(\gamma)$ holds at $\gamma = |\mu_i|$. Interestingly a similar derivation was made in the context of blended implicit methods [4, 5, 6].

THEOREM 5.1. *The solution $\gamma_{\min}$ of the optimization problem (5.3) satisfies either $\gamma_{\min} \in \{|\mu_1|, \ldots, |\mu_s|\}$ or lies at the intersection of two curves f_i, i. e., $f_j(\gamma_{\min}) = f_k(\gamma_{\min})$ for two distinct indices j and k.*

Proof. The functions $f_i(\gamma)$ are convex and satisfy

$$\lim_{\gamma \to 0^+} f_i(\gamma) = +\infty, \quad \lim_{\gamma \to +\infty} f_i(\gamma) = +\infty.$$

Hence, the result follows. □

For $s = 2$ and $\mu_1 = \overline{\mu}_2$ we obtain $\gamma_{\min} = |\mu_1|$. For example, for the 2-stage Radau IIA method (4.8) we have $\gamma_{\min} = 1/\sqrt{6}$ and $\phi_{\infty,\min} = 1 - \sqrt{6}/3 \approx 0.184$ in (5.2).

A second choice for ϕ similar to (5.2) is given by

$$\phi_1(\gamma_1, \ldots, \gamma_s) := \max_{\mathrm{Re}(z) \leq 0} \left(\sum_{i=1}^{s} |\lambda_i(B(z)) - 1| \right). \tag{5.4}$$

Finding a solution to the global optimization problem (5.1)-(5.4) is also an open question. If we assume in addition that all parameters γ_i are equal to a unique value γ, we obtain

$$\phi_1(\gamma, \ldots, \gamma) = \max_{\mathrm{Re}(z) \leq 0} \left| \frac{\gamma z}{(1-\gamma z)^2} \right| \sum_{i=1}^{s} \left(\frac{|\mu_i|}{\gamma} + \frac{\gamma}{|\mu_i|} - 2\cos(\theta_i) \right).$$

Hence, we need to determine γ as the optimal solution of

$$\min_{\gamma > 0} f(\gamma) \tag{5.5a}$$

where

$$f(\gamma) := \frac{1}{\gamma} \sum_{i=1}^{s} |\mu_i| + \gamma \sum_{i=1}^{s} \frac{1}{|\mu_i|} - 2 \sum_{i=1}^{s} \cos(\theta_i). \tag{5.5b}$$

The minimum value of this optimization problem holds at

$$\gamma_{\min} = \sqrt{\frac{\sum_{i=1}^{s} |\mu_i|}{\sum_{i=1}^{s} 1/|\mu_i|}}.$$

For $s = 2$ and $\mu_1 = \overline{\mu}_2$ we obtain again $\gamma_{\min} = |\mu_1|$. For example, for the 2-stage Radau IIA method (4.8) we obtain again $\gamma_{\min} = 1/\sqrt{6}$.

A third natural choice for ϕ in (5.1) is given by

$$\phi_F(\gamma_1, \ldots, \gamma_s) := \max_{\mathrm{Re}(z)\leq 0} \|B(z) - I_s\|_F \tag{5.6}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. The goal here is to have the preconditioned approximate Jacobian as close to the identity matrix as possible. From the maximum principle we have

$$\max_{\mathrm{Re}(z)\leq 0} \|B(z) - I_s\|_F = \max_{\mathrm{Re}(z)=0} \|B(z) - I_s\|_F.$$

Finding a solution to the global optimization problem (5.1)-(5.6) is also an open question. Nevertheless, for example for the 2-stage Radau IIA (4.8) and $T = I$ we have obtained numerically $\gamma_{1,\min} \approx 0.32, \gamma_{2,\min} \approx 0.60$ and $\phi_{F,\min} \approx 0.16$ in (5.6). Assuming $\Gamma = \gamma \cdot I_s$, we have

$$\|B(z) - I_s\|_F = \left|\frac{\gamma z}{(1-\gamma z)^2}\right| \|\gamma A^{-1} + \gamma^{-1} A + 2I_s\|_F.$$

Hence, we need to determine γ as the optimal solution of

$$\min_{\gamma>0} g(\gamma) \tag{5.7a}$$

where

$$g(\gamma) := \|\gamma A^{-1} + \gamma^{-1} A + 2I_s\|_F^2. \tag{5.7b}$$

The solution $\gamma_{\min}$ of the optimization problem (5.7) must be one of the roots of $g'(\gamma)$. For example, for the 2-stage Radau IIA method (4.8) we obtain

$$g(\gamma) = 29\gamma^2 + 16\gamma + \frac{11}{2} + \frac{8}{3\gamma} + \frac{29}{36\gamma^2}.$$

We have

$$g'(\gamma) = \frac{58}{\gamma^3}\left(\gamma - \frac{1}{\sqrt{6}}\right)\left(\gamma + \frac{1}{\sqrt{6}}\right)\left(\gamma^2 + \frac{8}{29}\gamma + \frac{1}{6}\right) \text{ and } g''(\gamma) = 58 + \frac{16}{3\gamma^3} + \frac{29}{6\gamma^4}.$$

Therefore, the minimum of $g(\gamma)$ is once again attained at $\gamma_{\min} = 1/\sqrt{6}$.

6. A numerical experiment. We consider a reaction-diffusion problem, the *Brusselator* system in one spatial variable, see [20],

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2},$$
$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2}$$

where $x \in [0,1]$ and $\alpha \geq 0, A$, and B are constant parameters. The boundary conditions for u and v are $u(0,t) = 1 = u(1,t), v(0,t) = 3 = v(1,t), u(x,0) = 1 + \sin(2\pi x), v(x,0) = 3$. We apply the method of lines by discretizing the diffusion terms using finite differences on a grid of N points $x_i = i/(N+1)$ for $i = 1, \ldots, N$, $\Delta x = 1/(N+1)$. We consider the value $N = 500$ and parameters $A = 1, B = 3, \alpha = 0.02$. We obtain a system of $2N = 1000$ differential equations

$$\frac{du_i}{dt} = 1 + u_i^2 v_i - 4u_i + \frac{0.02}{(\Delta x)^2}\left(u_{i-1} - 2u_i + u_{i+1}\right), \tag{6.1a}$$

$$\frac{\partial v_i}{\partial t} = 3u_i - u_i^2 v_i + \frac{0.02}{(\Delta x)^2}\left(v_{i-1} - 2v_i + v_{i+1}\right) \tag{6.1b}$$

where $u_0(t) = 1 = u_{N+1}(t)$, $v_0(t) = 3 = v_{N+1}(t)$, $u_i(0) = 1 + \sin(2\pi x_i)$, and $v_i(0) = 3$ for $i = 1, \ldots, N$. We consider the time interval $[t_0, t_{\mathrm{end}}] = [0, 10]$ and the $s = 3$-stage Radau IIA method. The eigenvalues of the

Jacobian matrix J have a wide spectrum. The largest negative eigenvalue of J is close to -20000, so the system is stiff. By ordering the variables as $y = (u_1, v_1, u_2, v_2, u_3, v_3, \ldots)$, matrices $I - \gamma hJ$ have a bandwidth of 5. They are decomposed using the routine `DGBTRF` of LAPACK for banded matrices [1]. We have taken the absolute and relative error tolerances for each component equal to a certain error tolerance TOL. We give some statistics obtained with the code SPARK3 on this problem in Table 6.1. The *TOL-norm* $\|\cdot\|_{\text{tol}}$ is a scaled 2-norm which depends on absolute and relative error tolerances for each component, $ATOL_i$ and $RTOL_i$ respectively, to be specified by the user

$$\|y\|_{\text{tol}} := \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i}{D_i}\right)^2}, \qquad D_i := ATOL_i + RTOL_i|y_i|. \tag{6.2}$$

We consider the preconditioner with $T = I$ and $\gamma_i = \gamma$ for $i = 1, \ldots, s$ where $\gamma = |\mu_1| \approx 0.246232757526440536$ both for the sequential and parallel versions in order to obtain identical results but different timings in order to compare these versions. In the parallel version we decompose the matrix $M - \gamma hJ$ on each processor. For the results in Table 6.1, the preconditioner is applied to the system of linear equations only once per modified Newton iteration and we have set $ATOL_i = RTOL_i := TOL$.

TABLE 6.1
Results for the 3-stage Radau IIA method on the Brusselator equations (6.1) with 1 preconditioner solve per modified Newton iteration, $ATOL_i = RTOL_i := TOL$, $T = I$, $\gamma = |\mu_1| \approx 0.246232757526440536$

Error tolerance TOL	10^{-3}	10^{-6}	10^{-9}
Measured error in TOL-norm $\|\cdot\|_{\text{tol}}$	0.59	0.39	0.19
CPU-time [s] - sequential	1.19	2.38	6.35
CPU-time [s] - parallel	0.70	2.02	4.12
number of steps	24	49	190
number of rejected steps	0	5	1
number of function evaluations	243	531	1524
number of modified Newton iterations	75	176	508
number of J evaluations	6	1	1
number of P decompositions	21	29	28
number of P solves	81	177	508
number of matrix-vector products	0	0	0

TABLE 6.2
Results for the 3-stage Radau IIA method on the Brusselator equations (6.1) with 2 preconditioner solves per modified Newton iteration, $ATOL_i = RTOL_i := TOL$, $T = I$, $\gamma = |\mu_1| \approx 0.246232757526440536$.

Error tolerance TOL	10^{-3}	10^{-6}	10^{-9}
Measured error in TOL-norm $\|\cdot\|_{\text{tol}}$	0.11	0.81	0.11
CPU-time [s] - sequential	1.44	4.50	11.01
CPU-time [s] - parallel	0.82	2.60	6.61
number of steps	18	50	187
number of rejected steps	0	3	1
number of function evaluations	183	606	1533
number of modified Newton iterations	58	202	511
number of J evaluations	3	1	1
number of P decompositions	18	32	24
number of P solves	117	401	1019
number of matrix-vector products	56	199	508

For the results in Table 6.2 we consider the application of one preconditioned Richardson iteration as preconditioned linear iterative solver, corresponding to two applications of the preconditioner per modified Newton iteration. We do not observe a significant gain compared to Table 6.1. This demonstrates numerically the quality of the preconditioner.

The numerical experiments discussed in this section were made on a SGI Power Challenge using 3 processors for the parallel version.

7. Conclusion. We have considered the application of IRK methods to implicit systems of ODEs. The major difficulty and computational bottleneck for an efficient implementation of these numerical integration methods is to solve the resulting nonlinear systems of equations. Linear systems of the modified Newton method can be solved approximately with a preconditioned linear iterative method. In this paper we have developed a preconditioner which is asymptotically exact for stiff systems. The preconditioner requires m independent matrices of the type $M-\gamma_i hJ$ ($\gamma_i > 0$, $i = 1, \ldots, m$) to be decomposed. The integer parameter m of the number of decompositions satisfies $1 \leq m \leq s$ and is free, $m = 1$ being particularly advantageous for serial computers. Not only the decompositions parts of the preconditioner are parallelizable, but the solution parts involved when applying the preconditioner are also parallelizable, making the preconditioner truly parallel. This preconditioner has been implemented in the code SPARK3 and is shown to be effective on a problem with diffusion.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, Second Edition, 1995.

[2] P. N. Brown and A. C. Hindmarsh, *Matrix-free methods for stiff systems of ODE's*, SIAM J. Numer. Anal., 23 (1986), pp. 610–638.

[3] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, *Using Krylov methods in the solution of large-scale differential-algebraic systems*, SIAM J. Sci. Comput., 15 (1994), pp. 1467–1488.

[4] L. Brugnano and C. Magherini, *Blended implementation of block implicit methods for ODEs*, Appl. Numer. Math., 42 (2002), pp. 29–45.

[5] ———, *The BiM code for the numerical solution of ODEs*, J. Comput. Appl. Math., 164–165 (2004), pp. 145–158.

[6] L. Brugnano, C. Magherini, and F. Mugnai, *Blended implicit methods for the numerical solution of DAE problems*, J. Comput. Appl. Math., 189 (2006), pp. 34–50.

[7] S. L. Campbell, I. C. F. Ipsen, C. T. Kelley, and C. D. Meyer, *GMRES and the minimal polynomial*, BIT, 36 (1996), pp. 32–43.

[8] T. F. Chan and K. R. Jackson, *The use of iterative linear equation solvers in codes for large systems of stiff IVPs for ODEs*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 378–417.

[9] G. J. Cooper and J. C. Butcher, *An iteration scheme for implicit Runge-Kutta methods*, IMA J. Numer. Anal., 3 (1983), pp. 127–140.

[10] G. J. Cooper and R. Vignesvaran, *A scheme for the implementation of implicit Runge-Kutta methods*, Computing, 45 (1990), pp. 321–332.

[11] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[12] C. W. Gear and Y. Saad, *Iterative solution of linear equations in ODE codes*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 583–601.

[13] I. Gladwell and R. I. Hay, *Vector- and parallelization of ODE BVP codes*, Par. Comput., 12 (1989), pp. 343–350.

[14] G. Golub and C. F. van Loan, *Matrix computations*, Johns Hopkins Studies in the Math. Sci., The Johns Hopkins Univ. Press, Baltimore and London, Third Edition, 1996.

[15] S. González-Pinto, C. González-Concepción, and J. I. Montijano, *Iterative schemes for Gauss methods*, Comput. Math. Applic., 27 (1994), pp. 67–81.

[16] S. González-Pinto, J. I. Montijano, and S. Pérez-Rodríguez, *On the numerical solution of stiff IVPs by Lobatto IIIA Runge-Kutta methods*, J. Comput. Appl. Math., 82 (1997), pp. 129–148.

[17] ———, *Implementation of high-order implicit Runge-Kutta methods*, Comput. Math. Applic., 41 (2001), pp. 1009–1024.

[18] S. González-Pinto, J. I. Montijano, and L. Rández, *Iterative schemes for three-stage implicit Runge-Kutta methods*, Appl. Numer. Math., 17 (1995), pp. 363–382.

[19] A. Greenbaum, *Iterative methods for solving linear systems*, vol. 17 of Frontiers in Applied Math., SIAM, Philadelphia, 1997.

[20] E. Hairer and G. Wanner, *Solving ordinary differential equations II. Stiff and differential-algebraic problems*, vol. 14 of Comput. Math., Springer, Berlin, Second Revised Edition, 1996.

[21] W. Hoffmann and J. J. B. de Swart, *Approximating Runge-Kutta matrices by triangular matrices*, BIT, 37 (1997), pp. 346–354.

[22] L. O. Jay, *Structure preservation for constrained dynamics with super partitioned additive Runge-Kutta methods*, SIAM J. Sci. Comput., 20 (1999), pp. 416–446.

[23] ———, *Inexact simplified Newton iterations for implicit Runge-Kutta methods*, SIAM J. Numer. Anal., 38 (2000), pp. 1369–1388.

[24] ———, *Iterative solution of nonlinear equations for SPARK methods applied to DAEs*, Numer. Algor., 31 (2002), pp. 171–191.

[25] L. O. Jay and T. Braconnier, *A parallelizable preconditioner for the iterative solution of implicit Runge-Kutta type methods*, J. Comput. Appl. Math., 111 (1999), pp. 63–76.

[26] C. T. Kelley, *Iterative methods for linear and nonlinear equations*, vol. 16 of Frontiers in applied mathematics, SIAM, Philadelphia, 1995.

[27] M. Paprzycki and I. Gladwell, *A parallel chopping algorithm for ODE boundary value problems*, Par. Comput., 19 (1993), pp. 651–666.

[28] S. Pérez-Rodríguez, S. González-Pinto, and B. P. Sommeijer, *An iterated Radau method for time-dependent PDEs*, J. Comput. Appl. Math., (2009). In press.
[29] W. Rheinboldt, *Methods for solving systems of nonlinear equations*, SIAM Classics in Appl. Math., SIAM, Philadelphia, Second Revised and Expanded Edition, 1998.
[30] Y. Saad, *Iterative methods for sparse linear systems*, PWS Publ. Co., Boston, MA, 1996.
[31] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
[32] M. Schaub and B. Simeon, *Blended Lobatto methods in multibody dynamics*, ZAMM, 7 (2001), pp. 0–10.
[33] P. J. van der Houwen and J. J. B. de Swart, *Triangularly implicit iteration methods for ODE-IVP solvers*, SIAM J. Sci. Comput., 18 (1997), pp. 41–55.

Edited by: Pierluigi Amodio and Luigi Brugnano
Received: March 26, 2009
Accepted: May 15, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 373–383. http://www.scpe.org
ISSN 1895-1767

PARALLEL NUMERICAL SOLUTION OF ABD AND BABD LINEAR SYSTEMS ARISING FROM BVPS*

PIERLUIGI AMODIO† AND GIUSEPPE ROMANAZZI‡

Abstract. We consider linear systems with coefficient matrices having the ABD or the Bordered ABD (BABD) structures. These systems arise in the discretization of BVPs for ordinary and partial differential equations with separated and non-separated boundary conditions, respectively. We describe the cyclic reduction algorithm for the solution of BABD linear systems which allowed us to write the codes *BABDCR* and *GBABDCR* (the latter code is suitable for matrices with a more generic BABD structure). A comparison of the *GBABDCR* code with respect to the well-known sequential code *COLROW* on ABD linear systems is then analysed. We report some tests on an OpenMP Fortran 90 parallel version of the *GBABDCR* code and finally we discuss about the use of *GBABDCR* inside the BVP code *BVP_SOLVER*.

Key words: boundary value problems, ABD and BABD linear systems, parallel solution

1. Introduction. The discretization of Boundary Vale Problems for ordinary and partial differential equations leads often to linear systems with Almost Block Diagonal (ABD) structure in case of separated boundary conditions, and Bordered ABD structure (BABD) in case of non-separated conditions (see [3, 18, 24]).

BABD linear systems

$$Ax = f \tag{1.1}$$

have the coefficient matrix A with the following sparsity structure

$$A = \begin{pmatrix} B_a & & & & & & B_b \\ V_1 & & & & & & \\ & V_2 & & & & & \\ & & V_3 & & & & \\ & & & V_4 & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & V_N \end{pmatrix}. \tag{1.2}$$

We recognize the boundary blocks B_a and B_b on the first row, and the block rows V_i which have some columns that overlap the blocks on the adjacent rows. For this reason, each block V_i can be represented as

$$V_i = \begin{pmatrix} S_{i-1} & T_i & R_i \end{pmatrix}, \tag{1.3}$$

where the columns of S_{i-1} and R_i overlap columns of V_{i-1} and V_{i+1}, respectively. Also V_1 and V_N have the same structure in (1.3): the first columns of V_1 (i. e., S_0) are overlapped with those of B_a while the last columns of V_N (i. e., R_N) are overlapped with those of B_b, see (1.2). We set the size of each block V_i equal to $n_i \times (m_{i-1} + k_i + m_i)$, where m_i is the number of overlapped columns between the blocks V_i and V_{i+1}, and k_i

*Work developed within the project "Numerical methods and software for differential equations"
†Dipartimento di Matematica, Università di Bari, Bari, Italy (amodio@dm.uniba.it)
‡Departamento de Matemática, Universidade de Coimbra, Coimbra, Portugal (roman@mat.uc.pt)

is the number of the non-overlapped columns of V_i; the size of the boundary blocks B_a and B_b is $n_0 \times m_0$ and $n_0 \times m_N$, respectively. Since A is a square matrix we have

$$\sum_{i=0}^{N} n_i = m_0 + \sum_{i=1}^{N} (m_i + k_i); \tag{1.4}$$

it is also supposed that N is much larger with respect to each n_i, m_i and k_i.

The ABD structure differs from the BABD structure for the presence of boundary blocks that have some null rows (the non-null rows of B_a correspond to null rows of B_b and viceversa). In this case it is preferable to refer to B_a and B_b as blocks of size $n_a \times m_0$ and $n_b \times m_N$, respectively. We impose then that no rows are overlapped between B_a and B_b and we set $n_0 = n_a + n_b$ in (1.4). Moreover, B_b is located after V_N as the last block row so that the coefficient matrix can be represented as

$$A = \begin{pmatrix} B_a & & & & & & \\ V_1 & & & & & & \\ & V_2 & & & & & \\ & & V_3 & & & & \\ & & & V_4 & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & V_N & \\ & & & & & & B_b \end{pmatrix}. \tag{1.5}$$

We note that ABD linear systems can be also solved by BABD solvers; this implies an higher computational cost and fill-in, see [24]. Conversely, a doubling of the size of each block is required for solving a BABD linear system using an ABD solver.

Since ABD linear systems are easier to solve than BABD systems, historically the former problem has received much more attention and several codes have been proposed. We quote the package *SOLVEBLOK* [14] that uses Gaussian elimination with partial pivoting to ensure stability and requires fill-in. On the contrary, the packages *COLROW* and *ARCECO* in [15] are based on a modified version of Varah's alternate row and column stable elimination [27] which exploits the structure of the ABD matrices to avoid fill-in. In particular, *COLROW* solves ABD linear systems with blocks V_i of constant size $n \times (2m + k)$, so that we have $n = m + k$ and $n_i = n$, $m_i = m$ and $k_i = k$ for each $i = 1, \ldots, N$.

Most nonlinear BVP packages employ ABD packages. The BVP code *COLSYS* [9] uses *SOLVEBLOK* to solve ABD linear systems arising from the use of orthogonal spline collocation (OSC) at Gauss points with B-spline bases. *COLNEW* [11] uses a modified version of *SOLVEBLOK* to solve ABD linear systems arising from the application of OSC at Gauss points with monomial spline bases. Both the Mono Implicit Runge Kutta (MIRK) code with defect control *MIRKDC* [17] and its new implementation *BVP_SOLVER* [28] (the latter solves a wider class of BVPs with respect the former) use *COLROW* as solver for the obtained ABD systems. Modified versions of *COLROW* are used in the deferred correction code *TWPBVP* [13]. Other versions of *COLROW* are used in COLMOD [26], a modified version of *COLNEW*, and in *ACDC* [12], that uses automatic continuation and OSC at Lobatto points to solve singularly perturbed BVODEs.

For what concerns the numerical solution of BABD systems (1.1)-(1.2), a common idea is to double the size of the system (that is the number of unknowns) in order to obtain an equivalent ABD system that can be solved with ABD solvers. Of course, this involves a high computational cost. In fact, since the computational cost of a general ABD solver is cubic with the dimension of the blocks, then its cost for solving a BABD linear

system is eight times larger with respect to the solution of an ABD linear system with blocks of the same size as the given BABD system.

The first available package for solving BABD systems is *RSCALE*; it is a shared memory parallel code used inside *PMIRKDC* [21], a parallel version of *MIRKDC*. In [7, 8] the cyclic reduction algorithm is applied to the BABD matrix (1.2) with a simplified structure: the blocks S_i and R_i are square of dimension m and the blocks T_i are null. This structure is also used by *RSCALE*. The coefficient matrix considered is then block lower bidiagonal with an additional block in the right-upper corner. The sequential code *BABDCR*, introduced in [8] and available at the url

`http://www.netlib.org/toms/859`.

solves this kind of linear systems. Moreover, in [4] we have generalized the code *BABDCR* to solve BABD linear systems with the general structure (1.1)-(1.2) where each block V_i have size $n \times (2m+k)$, with $k > 0$ and $n = m + k$. This latter code is called *GBABDCR* and is available on the net at the url

`http://www.pitagora.dm.uniba.it/~romanazzi/babdcr.html`.

In [4, 5, 8, 24] *BABDCR* and *GBABDCR* have been compared to *RSCALE* and *COLROW* for solving BABD systems. The theoretical and numerical results show that *COLROW* has a computational cost which is till 3 times larger than *GBABDCR*. This means that each code for BVPs with separated boundary conditions (for example, *BVP_SOLVER*) using *COLROW* to solve the associated ABD linear systems, may be generalized to the solution of BVPs with non-separated boundary conditions by just replacing (the calls to) the linear solver *COLROW* with *GBABDCR*. Moreover, *BABDCR* performs better (resulting faster and more precise) and has the same degree of parallelism with respect to *RSCALE*. Therefore, the efficiency of *PMIRKDC* can be improved by replacing *RSCALE* with a parallel version of *BABDCR*.

This work originates from the observation that nowadays personal computers have motherboards with more CPUs (or core-processors), and this permits to overcome the physical limitations (such as speed and memory) of a single core-processor. Moreover these multi-core processors are easily accessible, and we can use them to speed-up the execution of each numerical code if the underlying algorithm is parallelizable. In our case, since cyclic reduction has an obvious parallel implementation, we can speed up the solution of ABD linear systems by implementing *GBABDCR* on such parallel architectures.

We propose, in fact, a shared memory implementation of *GBABDCR* and compare it with *COLROW* (that is not parallelizable) on multi-core computers. We believe that the replacement of *COLROW* in the existing (previously cited) BVODE packages with the parallel implementation of *GBABDCR*, can lead to a twofold advantage: the reduction of the computational cost when the code is run on multi-core processors, and the solution with no extra cost of BVODEs with non-separated boundary conditions.

The paper is organized as follows: in the next section we briefly sketch the cyclic reduction algorithm applied to general BABD linear systems, in Section 3 we explain the strategy used to parallelize the code on shared and distributed memory computers, and finally in the last section we compare the shared memory parallel implementation of *GBABDCR* with *COLROW* in the solution of ABD systems and we discuss the performance of *BVP_SOLVER* when *GBABDCR* replaces *COLROW*.

2. The cyclic reduction algorithm. Let us rewrite the coefficient matrix (1.2)-(1.3) as

$$A = \begin{pmatrix} B_a & & & & & & & & & B_b \\ S_0 & T_1 & R_1 & & & & & & & \\ & & S_1 & T_2 & R_2 & & & & & \\ & & & & S_2 & T_3 & R_3 & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & S_{N-1} & T_N & R_N \end{pmatrix}. \tag{2.1}$$

In accordance with the structure of (2.1) we define the right hand side f of the linear system (1.1) as $f = \begin{pmatrix} f_0^T & f_1^T & \dots & f_N^T \end{pmatrix}^T$ where each f_i is of length n_i, and the solution vector

$$x = \begin{pmatrix} z_0^T & w_1^T & z_1^T & \dots & w_{N-1}^T & z_N^T \end{pmatrix}^T,$$

where z_i and w_i are of length m_i and k_i, respectively.

We solve the system (1.1) using a block cyclic reduction algorithm, that is, a recursive approach that reduces the original linear system to subsystems with a smaller number of unknowns. In this process the first and the last unknowns, z_0 and z_N, are always among the unknowns of the successive reduced systems; moreover, the first row containing the boundary blocks is unchanged in the reduction process. The boundary blocks B_a and B_b are then maintained in the first row of each reduction step.

Following [4], we consider a reduction step to eliminate, locally in each block V_i, the odd unknowns w_i of the solution vector x. We observe that, since A is a non-singular matrix, the blocks T_i of size $n_i \times k_i$, have full rank k_i because they are not overlapped by adjacent V_i blocks. We may then compute the factorization:

$$\widetilde{P}_i T_i = \begin{pmatrix} \widetilde{L}_i \\ \widetilde{G}_i \end{pmatrix} \widetilde{U}_i = \begin{pmatrix} I & \\ \widetilde{F}_i & I \end{pmatrix} \begin{pmatrix} \widetilde{L}_i \widetilde{U}_i \\ O \end{pmatrix} \tag{2.2}$$

where $\widetilde{P}_i$ is a suitable permutation matrix, $\widetilde{L}_i$ and $\widetilde{U}_i$ are square matrices, and $\widetilde{F}_i = \widetilde{G}_i \widetilde{L}_i^{-1}$.

Multiplying V_i on the left by $\widetilde{P}_i$ and the inverse of the lower triangular matrix in the last term of (2.2) we obtain

$$\begin{pmatrix} I & \\ -\widetilde{F}_i & I \end{pmatrix} \widetilde{P}_i \begin{pmatrix} S_{i-1} & T_i & R_i \end{pmatrix} = \begin{pmatrix} \widetilde{S}_{i-1} & \widetilde{L}_i \widetilde{U}_i & \widetilde{R}_i \\ \widehat{S}_{i-1} & & \widehat{R}_i \end{pmatrix}. \tag{2.3}$$

Analogously, we perform the same operations on the right-hand side f_i, thus obtaining corresponding vectors $\tilde{f}_i$ and $\widehat{f}_i$ for the right side. The row with the boundary blocks and the second row of (2.3) (for each $i = 1, \ldots, N$) give the linear system

$$\begin{pmatrix} B_a & & & & B_b \\ \widehat{S}_0 & \widehat{R}_1 & & & \\ & \widehat{S}_1 & \widehat{R}_2 & & \\ & & \ddots & \ddots & \\ & & & \widehat{S}_{N-1} & \widehat{R}_N \end{pmatrix} \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix} = \begin{pmatrix} f_0 \\ \widehat{f}_1 \\ \widehat{f}_2 \\ \vdots \\ \widehat{f}_N \end{pmatrix}. \tag{2.4}$$

which has dimension equal to $\displaystyle\sum_{i=0}^{N} m_i = n_0 + \sum_{i=1}^{N} (n_i - k_i)$ and no longer depends on the unknowns w_i. These unknowns will be computed in the last step of the back-substitution phase (when all the z_i will be known), by using the first row of (2.3):

$$\tilde{L}_i \tilde{U}_i w_i = \tilde{f}_i - \tilde{S}_{i-1} z_{i-1} - \tilde{R}_i z_i.$$

Factorization (2.3) does not require additional memory since $\widetilde{F}_i$ may be saved together with L_i and U_i in place of T_i. Therefore, this first reduction should be considered as a (completely parallelizable) initial step to be applied to ABD or BABD matrices in order to simplify their structure.

Returning to the solution of (1.1), system (2.4) may be further on reduced by considering the cyclic reduction algorithm in [7, 8] (even if blocks $\widehat{S}_i$ and $\widehat{R}_i$ are not square). At first we compute the LU factorization of the $(n_i - k_i + n_{i+1} - k_{i+1}) \times m_i$ matrix (of rank m_i)

$$P_i \begin{pmatrix} \widehat{R}_i \\ \widehat{S}_i \end{pmatrix} = \begin{pmatrix} L_i \\ G_i \end{pmatrix} U_i = \begin{pmatrix} I & \\ F_i & I \end{pmatrix} \begin{pmatrix} L_i U_i \\ O \end{pmatrix}$$

that, applied to the block rows of index i and $i+1$ in (2.4), gives

$$\begin{pmatrix} I & \\ -F_i & I \end{pmatrix} P_i \begin{pmatrix} \widehat{S}_{i-1} & \widehat{R}_i & \\ & \widehat{S}_i & \widehat{R}_{i+1} \end{pmatrix} = \begin{pmatrix} \bar{S}_{i-1} & L_i U_i & \bar{R}_i \\ S'_{i-1} & & R'_i \end{pmatrix}. \tag{2.5}$$

The second row of the right hand-side of (2.5) is independent of z_i and allows to obtain, for $i = 1, 3, 5, \ldots$, a reduced system similar to (2.4) but with $\lceil N/2 \rceil + 1$ block rows and unknowns z_i with even indices. Conversely,

the first row in (2.5) may be used to compute z_i from z_{i-1} and z_{i+1}. Factorization (2.5) requires additional memory for the fill-in blocks F_i.

Iterating this last step on the successively reduced systems we obtain, after $\lceil \log_2 N \rceil$ steps, a 2×2 block (full) linear system

$$\begin{pmatrix} B_a & B_b \\ S_0^* & R_N^* \end{pmatrix} \begin{pmatrix} z_0 \\ z_N \end{pmatrix} = \begin{pmatrix} f_0^* \\ f_N^* \end{pmatrix}. \tag{2.6}$$

The factorization and the solution of (2.6) gives the first and the last unknowns of the vector x. Successively, a back substitution phase allows us to compute, in reverse order, all the other unknowns.

If m and k are constant (with $n = m + k$), the computational cost of this algorithm is

$$(\tfrac{14}{3}m^3 + 4m^2k + 2mk^2 + \tfrac{2}{3}k^3)N. \tag{2.7}$$

If $k = 0$ (all columns are overlapped), we have from (2.7) that the cost is $\frac{14}{3}m^3N$ as for the *BABDCR* algorithm, see [8]. The additional memory requirement (fill-in) is always m^2N (it does not depend on k) since the factorization of the blocks T_i in (2.2) does not require fill-in.

The typical dimensions of the BABD linear system arising from BVPs consists of small dimensions k, m of each row block, with respect to a large number of row blocks N.

In order to make a fair comparison, the computational cost of *COLROW* is

$$(\tfrac{5}{3}m^3 + 4m^2k + 3mk^2 + \tfrac{2}{3}k^3 + mn_an_b - (2m + k)kn_a)N, \tag{2.8}$$

where n_a and n_b, with $n_a + n_b = m$, denote the number of rows of the initial and of the final boundary block, respectively; remember that *COLROW* solves only ABD linear systems. Supposing $n_a = n_b = m/2$, we have that *GBABDCR* costs at most 2.4 times more than *COLROW* and this ratio decreases when k increases (if $k = m$ the ratio is 1.4).

3. Parallel implementation. In this section we analyze and compare the main properties of parallel cyclic reduction algorithms written for shared and distributed memory architectures.

The cyclic reduction algorithm has a straightforward parallel implementation which has been described in several papers. See, for example, [1, 2, 6] where parallel cyclic reduction is used to solve tridiagonal systems and [5, 7, 24] where it is applied to BABD systems.

Let p the number of processors used[1], the first phase of parallel reduction requires that each processor reduces a contiguous set of block rows,

$$\begin{pmatrix} S_{i-1} & T_i & R_i & & & & & \\ & & S_i & T_{i+1} & R_{i+1} & & & \\ & & & & & \ddots & & \\ & & & & & & S_{j-1} & T_j & R_j \end{pmatrix}, \tag{3.1}$$

to a single row

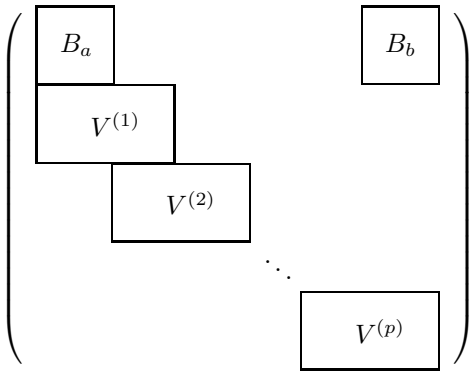
$$\begin{pmatrix} S_{i-1}^* & R_j^* \end{pmatrix}. \tag{3.2}$$

In particular, $q = N - \lceil N/p \rceil p$ processors reduce $\lfloor N/p \rfloor$ block rows and the remaining $p - q$ processors reduce $\lceil N/p \rceil$ block rows each. Note that, when q is non-zero, a different workload per processor is required.

At the end of this phase, each processor stores the block row (3.2) in the first and last m^2 locations of the assigned contiguous block rows (3.1) in place of S_{i-1} and R_j. The fill-in of these reductions is stored locally in a private array of each processor, when a distributed memory architecture is used, or consists of contiguous blocks generated by each single processor, in case of a shared memory architecture.

[1]We use the generic term "processor" to refer to each core per physical processor used in the parallel architecture considered.

Therefore, the first phase of reduction produces a reduced BABD system with coefficient matrix



, (3.3)

where $V^{(i)}$ is computed by the i-th processor; this phase is then completely parallelizable.

Let us now analyze the parallel solution of the system with the coefficient matrix (3.3). Supposing, for sake of simplicity, that p is a power of 2, the parallel MPI algorithm (as in [7, 24, 5]), for a distributed memory architecture, requires $\log_2 p$ synchronizations, communications between processors, and reduction steps. On shared memory architectures, clearly we do not require communications between processors because both the coefficient matrix and the right-hand side of the original system (1.1) are shared among processors. However, $\log_2 p$ synchronizations between processors are necessary.

This part of the algorithm determines, in both the architectures, the 2×2 block system (2.6). The solution of (2.6) is the only sequential part of the algorithm and is followed by the back-substitution phase which still has much parallelism inside.

On a distributed memory architecture it is made possible that all the back-substitution phase requires no more synchronization or communication among the processors by considering bidirectional communications in the reduction phase (see Figure 3.1 and [6] for more details). This means that a copy of (2.6) is solved (concurrently) on all the processors.

On the other hand, on shared memory architectures, the number of processors is halved at each of the last $\log_2 p$ reduction steps and (2.6) is solved on a single processor. Then the number of processors is doubled for the first $\log_2 p$ steps of back-substitution with a synchronization after each step.

In conclusion, if $p \ll N$ the total operation count for each processor is essentially $1/p$ times the cost of (2.7). On shared memory architectures, the number of synchronizations is $2 \log_2 p$. The parallel code does not require additional memory with respect to the sequential code and only a few local variables are needed on each processor. On distributed memory architectures, the number of synchronizations is only $\log_2 p$ but each processor needs to maintain $\log_2 p$ blocks of size $m \times 2m$ to avoid synchronization in the back-substitution and, moreover, $\log_2 p$ communications of $m \times 2m$ arrays and vectors of length m are necessary in the reduction phase.

Since distributed parallel architectures can have even hundreds processors, the main advantage of the parallel algorithm for these architectures is the possibility to strongly reduce the cost of the reduction and back-substitution phases by increasing the number of processors used. In such a case, however, a moderate overhead (of order $\log_2 N$) appears due to the presence of synchronizations and communications.

As already mentioned, shared memory architectures are nowadays much easier to access. They have a limited number of processors (up to 4 in our tests), but the parallel algorithms for these architectures have a double advantage: there is no need to send the initial data to the local memory of each processor and there is no large communication step after each synchronization. This means that, for a given number of processors, these algorithms can lead to a better speed-up than that observed on distributed memory architectures.

4. Numerical results. In this section we report some numerical tests to compare *COLROW* with the shared memory (OpenMP) parallel version of *GBABDCR* (here called *GBABDCR_OMP*) in the solution of ABD systems on multi-core computers. Our aim is to stress that *GBABDCR_OMP* may be efficiently employed in each BVP solver that requires the solution of ABD/BABD systems on the new multi-core personal computers.

For this reason we analyze the execution times of sequential runs performed on an alpha EV6.7 (21264A, 667 MHz) and of parallel runs performed on an Intel Core 2, Quad CPU Q9550, (2.83 GHz) multi-processor with 4 cores. We consider ABD linear systems with blocks of the same size and with the same overlap ($m_i = m$, $k_i = k$ and $n_i = n = k + m$). Moreover, we set the number of initial and final conditions equal to $m/2$.

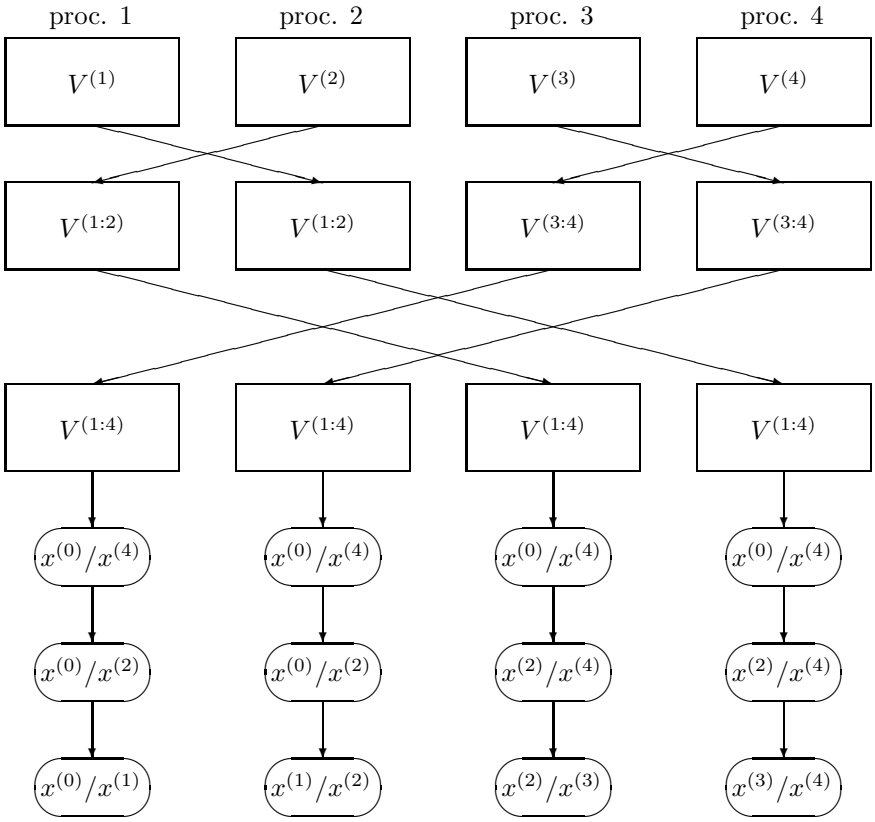


FIG. 3.1. *Communications among processors on a distributed memory architecture with* $p = 4$.

In Tables 4.1 and 4.2 we compare the elapsed time of some *GBABDCR* and *COLROW* runs using the sequential computer to estimate the theoretical operation counts. We observe that, for $k = 0$ the ratio between the elapsed time of *GBABDCR* and *COLROW* decreases from 3.26 to 2.32 (the expected value is 2.4) as we increase n while for $k = m$ the ratio decreases from 4.75 to 1.64 (the expected value is 1.4). A reason of this strange behaviour is due to the large use in *GBABDCR* of BLAS3 routines [16] (such as DGEMM, DTRSM) which have poor performance for small matrices (see tests in [22, 23]) and make this algorithm more efficient for large k and m only.

TABLE 4.1
Elapsed times (in seconds) of GBABDCR and COLROW for solving ABD linear systems with $N = 10000$ *and variable* $n = m$ *(*$k = 0$*)*

n	GBABDCR	COLROW	GBABDCR/COLROW
4	0.0795	0.0244	3.2600
8	0.2525	0.0947	2.6675
12	0.5768	0.2294	2.5149
16	1.0753	0.4377	2.4565
20	1.6946	0.7291	2.3243

These results show therefore that, for small size n of each block, it is quite difficult that *GBABDCR* overcomes the performance of *COLROW* on parallel machines. Effectively for $n < 8$ we were not able to lower *COLROW* timings by using *GBABDCR_OMP* on 4 cores.

In Tables 4.3-4.6 we show the performance of *GBABDCR_OMP* running on shared memory architectures for $N = 10000, 20000, 40000$ and $n = 8, 16$.

We note that, when $N \gg p$, the computational cost of the parallel algorithm scales with the number of processors and in all the considered cases *GBABDCR_OMP* on 4 cores is faster than *COLROW*. Therefore, on

TABLE 4.2
Elapsed times of GBABDCR and COLROW for solving ABD linear systems with $N = 10000$ and variable $n = 2m$ $(k = m)$

n	GBABDCR	COLROW	GBABDCR/COLROW
4	0.0986	0.0207	4.7529
8	0.1840	0.0712	2.5822
12	0.3582	0.1940	1.8465
16	0.5558	0.3235	1.7179
20	0.9118	0.5546	1.6441

TABLE 4.3
Elapsed times of GBABDCR_OMP and COLROW on a quad-core machine for ABD linear systems with $n = m = 8$ $(k = 0)$ and variable N. Speed-up is computed as the ratio between the elapsed time of COLROW and GBABDCR_OMP

		GBABDCR_OMP			speed-up	
N	*COLROW*	1 core	2 cores	4 cores	2 cores	4 cores
10000	0.091	0.199	0.102	0.062	0.892	1.468
20000	0.186	0.398	0.204	0.132	0.912	1.409
40000	0.360	0.797	0.410	0.224	0.878	1.607

a multicore architecture, we can effectively speed-up the performance of any BVP solver when the size of the problem is sufficiently large, just replacing *COLROW* with *GBABDCR_OMP*. Since a code for BABD systems may be directly applied to ABD systems, the modifications in the code are limited to a few instructions.

In particular, in *BVP_SOLVER* we have inserted a subroutine that creates the non-separated boundary blocks of the linear system starting from the given separated boundary conditions and we have added a fill-in vector which is only required by *GBABDCR_OMP*. We point out that in *BVP_SOLVER* the resulting linear systems have a BABD structure with $k = 0$ and $m = n$.

As an example, we have considered the solution of a 8×8 nonlinear system of equations describing fluid injection through one side of a long vertical channel (see [10, Example 1.4]). We have taken into account this problem with the following parameters

$$R = 1000, \quad P = 0.7 * R, \quad \text{METH} = 2$$

and fixed equal to 1 both the initial guess for the solution and the unknown parameter A.

Table 4.7 shows the results obtained by setting the exit tolerance equal to 10^{-6}, 10^{-7} and 10^{-8}. A few comments need to be done on this example. The number of requested linear system solutions (SOLV) is about 8 times the number of factorizations (FACT). Since this size of the ODE is just $n = 8$ and in any ABD/BABD code the computational cost of the factorization phase (FACT) is about n times that of the solution phase (SOLV), then we expect the execution time of the two phases to be approximately equivalent.

From Table 4.7 we observe that the percentage of time required by SOLV using *GBABDCR* is higher than that requested by FACT (the opposite happens when we use *COLROW*) and it is therefore difficult to obtain a large speed-up with the replacement of the linear algebra solver. By considering only the linear algebra part (see % LIN_ALG in the table) of *BVP_SOLVER*, *GBABDCR* reduces its elapsed time of a factor in the range [1.5, 1.7] when we use 4 cores instead of 1. This is anyway sufficient to obtain on 4 cores an execution faster than *COLROW*.

We have to specify that on BVPs of size smaller than 8 we were not able to improve timings obtained with *COLROW*. The size of the considered problem is anyway small, and this implies two negative aspects: linear algebra is not the most time consuming part of the algorithm and, for such small dimensions, *COLROW* is much better than *GBABDCR*. Nevertheless, it is clear that parallelism is really more useful to reduce timing in presence of very large nonlinear BVPs.

A better speed-up is, in fact, obtained when we solve BVODE of larger size. Table 4.8 shows, for example, some results of the application of the *BVP_SOLVER* to a kidney problem of size 21×21 that describes the mass and the energy balance of a renal counter-flow system, see [10]. We note that the Linear Algebra part requires a larger portion of computation (between 60% and 70%) with respect to the previous example and, therefore, the use of *GBABDCR* sensibly improves the performance of *BVP_SOLVER*, when a multi-core computer is

TABLE 4.4
Elapsed times of GBABDCR_OMP and COLROW on a quad-core machine for ABD linear systems with $n = m = 16$ ($k = 0$) and variable N. Speed-up is computed as the ratio between the elapsed time of COLROW and GBABDCR_OMP

		GBABDCR_OMP			speed-up	
N	*COLROW*	1 core	2 cores	4 cores	2 cores	4 cores
10000	0.584	1.038	0.525	0.299	1.112	1.953
20000	1.174	2.054	1.049	0.552	1.119	2.127
40000	2.332	4.131	2.098	1.092	1.112	2.136

TABLE 4.5
Elapsed times of GBABDCR_OMP and COLROW on a quad-core machine for ABD linear systems with $n = 2m = 8$ ($k = m$) and variable N. Speed-up is computed as the ratio between the elapsed time of COLROW and GBABDCR_OMP

		GBABDCR_OMP			speed-up	
N	*COLROW*	1 core	2 cores	4 cores	2 cores	4 cores
10000	0.060	0.122	0.066	0.041	0.909	1.463
20000	0.120	0.249	0.137	0.078	0.876	1.539
40000	0.240	0.500	0.271	0.175	0.886	1.371

used. Finally, we emphasize that the performance of *BVP_SOLVER* can be further improved with *GBABDCR*, by considering that the original problem has three nonseparated boundary conditions and, therefore, it can be recast so that a 18×18 BABD linear system is obtained.

REFERENCES

[1] P. Amodio, L. Brugnano. Parallel factorizations and parallel solvers for tridiagonal linear systems. *Linear Algebra Appl.* **172**, 347–364, 1992.

[2] P. Amodio, L. Brugnano, T. Politi. Parallel factorizations for tridiagonal matrices. *SIAM J. Numer. Anal.* **30**, 813–823, 1993.

[3] P. Amodio, J. R. Cash, G. Roussos, R. W. Wright, G. Fairweather, I. Gladwell, G. L. Kraut and M. Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numer. Linear Algebra Appl.* **7**, no. 5, 275–317, 2000.

[4] P. Amodio, I. Gladwell and G. Romanazzi. Numerical solution of general Bordered ABD linear systems by cyclic reduction. *J. Numer. Anal., Industrial and Applied Mathematics* **1**, no. 1, 5–12, 2006.

[5] P. Amodio, I. Gladwell and G. Romanazzi. An algorithm for the solution of Bordered ABD linear systems arising from Boundary Value Problems. *Multibody Dynamics 2007, ECCOMAS Thematic Conference*, C. L. Bottasso, P. Masarati, L. Trainelli (eds.), Milano (Italy), June 25–28, 2007.

[6] P. Amodio, N. Mastronardi. A parallel version of the cyclic reduction algorithm on a Hypercube. *Paral. Comput.* **19**, 1273–1281, 1993.

[7] P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comput.* **18**, no. 1, 56–68, 1997.

[8] P. Amodio and G. Romanazzi. BABDCR: a Fortran 90 package for the solution of Bordered ABD systems. *ACM Trans. Math. Software* **32**, no. 4, 597–608, 2006. (Available on the web address "http://www.netlib.org/toms/859")

[9] U. M. Ascher, J. Christiansen and R.D. Russell. Algorithm 569: COLSYS: Collocation Software for Boundary-Value ODEs. *ACM Trans. Math. Software* **7**, no. 2, 223–229, 1981.

[10] U. M. Ascher, R.M.M. Mattheij, and R.D. Russell. Numerical Solution of Boundary Value Problems for Ordinary Differential Equations. SIAM Classics In Applied Mathematics, 1995.

[11] G. Bader and U. Ascher. A new basis implementation for a mixed order boundary value ODE solver. *SIAM J. Sci. Statist. Comput.* **8**, no. 4, 483–500, 1987.

[12] J. R. Cash, G. Moore and R.W. Wright. An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems. *ACM Trans. Math. Software* **27**, no. 2, 245–266, 2001.

[13] J. R. Cash and R. W. Wright. A deferred correction method for nonlinear two-point boundary value problems: Implementations and numerical evaluation. *SIAM J. Sci. Statist. Comput.* **12**, no. 4, 971–989, 1991.

[14] C. De Boor and R. Weiss. SOLVEBLOK: A package for solving almost block diagonal linear systems. *ACM Trans. Math. Software* **6**, no. 1, 80–87, 1980.

[15] J. C. Diaz, G. Fairweather and P. Keast. FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Trans. Math. Software* **9**, no. 3, 358–375, 1983.

[16] J. J. Dongarra, J. Du Croz, I. S. Duff and S. Hammarlin, Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Software* **16**, 18–28, 1990.

[17] W. H. Enright and P. H. Muir. Runge-Kutta software with defect control for boundary value ODEs. *SIAM J. Sci. Comput.* **17**, no. 2, 479–497, 1996.

TABLE 4.6

Elapsed times of GBABDCR_OMP and COLROW on a quad-core machine for ABD linear systems with $n = 2m = 16$ ($k = m$) and variable N. Speed-up is computed as the ratio between the elapsed time of COLROW and GBABDCR_OMP

		GBABDCR_OMP			speed-up	
N	*COLROW*	1 core	2 cores	4 cores	2 cores	4 cores
10000	0.370	0.468	0.253	0.145	1.463	2.552
20000	0.742	0.938	0.506	0.278	1.466	2.669
40000	1.485	1.875	0.977	0.552	1.520	2.690

TABLE 4.7

Total elapsed times of BVP_SOLVER using COLROW and GBABDCR_OMP as linear algebra solver on a nonlinear 8×8 BVP. # FACT and # SOLV denote, respectively, the number of linear system factorizations and solutions performed by the code. % FACT and % SOLV denote, respectively, the percentage of the total elapsed time required by linear system factorizations and solutions in BVP_SOLVER. % LIN_ALG = % FACT + % SOLV denotes the percentage of the total elapsed time required by the linear algebra part.

TOL	1e-8	1e-7	1e-6
$N_{\max}$	63357	19281	6466
# FACT	21	20	20
# SOLV	170	161	151
COLROW	30.886	3.409	0.895
% FACT	5.83%	14.66%	21.16%
% SOLV	2.73%	9.20%	17.19%
% LIN_ALG	8.56%	23.86%	38.35%
GBABDCR (1 core)	31.831	3.737	1.067
% FACT	6.86%	14.84%	21.60%
% SOLV	5.42%	16.80%	29.56%
% LIN_ALG	12.28%	31.64%	51.16%
GBABDCR (2 cores)	30.881	3.435	0.917
GBABDCR (4 cores)	30.484	3.260	0.846

TABLE 4.8

Total elapsed times of BVP_SOLVER using COLROW and GBABDCR_OMP as linear algebra solver on a nonlinear 21×21 BVP. # FACT and # SOLV denote, respectively, the number of linear system factorizations and solutions performed by the code. % FACT and % SOLV denote, respectively, the percentage of the total elapsed time required by linear system factorizations and solutions in BVP_SOLVER. % LIN_ALG = % FACT + % SOLV denotes the percentage of the total elapsed time required by the linear algebra part.

TOL	1e-6	1e-4
$N_{\max}$	36128	3016
# FACT	285	177
# SOLV	738	481
COLROW	366.78	25.15
% FACT	56.7%	60.2%
% SOLV	13.7%	13.1%
% LIN_ALG	70.4%	73.3%
GBABDCR (1 core)	415.95	21.88
% FACT	39.5%	40.5%
% SOLV	21.7%	25.2%
% LIN_ALG	61.2%	65.7%
GBABDCR (2 cores)	246.51	14.42
GBABDCR (4 cores)	126.95	13.52

[18] G. Fairweather and I. Gladwell. Algorithms for almost block diagonal linear systems. *SIAM Rev.* **46**, no. 1, 49–58, 2004.
[19] B. Garrett and I. Gladwell. Solving bordered almost block diagonal systems stably and efficiently. *J. Comput. Methods Sci. Engrg.* **1**, 75–98, 2001.
[20] K. R. Jackson and R. N. Pancer. The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs. Techinical Report n. 255/91, Computer Science Department, University of Toronto, 1992.
[21] P. H. Muir, R. N. Pancer and K.R. Jackson. PMIRKDC: a parallel mono-implicit Runge-Kutta code with defect control for boundary value ODEs. *Paral. Comput.* **29**, 711–741, 2003.
[22] J. R. Herrero and J. J. Navarro. Improving Performance of Hypermatrix Cholesky Factorization. In *9th International Euro-Par Conference*, 461–469, August 2003.
[23] A. Remón, E. S. Quintana-Ortí and G. Quintana-Ortí. Cholesky factorization of band matrices using multithreaded BLAS. In B. Kågström, E. Elmroth, J. Dongarra, J. Wásniewski, eds., PARA 2006. Lect. Notes Comp. Sci. **4699**, 608–616. Springer, Heidelberg.
[24] G. Romanazzi. Numerical Solution of Bordered Almost Block Diagonal linear systems arising from BVPs. Ph.D. Thesis, Università di Bari, 2006.
[25] M. Snir, S.W. Otto, S. Huss-Lederman, D. Walker and J. Dongarra. *MPI: The Complete Reference.* The MIT Press, Cambridge, Massachusetts, 1996.
[26] R. W. Wright, J. Cash and G. Moore. Mesh selection for stiff two-point boundary value problems. *Numer. Algorithms* **7**, 205–224, 1994.
[27] J. M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.* **13**, 71–75, 1976.
[28] L. F. Shampine, P. H. Muir and H. Xu. A User-Friendly Fortran BVP Solver. *J. Numer. Anal. Ind. and Appl. Math.* **11**, no. 2, 1201–217, 2006

Edited by: Luigi Brugnano
Received: August 27, 2009
Accepted: September 20, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 385–396. http://www.scpe.org
ISSN 1895-1767

PARALLEL FACTORIZATIONS IN NUMERICAL ANALYSIS*

PIERLUIGI AMODIO† AND LUIGI BRUGNANO‡

Abstract. In this paper we review the parallel solution of sparse linear systems, usually deriving by the discretization of ODE-IVPs or ODE-BVPs. The approach is based on the concept of *parallel factorization* of a (block) tridiagonal matrix. This allows to obtain efficient parallel extensions of many known matrix factorizations, and to derive, as a by-product, a unifying approach to the parallel solution of ODEs.

Key words: ordinay differential equations (ODEs), initial value problems (IVPs), boundary value problems (BVPs), parallel factorizations, linear systems, sparse matrices, parallel solution, "parareal" algorithm.

1. Introduction. The numerical solution of ODEs requires the solution of sparse and structured linear systems. The parallel solution of these problems may be obtained in two ways: for BVPs, since the size of the associated linear system is large, we need to develop parallel solvers for the obtained linear systems; for IVPs we need to define appropriate numerical methods that allow to obtain "parallelizzable" linear systems.

In both cases, the main problem can then be taken back to the solution of special sparse linear systems, whose solution is here approached through the use of *parallel factorizations*, originally introduced for deriving efficient parallel tridiagonal solvers [2, 9], and subsequently generalized to block tridiagonal, Almost Block Diagonal (ABD), and Bordered Almost Block Diagonal (BABD) systems [10, 11, 15, 16, 17, 19].

With this premise, the structure of the paper is the following: in Section 2 the main facts about *parallel factorizations* and their extensions are briefly recalled; then, in Section 3 their application for solving ODE problems is sketched; finally, in Section 4 we show that this approach also encompasses the so called "Parareal" algorithm, recently introduced in [23, 24].

2. Parallel factorizations. In this section we consider several parallel algorithms in the class of partition methods for the solution of linear systems,

$$Ax = f, \tag{2.1}$$

where A is a $n \times n$ sparse and structured matrix, and x and f are vectors of length n. We will investigate the parallel solution of (2.1) on p processors, supposing $p \ll n$ in order for the number of sequential operations to be much smaller than that of parallel ones.

The coefficient matrices A here considered are (block) banded, tridiagonal, bidiagonal, or even Almost Block Diagonal (ABD). All these structures may be rearranged in the form

$$A = \begin{pmatrix} A^{(1)} & \boldsymbol{c}_1^{(1)} & & & & & \\ \boldsymbol{b}_1^{(1)^T} & a^{(1)} & \boldsymbol{c}_0^{(2)^T} & & & & \\ & \boldsymbol{b}_0^{(2)} & A^{(2)} & \boldsymbol{c}_1^{(2)} & & & \\ & & \boldsymbol{b}_1^{(2)^T} & a^{(2)} & & & \\ & & & & \ddots & & \\ & & & & & a^{(p-1)} & \boldsymbol{c}_0^{(p)^T} \\ & & & & & \boldsymbol{b}_0^{(p)} & A^{(p)} \end{pmatrix} \tag{2.2}$$

where the diagonal blocks are square and the superscript (i) indicates that this block is handled only by processor i. The size of the blocks $a^{(i)}$, $A^{(i)}$, $\boldsymbol{b}_j^{(i)}$, and $\boldsymbol{c}_j^{(i)}$ is in general independent of both i and j, and only depends on the sparsity structure of the coefficient matrix A. In particular, the size of the blocks $a^{(i)}$ is quite important, since the sequential section of the algorithm is proportional to it. Therefore, the blocks $a^{(i)}$ should be as small as possible. As an example, if A is (block) tridiagonal, $a^{(i)}$ reduces to a single (block) entry. Vice versa, in case of banded (block) matrices, the (block) size of $a^{(i)}$ equals to $\max(s, r)$, where s and r denote

*Work developed within the project "Numerical methods and software for differential equations"
†Dipartimento di Matematica, Università di Bari, Bari, Italy (amodio@dm.uniba.it)
‡Dipartimento di Matematica, Università di Firenze, Firenze, Italy (luigi.brugnano@unifi.it)

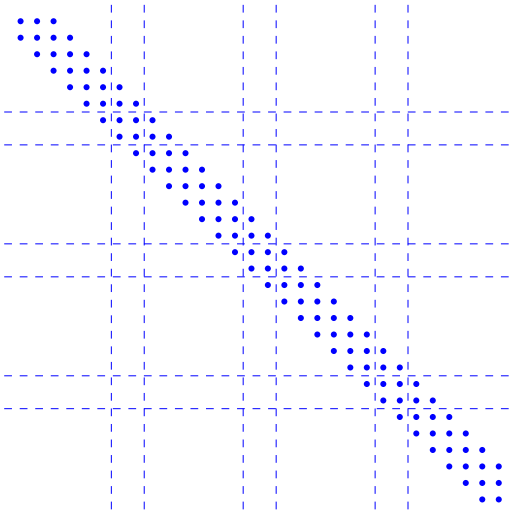


FIG. 2.1. *Partitioning of a banded matrix. Each point represents a (block) entry of the matrix.*

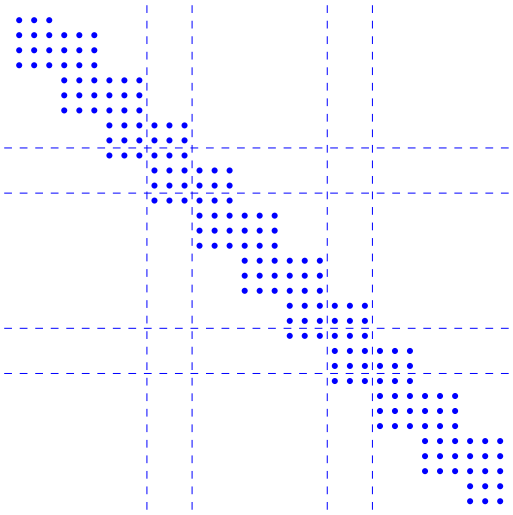


FIG. 2.2. *Partitioning of an ABD matrix. Each point represents an entry of the matrix.*

the number of lower and upper off (block) diagonals (see Figure 2.1), respectively. In case of ABD matrices, $a^{(i)}$ is a block of size equal to the number of rows in each block row of the coefficient matrix (see Figure 2.2). Since row and column permutations inside each block do not destroy the sparsity structure of the coefficient matrix, in ABD matrices we may permute the elements inside $a^{(i)}$ to improve stability properties. Blocks $A^{(i)}$ have the same sparsity structure as the original matrix, and are locally handled by using any suitable sequential algorithm.

In order to keep track of any parallel algorithm, we consider the following factorization [2, 9]

$$A = F\,T\,G, \tag{2.3}$$

where

$$F = \begin{pmatrix} N^{(1)} & \boldsymbol{o} & & & & & & \\ \boldsymbol{v}^{(1)^T} & I & \boldsymbol{w}^{(2)^T} & & & & & \\ & \boldsymbol{o} & N^{(2)} & \boldsymbol{o} & & & & \\ & & \boldsymbol{v}^{(2)^T} & I & \boldsymbol{w}^{(3)^T} & & & \\ & & & \boldsymbol{o} & N^{(3)} & \boldsymbol{o} & & \\ & & & & \boldsymbol{v}^{(3)^T} & I & & \\ & & & & & & \ddots & \\ & & & & & & I & \boldsymbol{w}^{(p)^T} \\ & & & & & & \boldsymbol{o} & N^{(p)} \end{pmatrix}, \tag{2.4}$$

$$T = \begin{pmatrix} \hat{I} & \boldsymbol{o} & & & & & & \\ \boldsymbol{o}^T & \alpha^{(1)} & \boldsymbol{o}^T & \gamma^{(2)} & & & & \\ & \boldsymbol{o} & \hat{I} & \boldsymbol{o} & & & & \\ & \beta^{(2)} & \boldsymbol{o}^T & \alpha^{(2)} & \boldsymbol{o}^T & \gamma^{(3)} & & \\ & & & \boldsymbol{o} & \hat{I} & \boldsymbol{o} & & \\ & & & \beta^{(3)} & \boldsymbol{o}^T & \alpha^{(3)} & & \\ & & & & & & \ddots & \\ & & & & & & \alpha^{(p-1)} & \boldsymbol{o}^T \\ & & & & & & \boldsymbol{o} & \hat{I} \end{pmatrix}, \tag{2.5}$$

$$G = \begin{pmatrix} S^{(1)} & \boldsymbol{y}^{(1)} & & & & & & \\ \boldsymbol{o}^T & I & \boldsymbol{o}^T & & & & & \\ & \boldsymbol{z}^{(2)} & S^{(2)} & \boldsymbol{y}^{(2)} & & & & \\ & & \boldsymbol{o}^T & I & \boldsymbol{o}^T & & & \\ & & & \boldsymbol{z}^{(3)} & S^{(3)} & \boldsymbol{y}^{(3)} & & \\ & & & & \boldsymbol{o}^T & I & & \\ & & & & & & \ddots & \\ & & & & & & I & \boldsymbol{o}^T \\ & & & & & & \boldsymbol{z}^{(p)} & S^{(p)} \end{pmatrix}, \tag{2.6}$$

I, $\hat{I}$ and $\boldsymbol{o}$ are identity and null matrices of appropriate sizes, and $N^{(i)}S^{(i)}$ is any suitable factorization of the block $A^{(i)}$. The remaining entries of F, T, and G can be derived from (2.3) by direct identification.

Factorization (2.3) may be computed in parallel on the p processors. For simplicity, we analyze the factorization of the sub-matrix identified by the superscript (i) (with obvious differences for $i = 1$ and $i = p$)

$$M^{(i)} = \begin{pmatrix} 0 & \boldsymbol{c}_0^{(i)^T} & \\ \boldsymbol{b}_0^{(i)} & A^{(i)} & \boldsymbol{c}_1^{(i)} \\ & \boldsymbol{b}_1^{(i)^T} & a^{(i)} \end{pmatrix}. \tag{2.7}$$

Following (2.2)–(2.6) we have

$$M^{(i)} = \begin{pmatrix} I & \boldsymbol{w}^{(i)^T} & \\ \boldsymbol{o} & N^{(i)} & \boldsymbol{o} \\ & \boldsymbol{v}^{(i)^T} & I \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & \boldsymbol{o}^T & \gamma^{(i)} \\ \boldsymbol{o} & I & \boldsymbol{o} \\ \beta^{(i)} & \boldsymbol{o}^T & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} I & \boldsymbol{o}^T & \\ \boldsymbol{z}^{(i)} & S^{(i)} & \boldsymbol{y}^{(i)} \\ & \boldsymbol{o}^T & I \end{pmatrix}, \tag{2.8}$$

where $\alpha_2^{(i)} + \alpha_1^{(i+1)} = \alpha^{(i)}$, and the other entries are the same as defined in (2.4)–(2.6).

Consequently, by considering any given factorization for $A^{(i)}$, it is possible to derive corresponding parallel extensions of such factorizations, which cover most of the parallel algorithms in the class of partition methods. In particular, the following ones easily derive for matrices with well conditioned sub-blocks $A^{(i)}$ (this means, for example, that pivoting is unnecessary or does not destroy the sparsity structure):

- *LU factorization*, by setting in (2.8) $N^{(i)} = L^{(i)}$ and $S^{(i)} = U^{(i)}$, where $L^{(i)}U^{(i)}$ is the LU factorization of the matrix $A^{(i)}$. In this case, the (block) vectors $\boldsymbol{y}^{(i)}$ and $\boldsymbol{v}^{(i)}$ maintain the same sparsity structure as that of $\boldsymbol{c}_1^{(i)}$ and $\boldsymbol{b}_1^{(i)}$, respectively, while the vectors $\boldsymbol{z}^{(i)}$ and $\boldsymbol{w}^{(i)}$ are non-null fill-in (block) vectors, obtained by solving two triangular systems.

- *LUD factorization* (which derives from the Gauss-Jordan elimination algorithm), by setting in (2.8) $S^{(i)} = D^{(i)}$, a diagonal matrix, and

$$\begin{pmatrix} I & \boldsymbol{w}^{(i)^T} & \\ \boldsymbol{o} & N^{(i)} & \boldsymbol{o} \\ & \boldsymbol{v}^{(i)^T} & I \end{pmatrix} = \begin{pmatrix} I & \boldsymbol{o}^T & \\ \boldsymbol{o} & L^{(i)} & \boldsymbol{o} \\ & \boldsymbol{v}^{(i)^T} & I \end{pmatrix} \begin{pmatrix} I & \boldsymbol{w}^{(i)^T} & \\ \boldsymbol{o} & U^{(i)} & \boldsymbol{o} \\ & \boldsymbol{o}^T & I \end{pmatrix},$$

  where $L^{(i)}$ and $U^{(i)}$ are lower and upper triangular matrices, respectively, with unit diagonal. Therefore, $\boldsymbol{v}^{(i)}$ and $\boldsymbol{w}^{(i)}$ maintain the same sparsity structure as that of $\boldsymbol{b}_1^{(i)}$ and $\boldsymbol{c}_0^{(i)}$, respectively, while $\boldsymbol{z}^{(i)}$ and $\boldsymbol{y}^{(i)}$ are non-null fill-in (block) vectors.

- *cyclic reduction* algorithm [2, 9] (see also [1, 12, 15, 16]), which is one of the best known parallel algorithms but that, in its original form, requires a synchronization at each step of reduction. In fact, the idea of this algorithm is to perform several reductions that, at each step, halve the size of the system. On the other hand, to obtain a factorization in the form (2.8), it is possible to consider cyclic reduction as a sequential algorithm to be applied locally,

$$M^{(i)} = (\hat{P}_1^{(i)}\hat{L}_1^{(i)}\hat{P}_2^{(i)}\hat{L}_2^{(i)}\cdots)\hat{D}^{(i)}(\cdots\hat{U}_2^{(i)}\hat{P}_2^{(i)^T}\hat{U}_1^{(i)}\hat{P}_1^{(i)^T}),$$

  where $\hat{P}_i^{(i)}$ are suitable permutation matrices that maintain the first and last row in the reduced matrix. The computational cost, which is much higher if the algorithm is applied to A on a sequential computer, becomes comparable to the previous local factorizations since this algorithm does not compute fill-in block vectors. As a consequence, the corresponding parallel factorization algorithm turns out to be one of the most effective.

- *Alternate row and column elimination* [25] which is an algorithm suitable for ABD matrices. In fact, for such matrices alternate row and column permutations always guarantee stability without fill-in. This feature extends to the parallel algorithm, by taking into account that row permutations between the first block row of $A^{(i)}$ and the block containing $\boldsymbol{c}_0^{(i)}$ (see (2.7)), still make the parallel algorithm stable without introducing fill-in. Such parallel factorization is defined by setting $N^{(i)} = P^{(i)}L^{(i)}$ and $S^{(i)} = U^{(i)}Q^{(i)}$, where $P^{(i)}$ and $Q^{(i)}$ are permutation matrices and $L^{(i)}$ and $U^{(i)}$, after a suitable reordering of the rows and of the columns, are 2×2 block triangular matrices (see [10] for full details). Finally, the (block) vectors $\boldsymbol{y}^{(i)}$ and $\boldsymbol{z}^{(i)}$ maintain the same sparsity structure as that of $\boldsymbol{b}_1^{(i)}$ and $\boldsymbol{c}_0^{(i)}$, respectively, whereas $\boldsymbol{w}^{(i)}$ and $\boldsymbol{v}^{(i)}$ are fill-in (block) vectors.

For what concerns the solution of the systems associated to the previous parallel factorizations, there is much parallelism inside. The solution of the systems with the matrices F and G may proceed in parallel on the different processors. Conversely, the solution of the system with the matrix T requires a sequential part, consisting in the solution of a *reduced system* with the (block) tridiagonal *reduced matrix*

$$T_p = \begin{pmatrix} \alpha^{(1)} & \gamma^{(2)} & & \\ \beta^{(2)} & \alpha^{(2)} & \ddots & \\ & \ddots & \ddots & \gamma^{(p-1)} \\ & & \beta^{(p-1)} & \alpha^{(p-1)} \end{pmatrix}. \tag{2.9}$$

We observe that the (block) size of T_p only depends on p and is independent of n.

For a matrix A with singular or ill-conditioned sub-blocks $A^{(i)}$, the local factorizations may be unstable or even undefined. Consequently, it is necessary to slightly modify the factorization (2.8), in order to obtain stable parallel algorithms. The basic idea is that factorization (2.8) may produce more than two entries in the *reduced system*. In other words, the factorization of $A^{(i)}$ is stopped when the considered sub-block is ill-conditioned (or

the local factorization with a singular factor). As a consequence, the size of the *reduced system* is increased as sketched below. Let then

$$M^{(i)} = \hat{L}_1^{(i)} \hat{D}_1^{(i)} \hat{U}_1^{(i)},$$

where

$$\hat{L}_1^{(i)} = \begin{pmatrix} I & \boldsymbol{w}_1^{(i)^T} & & & \\ \boldsymbol{o} & N_1^{(i)} & \boldsymbol{o} & & \\ & \boldsymbol{v}_1^{(i)^T} & I & \boldsymbol{o}^T & \\ & & \boldsymbol{o} & \hat{I} & \boldsymbol{o} \\ & & & \boldsymbol{o}^T & I \end{pmatrix}, \quad \hat{U}_1^{(i)} = \begin{pmatrix} I & \boldsymbol{o}^T & & & \\ \boldsymbol{z}_1^{(i)} & S_1^{(i)} & \boldsymbol{y}_1^{(i)} & & \\ & \boldsymbol{o}^T & I & \boldsymbol{o}^T & \\ & & \boldsymbol{o} & \hat{I} & \boldsymbol{o} \\ & & & \boldsymbol{o}^T & I \end{pmatrix},$$

$$\hat{D}_1^{(i)} = \begin{pmatrix} \alpha_1^{(i)} & \boldsymbol{o}^T & \gamma_1^{(i)} & & \\ \boldsymbol{o} & I & \boldsymbol{o} & & \\ \beta_1^{(i)} & \boldsymbol{o}^T & \alpha_2^{(i)} & \boldsymbol{c}_2^{(i)^T} & \\ & & \boldsymbol{b}_2^{(i)} & A_2^{(i)} & \boldsymbol{c}_3^{(i)} \\ & & & \boldsymbol{b}_3^{(i)^T} & \alpha_3^{(i)} \end{pmatrix},$$

when the sub-block $A_1^{(i)}$ of $A^{(i)}$,

$$A_1^{(i)} = \begin{pmatrix} N_1^{(i)} & \\ \boldsymbol{v}_1^{(i)^T} & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} S_1^{(i)} & \boldsymbol{y}_1^{(i)} \\ & I \end{pmatrix},$$

is singular, because the block $\alpha_2^{(i)}$ is singular (i. e., $\alpha_2^{(i)} = 0$, in the scalar case). Then, $\alpha_2^{(i)}$ is introduced in the *reduced system*. By iterating this procedure on $\hat{D}_1^{(i)}$, we obtain again the factorization (2.3), with the only difference that now the *reduced matrix* in T_p may be of (block) size larger than $p-1$ (compare with (2.9)). However, it may be shown that it still depends only on p, whereas it is independent of n [3, 4]. Consequently, the scalar section of the whole algorithm is still negligible, when $n \gg p$.

The parallel algorithms that fall in this class are [3, 4]:

- *LU factorization with partial pivoting*, defined by setting $N_1^{(i)} = (P_1^{(i)})^T L_1^{(i)}$ and $S_1^{(i)} = U_1^{(i)}$ where $P_1^{(i)}$ is a permutation matrix such that $L_1^{(i)} U_1^{(i)}$ is the LU factorization of $P_1^{(i)} A_1^{(i)}$. The remaining (block) vectors are defined similarly as in the case of the LU factorization previously described.
- *QR factorization*, defined by setting $N_1^{(i)} = Q_1^{(i)}$ and $S_1^{(i)} = R_1^{(i)}$. In this case both $\boldsymbol{w}_1^{(i)}$ and $\boldsymbol{z}_1^{(i)}$ are fill-in (block) vectors while $\boldsymbol{v}_1^{(i)}$ and $\boldsymbol{y}_1^{(i)}$ maintain the same sparsity structure as that of the corresponding (block) vectors in $M^{(i)}$.

Factorization (2.3)–(2.6), and the corresponding parallel algorithms mentioned above, are easily generalized to matrices with additional non-null elements in the right-lower and/or left-upper corners. This is the case, for example, of Bordered ABD (BABD) matrices (see Figure 2.3) and matrices with a circulant-like structure (see Figure 2.4). Supposing the non-null elements are located in the right-upper corner (this is always possible by means of suitable permutation), then the coefficient matrix is partitioned in the form

$$A = \begin{pmatrix} a^{(0)} & \boldsymbol{c}_0^{(1)^T} & & & & & & & b \\ \boldsymbol{b}_0^{(1)} & A^{(1)} & \boldsymbol{c}_1^{(1)} & & & & & & \\ & \boldsymbol{b}_1^{(1)^T} & a^{(1)} & \boldsymbol{c}_0^{(2)^T} & & & & & \\ & & \boldsymbol{b}_0^{(2)} & A^{(2)} & \boldsymbol{c}_1^{(2)} & & & & \\ & & & \boldsymbol{b}_1^{(2)^T} & a^{(2)} & & & & \\ & & & & & \ddots & & & \\ & & & & & & a^{(p-1)} & \boldsymbol{c}_0^{(p)^T} & \\ & & & & & & \boldsymbol{b}_0^{(p)} & A^{(p)} & \boldsymbol{c}_1^{(p)} \\ & & & & & & & \boldsymbol{b}_1^{(p)^T} & a^{(p)} \end{pmatrix}, \tag{2.10}$$

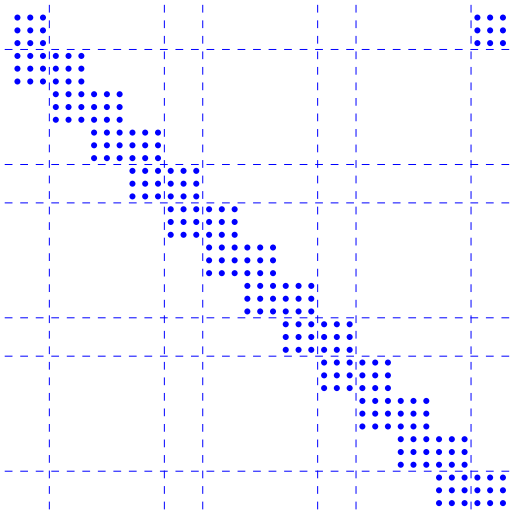


FIG. 2.3. *Partitioning of a BABD matrix. Each point represents an entry of the matrix.*

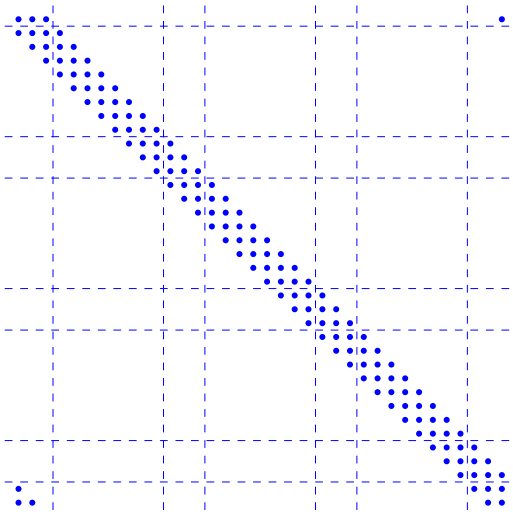


FIG. 2.4. *Partitioning of a matrix with a circulant-like structure. Each point represents a (block) entry of the matrix.*

where b is the smallest rectangular block containing all the corner elements.

A factorization similar to that in (2.3)–(2.6) (the obvious differences are related to the first and last (block) rows) produces a corresponding *reduced system* with the *reduced matrix*

$$T_p = \begin{pmatrix} \alpha^{(0)} & \gamma^{(1)} & & & \beta^{(0)} \\ \beta^{(1)} & \alpha^{(1)} & \gamma^{(2)} & & \\ & \beta^{(2)} & \alpha^{(2)} & \ddots & \\ & & \ddots & \ddots & \gamma^{(p)} \\ & & & \beta^{(p)} & \alpha^{(p)} \end{pmatrix}. \tag{2.11}$$

We observe that, for the very important classes of BABD and circulant-like matrices (the latter, after a suitable row permutation, see Figure 2.5), both the matrix (2.10) and the *reduced matrix* (2.11) have the form of a lower block bidiagonal matrix (i. e., $\boldsymbol{c}_j^{(i)} = 0$ and $\gamma^{(i)} = 0$ for all i and j) with an additional right-upper

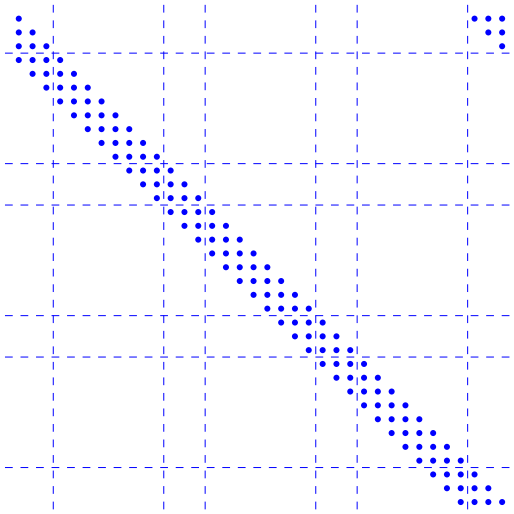


FIG. 2.5. *Partitioning of the matrix in Fig. 2.4 after row permutation. Each point represents a (block) entry of the matrix.*

corner block:

$$A = \begin{pmatrix} a^{(0)} & & & & & & b \\ \boldsymbol{b}_0^{(1)} & A^{(1)} & & & & & \\ & \boldsymbol{b}_1^{(1)^T} & a^{(1)} & & & & \\ & & \boldsymbol{b}_0^{(2)} & \ddots & & & \\ & & & \ddots & a^{(p-1)} & & \\ & & & & \boldsymbol{b}_0^{(p)} & A^{(p)} & \\ & & & & & \boldsymbol{b}_1^{(p)^T} & a^{(p)} \end{pmatrix},$$

and

$$T_p = \begin{pmatrix} \alpha^{(0)} & & & \beta^{(0)} \\ \beta^{(1)} & \alpha^{(1)} & & \\ & \ddots & \ddots & \\ & & \beta^{(p)} & \alpha^{(p)} \end{pmatrix}.$$

We have also to note that, for this kind of matrices, the overall computational cost of a parallel factorization algorithm has a very small increase. On a sequential machine, supposing to maintain the same partitioning of the matrix on $p > 1$ processors, we have a computational cost which is similar to that of any efficient sequential algorithm (the corner block b implies the construction of a fill-in (block) vector) but with better stability properties (see [26]). For this reason, a method that is widely and efficiently applied to matrices in the form (2.10), also on a sequential computer, is cyclic reduction (see [11, 17, 19] where cyclic reduction is applied to BABD matrices).

3. Parallel solution of differential equations. The numerical solution of ODE-BVPs leads to the solution of large and sparse linear systems of equations that represent the most expensive part of a BVP code. The sparsity structure of the obtained problem depends on the methods implemented. In general, one-step methods lead to ABD or BABD matrices, depending on the boundary conditions (separated or not, respectively), while multistep methods lead to block banded systems (with additional corner blocks in case of non-separated boundary conditions) [5, 6, 22]. The parallel algorithms previously described perfectly cope with this kind of systems. For this reason we do not investigate further on ODE-BVPs.

Conversely, we shall now consider the application of *parallel factorizations* for deriving parallel algorithms for numerically solving ODE-IVPs, which we assume, for sake of simplicity, to be linear and in the form

$$y' = Ly + g(t), \qquad t \in [t_0, T], \qquad y(t_0) = y_0 \in \mathbb{R}^m, \tag{3.1}$$

which is, however, sufficient to grasp the main features of the approach [5, 6, 7, 8, 20, 21, 22].

Let us consider a suitable *coarse mesh*, defined by the following partition of the integration interval in (3.1):

$$t_0 \equiv \tau_0 < \tau_1 < \cdots < \tau_p \equiv T. \tag{3.2}$$

Suppose, for simplicity, that inside each sub-interval we apply a given method with a constant stepsize

$$h_i = \frac{\tau_i - \tau_{i-1}}{N}, \qquad i = 1, \ldots, p, \tag{3.3}$$

to approximate the problem

$$y' = Ly + g(t), \quad t \in [\tau_{i-1}, \tau_i], \qquad y(\tau_{i-1}) = y_{0i}, \qquad i = 1, \ldots, p. \tag{3.4}$$

If $y(t)$ denotes the solution of problem (3.1), and we denote by

$$y_{ni} \approx y(\tau_{i-1} + nh_i), \qquad n = 0, \ldots, N, \quad i = 1, \ldots, p, \tag{3.5}$$

the entries of the discrete approximation, then, in order for the numerical solutions of (3.1) and (3.4) to be equivalent, we require that (see (3.2) and (3.5))

$$y_{01} = y_0, \qquad y_{0i} \equiv y_{N,i-1}, \quad i = 2, \ldots, p. \tag{3.6}$$

For convention, we also set

$$y_{01} \equiv y_{N0}. \tag{3.7}$$

Let now suppose that the numerical approximations to the solutions of (3.4) are obtained by solving discrete problems in the form

$$M_i \boldsymbol{y}_i = \boldsymbol{v}_i y_{0i} + \boldsymbol{g}_i, \qquad \boldsymbol{y}_i = \begin{pmatrix} y_{1i}, & \ldots & , & y_{Ni} \end{pmatrix}^T, \qquad i = 1, \ldots, p, \tag{3.8}$$

where the matrices $M_i \in \mathbb{R}^{mN \times mN}$ and $\boldsymbol{v}_i \in \mathbb{R}^{mN \times m}$, and the vector $\boldsymbol{g}_i \in \mathbb{R}^{mN}$, do depend on the chosen method (see, e.g., [5, 6], for the case of block BVMs) and on the problems (3.4). Clearly, this is a quite general framework, which encompasses most of the currently available methods for solving ODE-IVPs. By taking into account all the above facts, one obtains that the global approximation to the solution of (3.1) is obtained by solving a discrete problem in the form (hereafter, I_r will denote the identity matrix of dimension r):

$$M\boldsymbol{y} \equiv \begin{pmatrix} I_m & & & & \\ -\boldsymbol{v}_1 & M_1 & & & \\ & -V_2 & M_2 & & \\ & & \ddots & \ddots & \\ & & & -V_p & M_p \end{pmatrix} \begin{pmatrix} y_{N0} \\ \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_p \end{pmatrix} = \begin{pmatrix} y_0 \\ \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_p \end{pmatrix}, \tag{3.9}$$

$$V_i = [O \,|\, \boldsymbol{v}_i] \in \mathbb{R}^{mN \times mN}, \qquad i = 2, \ldots, p.$$

Obviously, this problem may be solved in a sequential fashion, by means of the iteration (see (3.6)-(3.7)):

$$y_{N0} = y_0, \qquad M_i \boldsymbol{y}_i = \boldsymbol{g}_i + \boldsymbol{v}_i y_{N,i-1}, \qquad i = 1, \ldots, p.$$

Nevertheless, by following arguments similar to those in the previous section, we consider the factorization:

$$M = \begin{pmatrix} I_m & & & & \\ & M_1 & & & \\ & & M_2 & & \\ & & & \ddots & \\ & & & & M_p \end{pmatrix} \begin{pmatrix} I_m & & & & \\ -\boldsymbol{w}_1 & I_{mN} & & & \\ & -W_2 & I_{mN} & & \\ & & \ddots & \ddots & \\ & & & -W_p & I_{mN} \end{pmatrix},$$

where (see (3.9))

$$W_i = [O \,|\, \boldsymbol{w}_i] \in \mathbb{R}^{mN \times mN}, \qquad \boldsymbol{w}_i = M_i^{-1} \boldsymbol{v}_i \in \mathbb{R}^{mN \times m}. \tag{3.10}$$

Consequently, at first we solve, in parallel, the systems

$$M_i \boldsymbol{z}_i = \boldsymbol{g}_i, \qquad \boldsymbol{z}_i = \left(\; z_{1i}, \; \ldots \; , \; z_{Ni} \; \right)^T, \qquad i = 1, \ldots, p, \tag{3.11}$$

and, then, (see (3.10) and (3.6)) recursively update the local solutions,

$$\begin{aligned} &\boldsymbol{y}_1 = \boldsymbol{z}_1 + \boldsymbol{w}_1 y_{01}, \\ &\boldsymbol{y}_i = \boldsymbol{z}_i + W_i \boldsymbol{y}_{i-1} \equiv \boldsymbol{z}_i + \boldsymbol{w}_i y_{0i}, \qquad i = 2, \ldots, p. \end{aligned} \tag{3.12}$$

The latter recursion, however, has still much parallelism. Indeed, if we consider the partitionings (see (3.8), (3.11), and (3.10))

$$\boldsymbol{y}_i = \begin{pmatrix} \widehat{\boldsymbol{y}}_i \\ y_{Ni} \end{pmatrix}, \qquad \boldsymbol{z}_i = \begin{pmatrix} \widehat{\boldsymbol{z}}_i \\ z_{Ni} \end{pmatrix}, \qquad \boldsymbol{w}_i = \begin{pmatrix} \widehat{\boldsymbol{w}}_i \\ w_{Ni} \end{pmatrix}, \quad w_{Ni} \in \mathbb{R}^{m \times m}, \tag{3.13}$$

then (3.12) is equivalent to solve, at first, the *reduced system*

$$\begin{pmatrix} I_m & & & \\ -w_{N1} & I_m & & \\ & \ddots & \ddots & \\ & & -w_{N,p-1} & I_m \end{pmatrix} \begin{pmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0p} \end{pmatrix} = \begin{pmatrix} y_0 \\ z_{N1} \\ \vdots \\ z_{N,p-1} \end{pmatrix}, \tag{3.14}$$

i. e.,

$$y_{01} = y_0, \qquad y_{0,i+1} = z_{Ni} + w_{Ni} y_{0i}, \qquad i = 1, \ldots, p-1, \tag{3.15}$$

after which performing the p parallel updates

$$\widehat{\boldsymbol{y}}_i = \widehat{\boldsymbol{z}}_i + \widehat{\boldsymbol{w}}_i y_{0i}, \qquad i = 1, \ldots, p-1, \qquad \boldsymbol{y}_p = \boldsymbol{z}_p + \boldsymbol{w}_p y_{0p}. \tag{3.16}$$

We observe that:

- the parallel solution of the p systems in (3.11) is equivalent to compute the approximate solution of the following p ODE-IVPs,

$$z' = Lz + g(t), \quad t \in [\tau_{i-1}, \tau_i], \quad z(\tau_{i-1}) = 0, \quad i = 1, \ldots, p, \tag{3.17}$$

  in place of the corresponding ones in (3.4);
- the solution of the *reduced system* (3.14)-(3.15) consists in computing the proper initial values $\{y_{0i}\}$ for the previous ODE-IVPs;
- the parallel updates (3.16) update the approximate solutions of the ODE-IVPs (3.17) to those of the corresponding ODE-IVPs in (3.4).

REMARK 1. *Clearly, the solution of the first (parallel) system in (3.11) and the first (parallel) update in (3.12) (see also (3.16)) can be executed together, by solving the linear system (see (3.6))*

$$M_1 \boldsymbol{y}_1 = \boldsymbol{g}_1 + \boldsymbol{v}_1 y_0, \tag{3.18}$$

thus directly providing the final discrete approximation on the first processor; indeed, this is possible, since the initial condition y_0 is given.

We end this section by emphasizing that one obtains an almost perfect parallel speed-up, if p processors are used, provided that the cost for the solution of the *reduced system* (3.14) and of the parallel updates (3.16) is small, with respect to that of (3.11) (see [5, 6] for more details). This is, indeed, the case when the parameter N in (3.3) is large enough and the coarse partition (3.2) can be supposed to be *a priori* given.

4. Connections with the "Parareal" algorithm. We now briefly describe the "Parareal" algorithm introduced in [23, 24], showing the existing connections with the parallel method previously described. This method, originally defined for solving PDE problems, for example linear or quasi-linear parabolic problems, can be directly cast into the ODE setting via the semi-discretization of the space variables; that is, by using the method of lines. In more detail, let us consider the problem

$$\frac{\partial}{\partial t} y = \mathcal{L}\, y, \qquad t \in [t_0, T], \qquad y(t_0) = y_0, \tag{4.1}$$

where $\mathcal{L}$ is an operator from a Hilbert space V into V'. Let us consider again the partition (3.2) of the time interval, and consider the problems

$$\frac{\partial}{\partial t} y = \mathcal{L}\, y, \qquad t \in [\tau_{i-1}, \tau_i], \qquad y(\tau_{i-1}) = y_{0i}, \qquad i = 1, \ldots, p. \tag{4.2}$$

Clearly, in order for (4.1) and (4.2) to be equivalent, one must require that

$$y_{0i} = y(\tau_{i-1}), \qquad i = 1, \ldots, p. \tag{4.3}$$

The initial data (4.3) are then formally related by means of suitable *propagators* $\mathcal{F}_i$ such that

$$y_{0,i+1} = \mathcal{F}_i y_{0i}, \qquad i = 1, \ldots, p-1. \tag{4.4}$$

The previous relations can be cast in matrix form as ($\mathcal{I}$ now denotes the identity operator)

$$F\boldsymbol{y} \equiv \begin{pmatrix} \mathcal{I} & & & \\ -\mathcal{F}_1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\mathcal{F}_{p-1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0p} \end{pmatrix} = \begin{pmatrix} y_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \equiv \boldsymbol{\eta}. \tag{4.5}$$

For solving (4.5), the authors essentially define the splitting

$$F = (F - G) + G, \qquad G = \begin{pmatrix} \mathcal{I} & & & \\ -\mathcal{G}_1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\mathcal{G}_{p-1} & \mathcal{I} \end{pmatrix},$$

with *coarse propagators*

$$\mathcal{G}_i \approx \mathcal{F}_i, \qquad i = 1, \ldots, p,$$

and consider the iterative procedure

$$G\boldsymbol{y}^{(k+1)} = (G - F)\boldsymbol{y}^{(k)} + \boldsymbol{\eta}, \qquad k = 0, 1, \ldots,$$

with an obvious meaning of the upper index. This is equivalent to solve the problems

$$\begin{aligned} &y_{01}^{(k+1)} = y_0, \\ &y_{0,i+1}^{(k+1)} = \mathcal{G}_i y_{0i}^{(k+1)} + (\mathcal{F}_i - \mathcal{G}_i) y_{0i}^{(k)}, \qquad i = 1, \ldots, p-1, \end{aligned} \tag{4.6}$$

thus providing good parallel features, if we can assume that the coarse operators $\mathcal{G}_i$ are "cheap" enough. The iteration (4.6) defines the "Parareal" algorithm, which is iterated until

$$\|y_{0i}^{(k+1)} - y_{0i}^{(k)}\|, \qquad i = 2, \ldots, p,$$

are suitably small. In the practice, in case of linear operators, problem (4.1) becomes, via the method of lines, an ODE in the form (3.1), with L a huge and very sparse matrix. Consequently, problems (4.2) become in the form (3.4). Similarly, the propagator $\mathcal{F}_i$ consists in the application of a suitable discrete method for approximating the solution of the corresponding ith problem in (3.4), and the coarse propagator $\mathcal{G}_i$ describes the application of a much cheaper method for solving the same problem. As a consequence, if the discrete problems corresponding to the propagators $\{\mathcal{F}_i\}$ are in the form (3.8), then the discrete version of the recurrence (4.4) becomes exactly (3.15), as well as the discrete counterpart of the matrix form (4.5) becomes (3.14).

We can then conclude that the "Parareal" algorithm in [23, 24] *exactly* coincides with the iterative solution of the *reduced system* (3.14), induced by a suitable splitting.

We observe that the previous iterative procedure may be very appropriate, when the matrix L is large and sparse since, in this case, the computations of the block vectors $\{\boldsymbol{w}_i\}$ in (3.10), and then of the matrices $\{w_{Ni}\}$ (see (3.13)) would be clearly impractical. Moreover, it can be considerably improved by observing that

$$w_{Ni} y_{0i} \approx \mathrm{e}^{(\tau_i - \tau_{i-1})L} y_{0i}.$$

Consequently, by considering a suitable approximation to the matrix exponential, the corresponding parallel algorithm turns out to become semi-iterative and potentially very effective, as recently shown in [8].

REFERENCES

[1] P. Amodio. Optimized cyclic reduction for the solution of linear tridiagonal systems on parallel computers. *Comput. Math. Appl.* **26** (1993) 45–53.

[2] P. Amodio and L. Brugnano. Parallel factorizations and parallel solvers for tridiagonal linear systems. *Linear Algebra Appl.* **172** (1992) 347–364.

[3] P. Amodio and L. Brugnano, The parallel QR factorization algorithm for tridiagonal linear systems. *Parallel Comput.* **21** (1995) 1097–1110.

[4] P. Amodio and L. Brugnano. Stable Parallel Solvers for General Tridiagonal Linear Systems. *Z. Angew. Math. Mech.* **76**, suppl. 1 (1996) 115–118.
[5] P. Amodio and L. Brugnano. Parallel implementation of block boundary value methods for ODEs, *J. Comput. Appl. Math.* **78** (1997) 197–211.
[6] P. Amodio and L. Brugnano. Parallel ODE solvers based on block BVMs, *Adv. Comput. Math.* **7** (1997) 5–26.
[7] P. Amodio and L. Brugnano. ParalleloGAM: a Parallel Code for ODEs, *Appl. Numer. Math.* **28** (1998) 95–106.
[8] P. Amodio and L. Brugnano. Parallel solution in time of ODEs: some achievements and perspectives. *Appl. Numer. Math.* **59** (2009) 424–435.
[9] P. Amodio, L. Brugnano and T. Politi. Parallel factorizations for tridiagonal matrices. *SIAM J. Numer. Anal.* **30** (1993) 813–823.
[10] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut and M. Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numer. Linear Algebra Appl.* **7**, no. 5 (2000) 275–317.
[11] P. Amodio, I. Gladwell and G. Romanazzi. Numerical solution of general Bordered ABD linear systems by cyclic reduction. *JNAIAM J. Numer. Anal. Ind. Appl. Math.* **1**, no. 1 (2006) 5–12.
[12] P. Amodio, N. Mastronardi. A parallel version of the cyclic reduction algorithm on a Hypercube. *Parallel Comput.* **19** (1993) 1273–1281.
[13] P. Amodio and F. Mazzia. A parallel Gauss-Seidel method for block tridiagonal linear systems. *SIAM J. Sci. Comput.* **16** (1995) 1451–1461.
[14] P. Amodio and F. Mazzia. Parallel iterative solvers for banded linear systems. *Lecture Notes in Comput. Sci.* **1196**, (Numerical Analysis and its Applications. L. Vulkov, J. Wàsniewski, and P. Yalamov editors, Springer, Berlin), 17–24, 1997.
[15] P. Amodio and M. Paprzycki. Parallel solution of Almost Block Diagonal systems on a Hypercube. *Linear Algebra Appl.* **241-243** (1996) 85–103.
[16] P. Amodio and M. Paprzycki. On the parallel solution of Almost Block Diagonal systems. *Control Cybernet.* **25**, no. 3 (1996) 645–656.
[17] P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comput.* **18**, no. 1 (1997) 56–68.
[18] P. Amodio, M. Paprzycki and T. Politi. A survey of parallel direct methods for block bidiagonal linear systems on distributed memory computers. *Comput. Math. Appl.* **31** (1996) 111–127.
[19] P. Amodio and G. Romanazzi. BABDCR: a Fortran 90 package for the solution of Bordered ABD systems. *ACM Trans. Math. Software* **32**, no. 4 (2006) 597–608. (Available on the url: `http://www.netlib.org/toms/859`)
[20] L. Brugnano and D. Trigiante. On the potentiality of sequential and parallel codes based on extended trapezoidal rules (ETRs), *Appl. Numer. Math.* **25** (1997) 169–184.
[21] L. Brugnano and D. Trigiante. Parallel implementation of block boundary value methods on nonlinear problems: theoretical results, *Appl. Numer. Math.* **78** (1997) 197–211.
[22] L. Brugnano and D. Trigiante. *Solving Differential Problems by Multistep Initial and Boundary Value Methods*, Gordon and Breach, Amsterdam, 1998.
[23] J. L. Lions, Y. Maday and G. Turinici. Résolution d'EDP par un schéma en temps "pararéel", *C. R. Acad. Sci. Paris, Série I* **332** (2001) 661–668.
[24] Y. Maday and G. Turinici. A parareal in time procedure for the control of partial differential equations, *C. R. Acad. Sci. Paris, Série I* **335** (2002) 387–392.
[25] J. M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.* **13** (1976) 71–75.
[26] S. J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Stat. Comput.* **14** (1993) 231–238.

Edited by: Francesca Mazzia
Received: June 27, 2009
Accepted: September 10, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 397–411. http://www.scpe.org
ISSN 1895-1767

MIRRORING INFORMATION WITHIN AN AGENT-TEAM-BASED INTELLIGENT GRID MIDDLEWARE; AN OVERVIEW AND DIRECTIONS FOR SYSTEM DEVELOPMENT

MARIA GANZHA, MARCIN PAPRZYCKI, MICHAL DROZDOWICZ*, MEHRDAD SENOBARI†, IVAN LIRKOV, SOFIYA IVANOVSKA‡, RICHARD OLEJNIK§, AND PAVEL TELEGIN¶

Abstract. This work concerns part of our project, devoted to the development of an agent-team-based Grid resource brokering and management system. Here, open issues that have to be addressed in the process, concern agent team preservation. In our earlier work it was suggested that this can be achieved through mirroring of key information. Here, we discuss in detail sources of useful information generated in the system (an agent team in particular) and consider which information should be mirrored, when and where, to increase long-term sustainability of an agent team.

1. Introduction. Our work is devoted to the development of an agent-team-based high-level intelligent Grid middleware. In our earlier work we have conceptualized a number of scenarios which require access to information that was generated within the team. For instance, it is assumed that when a team leader (the *LMaster* agent) receives (from an agent representing a *User*; the *LAgent*) a *Call for Proposals* (*CFP*) asking about conditions of executing a specific job, its response (an offer, or a rejection) should be based on its knowledge of the potential client (*User* that the *LAgent* represents), as well as of current market conditions (see, [20, 13]). Such knowledge is to be grounded in historical data collected during past interactions with potential client(s). For instance a series of unsuccessful bids for jobs may indicate that the suggested price is too high. Since we assume ([25]) that a global Grid is a highly dynamic structure, in which nodes can disappear practically without any notice, it is important to assure that the team knowledge will not be lost when its leader crashes. Therefore, in [14, 26] we have suggested that support for long-term existence of agent teams may be achieved through utilization of information *mirroring*. Specifically, we have proposed that in each team an *LMirror* agent should be created, and its role should be to store a copy of all information necessary to prevent team disintegration in the case of crash of the *LMaster*. However, thus far we have not delved into any details as to what needs to be mirrored, when and how.

The aim of this paper is to describe sources of information that can be useful for an agent team in the context of its survival. Furthermore, we discuss when such information should be mirrored and how. To this effect we proceed as follows. In the next section we present a brief overview of the proposed system, as well as arguments for the need of information mirroring. We follow with presentation of sources of information that is to be collected. In each case we discuss when and where this information has to be persisted. Finally, we discuss what consequences does utilization of an outsourced data warehouse (to mirror team data) on procedures involved in recovering crashed *LMaster* and *LMirror* agents. Discussion includes a proposal for the structure of such data warehouse. Material presented here extends results introduced in [19].

2. System overview. To present a high-level overview of the system as well as to discuss two of its main functionalities: (i) *Worker* joining a team, and (ii) team accepting a job to be executed; we will utilize an AML Social Diagram ([11]) in figure 2.1.

In the approach utilized in our system, agents work in teams. Each team is managed by its "leader," the *LMaster* agent. Agent teams utilize services of the *Client Information Center* (represented by the *CIC* agent), to advertise work they are ready to do and, possibly, *Workers* they are seeking. In addition to the *LMaster* and *Workers*, each team consists of an *LMirror* agent. This agent stores a copy of information necessary for the team to persist in case when the *LMaster* crashes.

When the *User* is seeking a team to execute its job, it specifies job constraints to its representative, the *LAgent*. The *LAgent* contacts the *CIC* to obtain the list of teams that can execute its job and utilizes trust information and the FIPA Contract Net protocol ([5]) to either find a team that can do the job, or establish that such team cannot be found (within the specified constraints).

*Systems Research Institute Polish Academy of Science, Warsaw, Poland
†Tarbiat Modares University, Tehran, Iran
‡Institute for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria
§University of Sciences and Technologies of Lille, Lille, France
¶SuperComputing Center Russian Academy of Sciences, Moscow, Russia

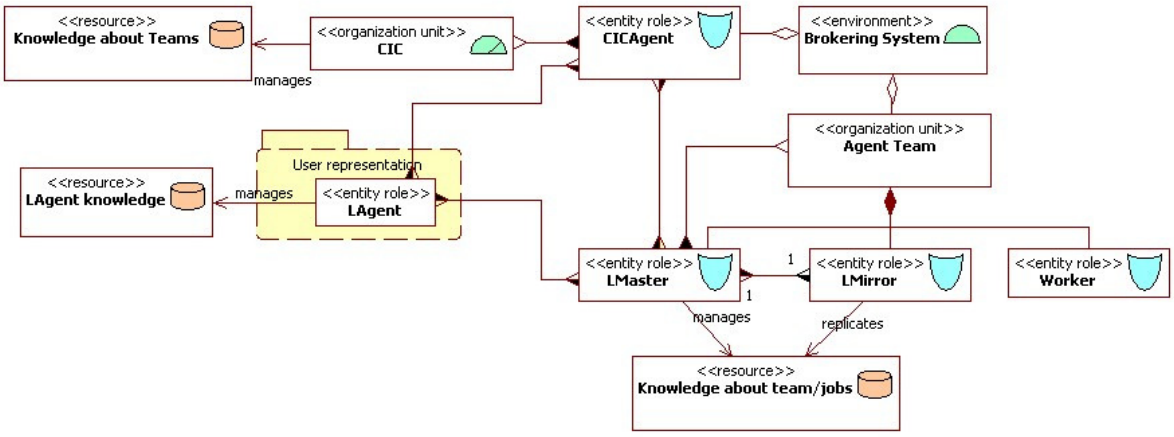


FIG. 2.1. *AML social diagram of the proposed system*

When the *User* would like to earn money by offering services of its computer(s), by becoming a *Worker* in a team, it specifies its conditions of joining (working for the team, e.g. the minimal "salary"). The *LAgent* interacts with the *CIC* to obtain a list of teams that are seeking *Workers* (satisfying characteristics of hardware and software it represents). Next, it utilizes trust information and the FIPA Contract Net protocol to establish if a team to join can be found. In both cases, trust information is a part of *LAgent*'s knowledge. At the same time, when responding to the CFP from *LAgents*, the *LMaster* utilizes the *Team knowledge* (more about *LMaster* responses can be found in [26, 20]).

3. Need for information mirroring. While issues involved in collecting knowledge by the *LAgent*, and utilizing it in decision making supporting the *User* are interesting, we focus our attention on processes taking place within the agent-team. This is especially so, since persisting information stored by the *LAgent* can be done by regularly backing-up the data (the same way as any *User* data should be backed-up).

Now, let us recall, that one of our key assumptions is that in the case of a *global Grid* any node can disappear practically without warning, and for an unspecified time (in the global Grid nodes are assumed to belong to individuals and thus, for all practical purposes, there are no guarantees of service). Obviously, this assumption has to be applied also to the *LMaster* agent (specifically, the node that it resides on). Let us now see what happens when the *LMaster* becomes unavailable and there is no mirroring in the team.

First, let us consider the case of a short-term disappearance of the *LMaster*. Since it is the "gateway agent" for the team, problems arising in this case involve, among others, contract negotiations (sending CFP to the *LMaster* or awaiting its response). Here, a short-term disappearance of the *LMaster* could result in loss of some potential contract (there would be no agent to send the CFP to, receive a response from, or confirm a contract with). Furthermore, since the *LMaster* is the gateway through which contracted jobs are sent to the team, and results sent back to *Users*, its short-term crash would result in some *Users* starting to believe that the team disintegrated. As a result, the reputation of the team would be damaged and some *Users* would not work with it again (these *Users* that were not able to contact the team during the *LMaster* crash). Recall that after the *CIC* delivers to the *User* the list of teams that potentially can complete its task, this list is pruned out of all teams that are not considered trustworthy.

Second, let us consider the fact that the *LMaster* is the only agent in the team that *Workers* know about (we assume that *Workers* do not know each other; at least such knowledge does not originate from the setup of our system). Thus, disappearance of the *LMaster* that lasts more than a few minutes, is likely to result in *Workers* not being able to send results of completed tasks and/or receive new tasks to work on (the only reprieve for the *LMaster* would be if all workers would execute long-lasting jobs and the *LMaster* would be back before any of them was completed). As a result *Workers* would consider their team (leadership) not trustworthy, leave it and not come back (see, [20]).

Finally, there is a scenario in which the *LMaster* crashes and "does not recover" (long-term disappearance, possibly with a serious data loss). Let us say that there was a fatal hard drive crash and there was no backup (or

the last backup was done long time before the crash). In this case the above mentioned problems are seriously magnified:

- since the *LMaster* is the only gateway between *Users* and the team, all jobs will be lost and thus the "User-reputation" of the team will be damaged very seriously; it is doubtful that *Users* would be willing to work with this *LMaster*, as a team leader, again
- the team will disintegrate, as *Workers* will have no *LMaster* to lead them; it is doubtful that any *Worker* affected by the crash would be willing to work with this *LMaster* again; furthermore, the *LMaster* after recovery will have no up-to-date knowledge about make-up of its team
- substantial part of other collected knowledge will be gone and thus the *LMaster*, even if it returns and tries to form a new team, will have to start from the scratch; but this would be very cumbersome indeed (while the team knowledge is gone, the competition / other teams will capture the "market share").

Thus, even a temporary disappearance of the *LMaster* would have serious consequences because of both *Users* and *Workers* considering it not worthy of trust. As a result, the team would lose *Workers* and stream of revenue (*Users* would not contract jobs with it) and would very likely disintegrate.

In response to these challenges we have decided to use a well-known strategy of data mirroring [1, 4, 23, 22, 33]. In [12, 25, 27] we have proposed that an *LMirror* agent should be created, with the role of keeping a copy of information needed to keep the team alive and competitive. Furthermore, in [28] we have specified procedures of restoring the *LMaster* and the *LMirror* in the case when either one of them crashes.

Obviously, we are well aware of the fact that instantaneous (or, even, almost instantaneous) crash of both the *LMaster* and the *LMirror* would have exactly the same consequences as crash of the *LMaster* that did not have an *LMirror* and data backup (would result in destruction of the team). However, our approach is not focused on developing a completely bullet-proof system, but our goal is to create a reasonably resilient infrastructure. In this context observe that, as the time passes, it can be assumed that, in best teams, roles of the *LMaster* and the *LMirror* will be played by more-and-more reliable machines. Specifically, when the *LMaster* selects the *LMirror*, it will pick for this job the "best possible" *Worker*. It will do so by utilizing the reliability information it has accumulated thus far (see, [20]). When this *LMirror* becomes the next *LMaster* it will also select the best *LMirror* available at this time. In this way the overall strength and reliability of "team leaders" should continue to increase. Obviously, from what we described here follows that as the time passes weakest teams may be naturally eliminated. Specifically, teams with unreliable *LMasters* and *LMirrors* will either fall apart when both these agents crash at almost the same time, or due to repeated temporary disabilities, lose chances of winning new contracts or fulfilling existing ones. As a consequence such teams will lose *Users*' trust, while their *Workers* will not come back (to work for a "bad team"); and this will result in diminishing funding and, finally, team disappearance.

In our earlier work we have focused our attention on the structure of the team, and on the way that lost leader agents are replaced (and the structure of the team restored). Now, we will consider three key issues: (1) what information should be mirrored, (2) when should it be mirrored, and (3) where should it be mirrored.

4. What should be mirrored, when and where.

4.1. Team member data. For any team, the most crucial is information about its members. Each time a new member joins the team, the *LMaster* obtains the following information (note that all information is "managed" utilizing an extended Grid ontology; see [17]):

1. *ID* of the *LAgent* that is joining the team (*Worker* name). Since we have assumed that each agent utilizing our system has to be registered within it (this function is one of the roles of the *CIC* infrastructure), it is possible to check the registration of the potential *Worker*, before accepting it into the team (see, [15] for more details). This can be treated as a mini-security measure; one that can be extended as the system develops.
2. *Available resources.* While in [13] we have proposed a very minimalistic ontology of resources, we have known all along that a more detailed one is needed. More recently, in [30] we have presented an overview of Grid and agent-Grid ontologies and came to the conclusion that an extended (and somewhat modified) CoreGrid ontology [37] will be the base ontology for our system. Thus, this ontology will be also used to formulate an offer to join the team. Specifically, in the CFP the *LAgent* will include information about resources offered to the team. At this point, as we are considering resources in terms of computing nodes (consisting of hardware and software), the top entity of the resource description will be the *ComputingComponent* class and its subclasses—*ComputingElement* and *WorkerNode*. This

will enable *Worker* agents to represent either a single computing resource (*WorkerNode*) or a set of computing nodes, managed by some low level resource broker (*ComputingElement*, containing a set of *WorkerNodes*). Instances of both of these classes can have their hardware and software capabilities described by properties such as (this list is not exhaustive):

- *hasCPU*—related to the CPU class and describing the processor in terms such as: *platform*, *vendor*, *model*, *clockSpeed*, *coreCount*, *l1CacheSize*, *l2CacheSize*, etc.
- *hasMemory*—linking to the *Memory* concept described by the *Size* property.
- *hasStorage*—related to the *StorageSpace* class described further by the *StorageInterface* and the *FileSystem* concepts.
- *installedSoftware*—describing any software components of the node (the *Software* class) such as the operating system, application environment and installed libraries.
- *runningService*—any services running on the resource (e.g. OpenPBS broker running on the *ComputingElement* node).

This information about the new *Worker* will be then stored by the *LMaster* and used, among others, for (1) contract negotiations, and (2) job scheduling (see also [31]).

3. *Details of the contract.* As can be found in [28], we have suggested that the contract proposal should contain: (a) specification of the time of availability (e.g. every night from 1AM till 6AM), (b) length of contract (e.g. 2 weeks). However, results obtained by the MiG project (see, for instance papers available at [3]) illustrate that more complex sets of informations can be involved in both contract negotiations as well as in the contract (e.g. a contract with max payment value). Obviously, in the response the proposed payment for the *Worker* (and its structure) is to be provided. These details (regardless of their final complexity, resulting from extended analysis of possible economical models) constitute the description of the contract between the *LMaster* representing the team and the *Worker*.
   The exact way of modeling these concepts in our Grid ontology is in progress, however currently, they are described using the *WorkerContract* class along with the following properties:
   - *revenuePerHour*—numerical value describing the price the *Worker* is paid for each hour of its readiness to work.
   - *availability*—a multivalued property containing times of days during a week during which the *Worker* is available for job processing. The domain of this property is a list of *AvailabilityPeriod* class instances, described by properties: *startDayOfWeek*, *startTimeOfDay*, *endDayOfWeek*, *endTimeOfDay*.
   - *contractStartTime*—date and time when the contract came into effect.
   - *contractLength*—the length of the contract in days.

Out of these three items, one has to be mirrored immediately—the *ID* of the *Worker*. Without it it will be impossible to contact it in the future. The remaining two could be recovered from the *Worker*.

As far as the resources of the *Worker* are concerned, we have also to take into account the fact that in the system under consideration it is expected to that the *LAgent* will "remain the same" (its ID does not change) for a long time, while the resources it represent may change. Let us consider two scenarios. First, the *LAgent* represents one or more "home PC's." Here, machines can be added to the pool, removed from the pool, and/or upgraded. Obviously, such changes are not likely to happen each time the *LAgent* interacts with the team. However, it is even possible that one of home machines will be relatively regularly added to, and removed from, the pool of available resources. Second, the *LAgent* represents a Grid (e.g. an ADAJ based desktop Grid; see [15] for more details). This case can be expected to be even more dynamic as it is possible that each time the *LAgent* joins the team it will represent a Grid that is configured out of a different set of machines. Therefore, information about resources represented by a given *LAgent* is not static and needs to be stored each time it joins the team. Furthermore, it should now be obvious that in the case of long-term contracts it may be necessary to regularly update information about available resources (particularly in the case of an *LAgent* representing a Grid or a Cloud). Therefore, as a result of possibility of such changes a different type of a contract may need to be negotiated. This latter issue we will, however, omit as being out of scope of this paper.

Let us now consider the question of preservation of details of the contract. When designing our system we have assumed that participating agents will not only be benevolent, but also their behaviors will be "team-supporting" (pro-social). In other words, all participating agents will try, to the best of their abilities, to cooperate with others. However, such assumption is rather unrealistic when real-world situations are to be considered. Moreover, in the agent literature it is quite often the case that non-collaborating, anti-social

agent behaviors are considered (see, among others, work of D. Grosu, for instance [21]; or H. Hexmoore, for instance [36]). Taking this into account one can envision a malevolent *Worker* who tries to take advantage of crashing of the *LMaster* by providing the *LMirror* with details of the contract that have been modified to its advantage. Therefore, one of the easy ways of preventing such misbehavior of the *Worker*, is to mirror details of all contracts. In other words, information about the contract of the new/returning *Worker* should be mirrored as soon as the contract is signed (the FIPA Contract Net Protocol is completed).

Summarizing, information about ID of the worker (item (1), above) has to be mirrored immediately; information about details of the contract (item (3), above) should be mirrored immediately to prevent potential fraud; while information of resources represented by a given *Worker* (item (2), above) does not need to be mirrored. However, since information about items (1) and (3) is to be mirrored, it seems reasonable to mirror also information describing available resources (item (2)). In this way crashing of the *LMaster* will not disturb *Worker*s with unnecessary exchanges of messages with the new *LMaster*; preserving resources of both sides. Note that the total amount of information mirrored here is proportional to the number of *Workers* in the team (modulo the number of machines that each one of them represent) and thus should not be very large. Furthermore, change in the total size of mirrored data can be estimated in the case of planned team expansion or reduction (e.g. it is easy to establish how much disk space will be required if another *Worker* representing N machines is to join or leave the team).

4.2. Job contracts and their execution. In [14, 13, 12] we have specified a very limited set of parameters describing job contracts. In the CFP, the potential *User* could specify the following parameters: (a) job start time, (b) job end time, and (c) resource requirements (where resources have been limited to available hardware—matching resource information obtained from each *Worker*; see the previous section). In the response, the price for executing that job was proposed by the *LMaster*. As noted above, when introducing to our system a full-blown ontology of the Grid, additional job constraints are going to be added. However, this does not affect considerations presented here. Obviously, as soon as the contract is signed (FIPA Contract Net Protocol is completed), all data concerning it has to be mirrored. Information about signed contracts is crucial to the continued existence of the team, as each signed contract means income for the team. Furthermore, losing information about a contract would mean that it would either not be completed, or results would have no *User* to be sent to (the latter would happen in the case, when information was lost while the task was already being executed by one of the *Workers*). In both cases, the team would rapidly lose its reputation and potential future contracts ([20]). Furthermore, as noted above, as soon as the assumption about pro-social, benevolent attitude of all agents in the system is relaxed, one of the ways to protect interest of all parties is to mirror all contract information.

Signing a contract leads to an issue that is interesting in its own right: what should happen with job-related files? As it is obvious for anyone working with Grids (and illustrated in [35]), a typical job description involves a file (or multiple files, when job orchestration is concerned) that contains the code, and one or more data files. The first possibility would be to request files from the *LAgent* at the time when the job is "about to start." This approach would reduce the burden on the team. When the job was to start (either because of contract stipulation—the *job start time* condition—or because time to execute a given job has come) the *LMaster* would contact the *LAgent* with a request to send all necessary files. The disadvantage of this solution is that there is no guarantee (and in this case there should be none) that the *LAgent* is going to be available at the time when the files are to be requested. From the point of view of the *LAgent* its job is to negotiate the contract, deliver the files to the team that is to complete the task, and receive the results. In the meantime the *LAgent* can be off-line (computer it resides on can even be switched off). This is particularly the case when the *job end time* condition is a part of the contract (it will come back to life when it is time to pick up the results). This brief analysis points out to the fact that we have to assume that completing the contract is primarily the responsibility of the agent team. As soon as the *LMaster* signs the contract, the team takes responsibility for completing the task (note that the payment for completing the task can be assured using appropriate proxy mechanisms; [20]). Therefore, solution that expects files defining the task to be transferred when the job is about to start is not acceptable. Such files have to be transferred to the *LMaster* immediately after the contract is signed.

With this in mind, let us consider possible actions of the *LMaster* that has received all files pertinent to a newly contracted job. Naturally, the *LMaster* could immediately establish which *Worker* is to execute the job and initiate job staging, by sending all files to that *Worker*. Unfortunately this is the situation when, our fundamental assumption—about highly dynamic nature of nodes in the global Grid comes to play. Determining,

well ahead of job start time, which *Worker* is going to execute it, seems to be against the very nature of utilization of agent systems. Here, the robustness and efficiency of such systems comes from their flexibility (e.g. ability to negotiate and re-negotiate "job contracts;" as described, for instance, in [32, 8, 7]). Therefore, it seems that it would be much better to stage jobs according to the *actual* state of the system at the the time of job start (rather than on the basis of the predicted state of the system). This approach should allow to avoid influence of unexpected factors, such as, for instance failure of some *Workers*. This means that data files should not be immediately transfered from the *LMaster* to the *Worker*. Combining these two observations we realize that all files necessary to complete the job have to be held by the *LMaster* until the start of job execution. This means, in turn, that these files have to be mirrored.

As soon as the job is completed, its results have to be sent back to the *User*. Now, let us recall that the only agent that knows the *User* is the *LMaster* and thus the *LMaster* has to send them back. Here, there exist two possible mirroring options. First, while the *LMaster* keeps a copy of results, the second copy is kept by the *Worker*; until it receives a confirmation from the *LMaster* that they were successfully delivered to the *User*. In this way we, again, have two copies of sensitive material available in the system. Furthermore, note that the overall reliability of the *Worker* is assumed to be smaller than that of the *LMaster* and the *LMirror* (the latter two agents have been selected on the basis of much more stringent criteria, and their overall capabilities and their reliability should improve as the team continues to exist). Therefore, the situation in which important data is not "properly" mirrored should not be sustained for an extended time. However, note that the *Worker* can end its contract and not come back to the team. In this case, if the *LMaster* crashes, results could be lost before they are delivered to the *User*. To deal with such situation, if data cannot be delivered immediately to the *User* (e.g. an appropriate *LAgent* is not available), results have to be mirrored as any other sensitive data and the *Worker* should be released from the duty of keeping a copy.

Information about job completion and successful delivery of results has to be stored. It is going to be useful, first, for utilization in job scheduling (see, for instance, [31] and references collected there). Second, for contract disputes, and third, for considerations concerning trust. Interestingly, only the fact that the job has been completed and its results successfully delivered to the customer, has to be mirrored immediately (this information describes the current status of the team, which has to be kept up to date). Information needed for future job scheduling does not have to be mirrored immediately. Losing some data about execution times of some jobs (due to the crash of *LMaster*) does not pose a threat to the existence of the team. Similarly, some information related to trust assessment of a *Worker* may be lost without damaging the team. Therefore, these two item-sets can be mirrored in a digested form in pre-specified time intervals (see subsequent sections).

4.3. Trust information. Thus far we have dealt primarily with information that has to be mirrored immediately, now we devote our attention to the remaining data generated in the system. In [20] we have considered issues related to trust in relationships between *Users* and *LMasters* (teams) and between *LMasters* and their *Workers*. We have shown that, utilizing a proxy environment (similar, for instance, to PayPal [2]) it is possible to assure that payment for a completed job is always delivered. Therefore, for the purpose of this paper, we are interested only in relationships between the *LMaster* and its *Workers*. Obviously, since we are interested in information mirroring within agent teams, the way that trust information is to be stored by *Workers* is not considered.

Following the discussion presented in [20] we assume that, for the time being at least, contract between *Worker* and the team (the *LMaster*) is based on paying the *Worker* for availability (not for actual work done); i. e. *Worker* contracted for 6 hours every night will be paid for this many hours and will be expected to be available within that time-frame to do work, if called. It was also stipulated that, at least for some contracts, *Worker* will have a right to not to be available for a limited number of times during the contracted time. Here, we can use the "pinging-mechanism" described in [12] to monitor *Worker* availability. Under these assumptions, in [20] we have proposed that for each *Worker* we will collect information how many times (a) it *fulfilled the contract* ($\#fc$)—was available all the time when it was expected to be available (if in the contract it was stated that it can be missing for a certain number of times, this is exactly the number of times it was missing); (b) violated the contract ($\#vc$)—was not available when it was supposed to be; and (c) did more than contract required ($\#ac$)—could have been unavailable for a specific number of times during a contract, while was available all the time. This combined data will be stored, for each *Worker*, as a quadruple $(n, \#fc, \#vc, \#ac)$, where n is the total number of contracts. While this information is collected in a cumulative form (and thus is of size proportional to the number of *Workers* in the team), we have also access to additional information

that can be used for assessment of trust / reliability of a *Worker*. First, detailed information about results of each "pinging-procedure," and second, detailed information about completion (or not) of each assigned job (e.g. how much time a given job took, was it completed on time, before time, or delayed, etc.). Note that the results of the "pinging-procedure" can be used also to adjust time within which response from a given agent is expected (in this way system will be able to adapt its behavior to the network conditions of each *Worker*; see, also [28]). Since the latter data is not proportional to the number of *Workers* in the team, but to the number of "pinging-procedures" and processed jobs, we consider it separately in section 4.4.

Obviously, the trust-related information has to be mirrored. However, it is easy to realize that loss of some of this information is not catastrophic for the team; e.g. not updating immediately number of fulfilled contracts for a specific *Worker* will result in proceeding with the old trust assessment (note that the basic data is cumulative). Therefore, it is enough to update information about trust related issues in a digested format at predefined time-intervals. Frequency of updates depends on the nature of jobs that a given team is executing and on the nature of *Worker* contracts. If a team is contracting long-lasting jobs and its members have long-term contracts, then update of trust information can occur much less frequently than in the case when large numbers of short jobs are completed by a team consisting of *Workers* with short-term contracts. We believe that a good approximation of the right time to send an update of trust information to the *LMirror* is when, in average, each *Worker* has completed one contract. Note that updates of information can be sent at any time, which allows the *LMaster* to adaptively adjust time between updates depending on the nature of jobs and contracts currently existing in its team.

4.4. High volume data collection and storage. Most information considered thus far within the system was relatively small in volume—on order of the size of the team, or of the number of currently contracted jobs (note that in the latter case files related to some jobs may be large, but this is unavoidable). There are, however, important data sets that grow over time and can become very large. First, data generated during negotiations between the *LMaster* and *Users* who would like either to contract job execution, or join the team. This data is to be used to establish market conditions and introduce team behavior adaptivity to the system (see, also [12, 20]). Second, information about execution of completed jobs (e.g. hardware used, software required, execution time, *Worker* completing the job, etc.). This data can be used to apply advanced scheduling techniques (for more details, see [31], and references collected there). Third, detailed information about behavior of each worker (e.g. detailed history of execution of each job, results of "pinging-procedures," etc.). This information is crucial to build a realistic and comprehensive economy-based job scheduling model (which includes also trust and reliability related considerations).

Let us first consider contract negotiations. In [34, 6, 9, 10] a large variety of negotiation mechanisms that can be applied in a Grid have been described. However, in the initial implementation of our system, for both types of contract negotiations (job execution, or joining a team), we have decided to utilize the FIPA Contract Net Protocol ([5]). The primary reason was that while allowing for "calls for proposals" with practically unlimited variety of constraints, it generates the contract (or establishes impossibility of an agreement) within a single round of negotiations. This also reduces the total amount of information that needs to be stored for further use (compare, for instance, with the amount of data generated during any of the multi-round auction mechanisms; see [16]). Therefore, in our system, information about a single job contract negotiation would consists of:

- *Content of the CFP*—containing, among others, information what hardware and/or software was sought; job start time (if specified); how much time was to be contracted / or was it an open-ended contract?
- *Content of the response*—what was the price proposed by the *LMaster*, as well as other details of the proposed contract; e.g. the proposed hardware (when the CFP specifies the minimum requirements, while the response proposes the actual configuration that the task will be executed on), etc.
- *Result of the negotiation*—was it a success, or not?

Note that, with a rich vocabulary provided by the extended Grid ontology, it will be possible to reconsider the utility of multi-round Service Level Agreement negotiations.

In the case of negotiation concerning a *Worker* joining the team, at least the following information becomes available (introduction of a full-blown Grid ontology will allow utilization of more complex negotiation scenarios):

- *Content of the CFP*—including information about available hardware and software, length of contract and conditions of availability,

- *Content of the response*—the proposed price
- *Result of the negotiation*—did the *Worker* join the team, or not

Note that, regardless of the particular form of negotiation used, amount of available data will be large. Specifically, it will be proportional to the total number of contract negotiations that a given team participated in. Furthermore, this dataset will be constantly increasing in volume.

Interestingly, in all cases, data collection is resilient to loss of some information. In other words, losing some of the data may decrease competitiveness of the team (e.g. when performing data mining to find the current "value" of various resources that the team currently consists off, the result may be somewhat incorrect due to the fact that some data was lost), but the process is characterized by graceful degradation. Therefore, it is possible to perform updates in a digested fashion at times of reduced utilization of the system, while risking that some data will be lost if the *LMaster* crashes in the meantime.

It should be now obvious that constantly increasing volume of data makes mirroring very costly. We can point to at least three important issues that need to be taken into account.

1. The total amount of data that has to be regularly passed from the *LMaster* to the *LMirror* is not small. This is the case even when digested information is transferred in a compressed form. Here, we deal with a constant stream of data, size of which depends on the size of the team and the level of activity within it.
2. Constantly increasing volume of data has to be stored by *both* the *LMaster* and the *LMirror*. Even assuming that over time capabilities of both managerial agents improve, and taking into account that storage is cheap and plentiful, collecting data generated during management of a large and active team that exists for an extended period of time, may be prohibitive for most home PC's. This will, in turn, introduce a new stratification into the Grid. Its *Users* will be divided into those who can and those who cannot manage the team due to the storage restrictions of their computers.
3. The time of recovery of a crashed *LMirror*, and of the second step of replacing a crashed *LMaster* (in the first step the new *LMAster* replaces the *LMirror* on the same machine, and thus no data transfer is needed) is going to grow with the data size. It is important to realize that (re)creation of a new *LMirror* involves copying an entire set of mirrored data to the new location. For large, active teams that exist for a long time, such data set can be very large. As a result transfer involving ACL messages is not feasible. The only reasonable solution seems to be a direct data transfer between computers that host both agents. However, even this process can take considerable amount of time, if it is performed over the Internet. Now, let us take into account that, as suggested in [27], recovery of a crashed managerial agent stops the other managerial agent from doing anything else (to re-establish security of existence of the team; brought about by data mirroring). This means that as the time passes each crash of the managerial agent takes longer to recover from (more data to be mirrored) and, for all practical purposes, stops the team operation for an ever increasing time. Therefore, such approach seems infeasible in a long run.

In this context note that, to reduce potential impact of increasing volume of collected data, it would be possible to store only the newest information (e.g. a "sliding window" consisting of K most current data items). However, the primary reason to collect a "complete set of historical data" is to be able to apply data mining techniques to extract useful information out of it. Decreasing the amount of collected data, by including only the most current items, could reduce effectiveness of data mining and thus contradict the reason to collect this data in the first place. Therefore, it seems to be in the best interest of the team to preserve as much of historical data as possible.

Considering this, on the basis of the current trend of IT infrastructure—outsourcing and specialization, we propose that an outsourced data warehouse will be utilized for team data storage. Specifically, within our system, data warehousing could be outsourced to a team that *specializes* in data storage. This is similar to, for instance, a popular e-commerce scenario, where merchants contract software companies to design, implement and run infrastructure of multi-front e-stores, while software companies contract storage companies to actually store the data. Note that while we use the name team; it is quite possible that actually this will be a single entity (as in the case of e-commerce solutions) that will be interfaced with our teams through the agent infrastructure. In other words, the front end of the data warehouse could be one or more agents, but there could be no actual agent team working "behind it."

Obviously, it would be also possible to hire *Workers* with the right infrastructure to facilitate data warehousing within the team itself. However, this solution has a number of potential problems. First, recall the

assumption that each node (*Worker*) can disappear without advanced warning. This assumption would apply also to the *Worker*(s) responsible for team data storage. As a result, potential problems that lead us to propose data mirroring in the first place, would remain unsolved. Second, idea of internal data warehousing goes against the current trend in IT, which is to outsource anything that is not the core functionality (as long as it can be outsourced, e.g. currently it is unfeasible that a bank would outsource its IT operations). The reasons are the same, as data warehousing is not expected to be the core business of most teams. Data warehousing requires a specific infrastructure (e.g. a server farm) and procedures to assure data persistence and security. Therefore, on the one hand, it is better to create a facility that specializes in this type of operation and sells its services to others; on the other it is better to buy service that assures quality of data preservation.

Finally, let us observe one more benefit of utilization of an external storage facility for system data. One of the important features of the system (see, also Figure 2.1) is that all collected data will be used for knowledge extraction. The question thus arises, how will this be done? For instance, assuming that data is stored by the *LMaster* and the *LMirror*, which dataset will be used for data mining? Since the *LMaster* is the gateway and responsible for all contacts with *Users*, it seems that this is not the right node to stage the second agent, running data mining algorithms. At the same time, the *LMirror* may not have the most current set of data. However, since it has "much less to do," this node is the one that could be easily used for data mining (possibly, performed by a separate data mining agent). However, the situation is much simpler in the case of an outsourced data storage. Here the data mining agent can be instantiated within this (external) infrastructure, or run as one of *Workers* within the team (if running it within the storage facility is prohibited). In this way we can clearly separate the knowledge management function of the system and devote appropriate resources to it.

4.5. Structure of the data warehouse. Let us now discuss how a data warehouse that could be used in our system to store the above indicated data, may look like. What we present below is a draft design of a data warehouse that could be used for storing historical data for efficient analyzing and reporting. Obviously, additional storage would be needed for preserving short term / dynamic data described in sections 4.1, 4.2. However, in this case it sill remains an open question if such data should be stored only in the warehouse, or if it should be mirrored also by the two managerial agents (see, section 4.6. Data warehouse design presented here is based on principles laid out in [24], and is built around the star table schema. Note that some details of the proposed design may change as the system evolves (e.g. when new types of complex negotiations are added). However, the general schema, which is based on the extended Grid ontology, should remain unchanged.

4.5.1. Worker negotiation. Let us start from storage of information about negotiations between the *LMaster* and the *Worker* wanting to join the team. This data is going to be stored in the *Worker Negotiation Fact* table (see Figure 4.1.

Each row of the table will correspond to a single negotiation and is going to be identified by the following dimensions:

- *User*. Identifies the *User* of the system. In this preliminary draft it contains only the *Name* of the *User*.
- *Worker*. Describes the identity (i. e. the *ID* of the *LAgent*), configuration and proposed contract conditions of the *Worker* requesting to join the team.
- *Software Group*. A multivalued dimension that describes the software configuration that is going to be contributed by the *Worker*. Each *Software Group* can be composed of many rows of the *Software Dimension* table, where each of them describes a single software component.
- *Team*. The dimension identifying the team. So far it does not contain any additional meaningful information.
- *Date*. This is a typical date dimension, as described in [24], which enables easy analysis and reporting of aggregated time series data.

Measures of the *Worker Negotiation Fact* table consist of data about money offered as compensation for the *Worker* and the information whether the *Worker* was accepted. These can be used, for example, to estimate the market value of capabilities offered by the *Worker* (e.g. repeated rejections could mean that the offered compensation is too low).

4.5.2. Job negotiation. The *Job Negotiation Fact* table gathers information about the job submission negotiations. The diagram of this fact table along with its dimensions can be found in Figure 4.2.

Similarly to the case of the *Worker Negotiation Fact* table, each row corresponds to a single negotiation, and is identified using the following dimensions:

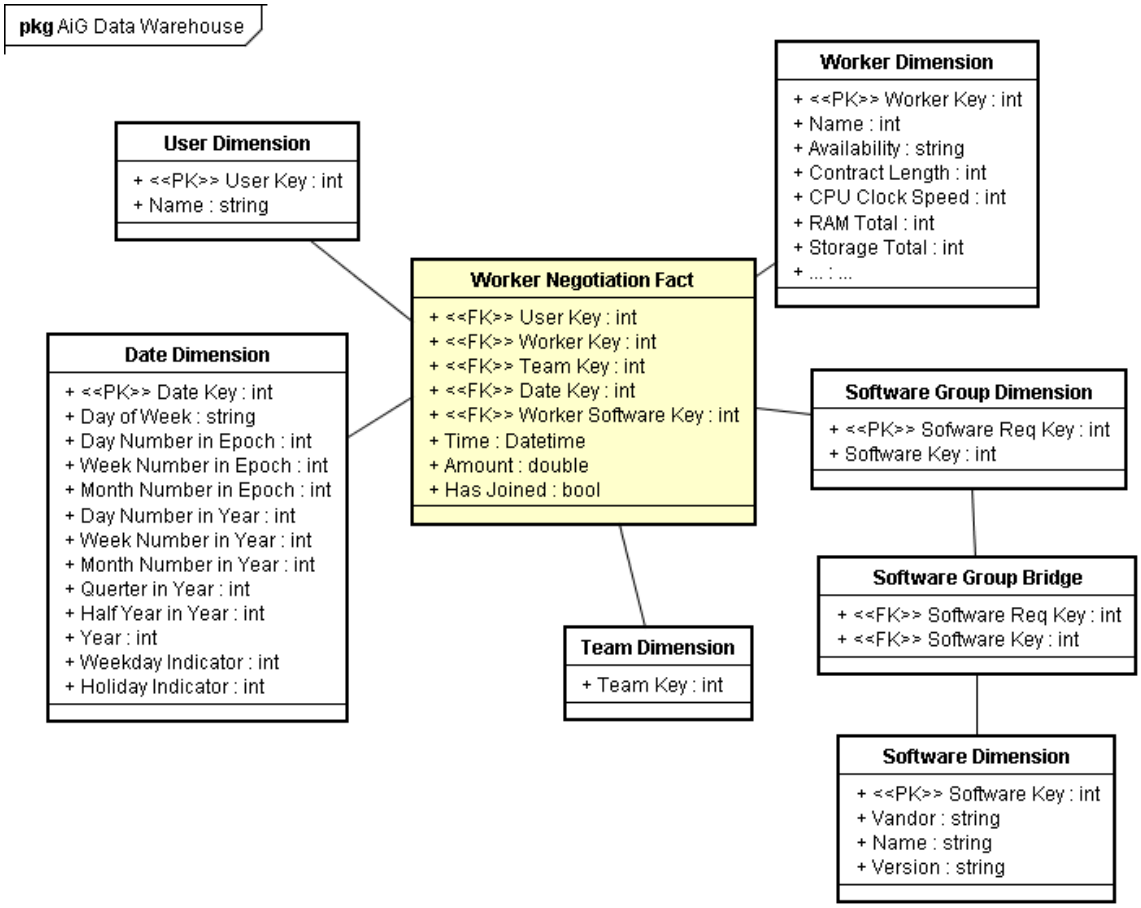


FIG. 4.1. *Worker Negotiation Fact Table*

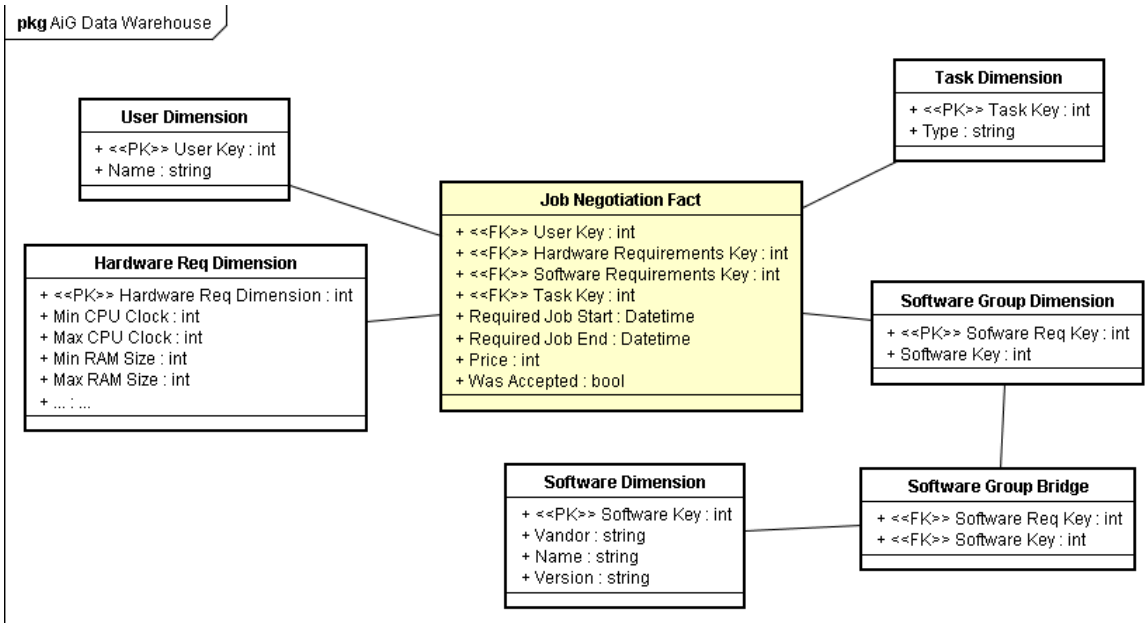


FIG. 4.2. *Job Negotiation Fact Table*

- *User*. Identifies *User* who would like to have a job executed.
- *Hardware Requirements*. This dimension describes the hardware requirements as specified by the *User*. Properties of this dimension are specified using a subset of concepts from the ontology, but flattened out and used as constraints.
- *Required Software*. Links to the *Software Group* dimension that describes software required to execute the job.
- *Task*. Describes the task (job) that the *User* would like to have executed. Details of how a task can be described or categorized have not yet been fully conceptualized; therefore, this dimension is currently empty.

Measures currently stored in the table are: the price proposed by the *LMaster* for executing the job and a boolean value specifying whether the *User* accepted the proposed conditions. Similarly as above, these dimensions may be extended in case of more complex negotiations. Furthermore, they are to be used, among others, to evaluate market conditions and will play key role in the economic model that the system is based on.

4.5.3. Task execution. Information about every task executed by the team is stored in the *Task Execution Fact* table (see Figure 4.3).

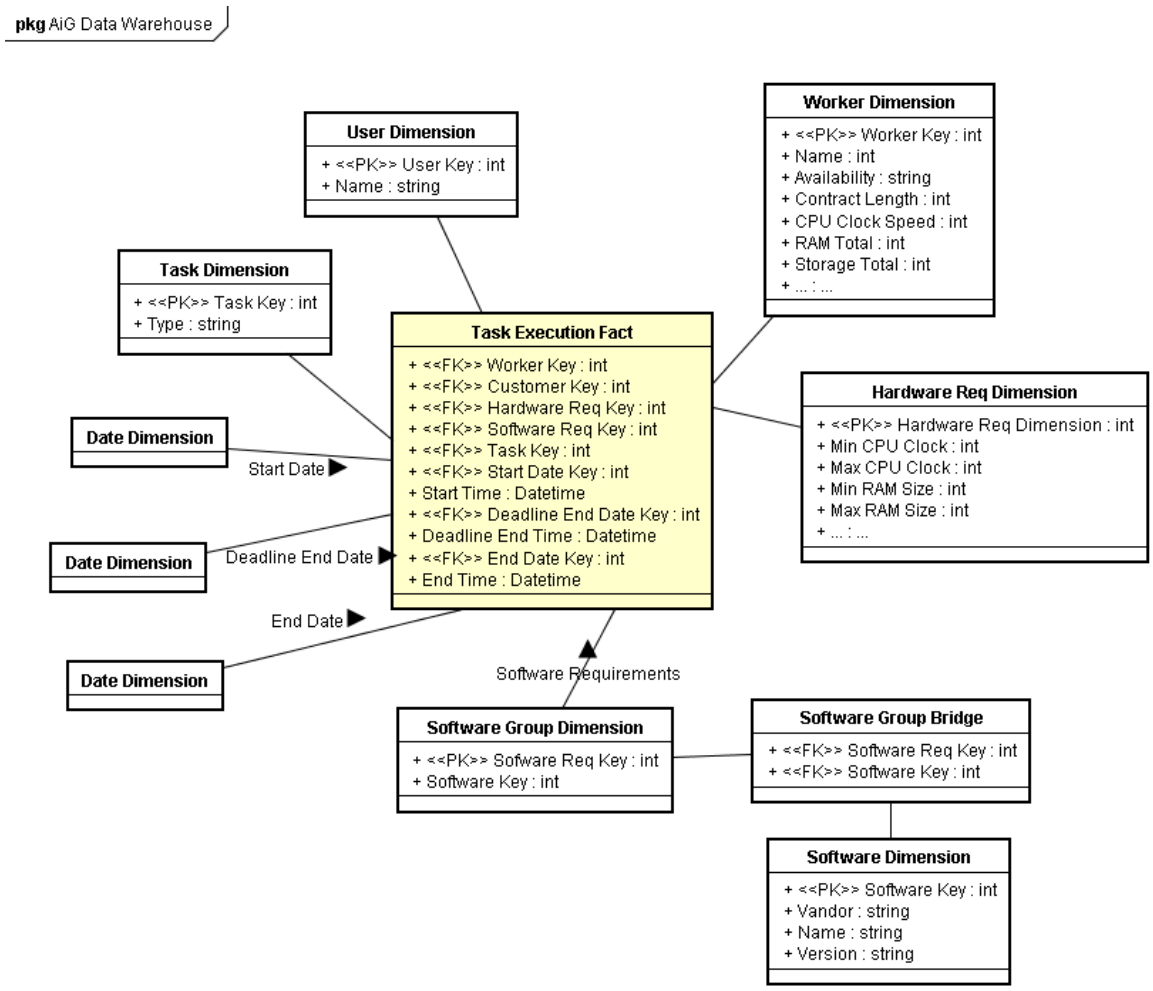


FIG. 4.3. *Task Execution Fact Table*

The table is described using the following dimensions:

- *Worker*. The worker that performed the task.
- *Customer*. The *User* that submitted the job.
- *Hardware Requirements*. Hardware that was required to perform the task.

- *Software Requirements*. Software necessary to complete the task.
- *Task*. The description of the executed task.
- *Start Date*. Link to the *Date* dimension, specifying the start of the task execution.
- *Deadline End Date*. Link to the *Date* dimension, specifying the deadline, as it was negotiated with the customer.
- *End Date*. The actual end of task execution.

For all of the *Date* dimensions (start, deadline, and actual end) there are also separate *Date* and *Time* properties to facilitate precise measurements. These measurements can be used, among others for task scheduling (based on historical data, see [31]).

4.5.4. Worker responsiveness. The information about the "responsiveness" of *Workers* within the team is stored in the *Worker Responsiveness Fact* table. It contains all data necessary to analyze the actual availability of the *Worker*, and conformance to the Service Level Agreement between the *Worker* and the *LMaster*. Recall, that in the current design of our system it is assumed that *Workers* are paid for being available at specific times. The design of this table is rather simple and is presented in Figure 4.4.

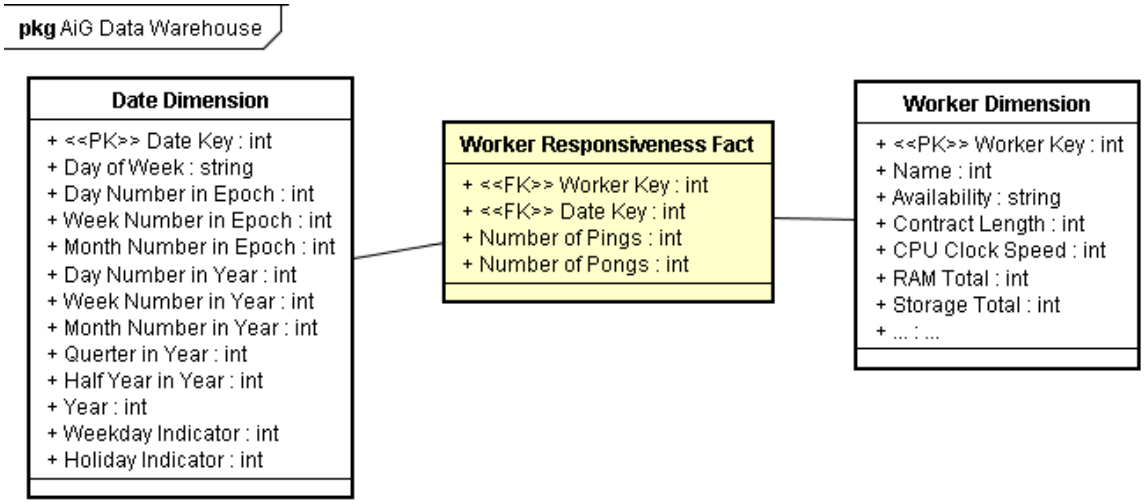


FIG. 4.4. *Worker Responsiveness Fact Table*

The table is identified by two dimensions:

- *Worker*. Identifies the *Worker* that the 'responsiveness" data concerns.
- *Date*. Specifies dates for which the measures apply.

Collected measures that can be used to analyze *Worker* "responsiveness" are:

- *Number of pings*. The number of messages checking the availability of the *Worker* sent by the *LMaster*.
- *Number of pongs*. The number of responses confirming the availability of the *Worker*.

It is likely that the collected measures are going to be extended, but they will remain stored in the same table.

4.6. Consequences of utilization of outsourced data storage. Let us now observe that utilization of a data warehouse allows us to redefine "roles" of and interactions between the *LMaster* and the *LMirror*. Thus far the *LMaster* and the *LMirror* were assumed to store and regularly synchronize **all** data pertinent to the long-term existence and success of the team. In the new design, the *LMaster* stores all such information in the contracted storage facility and thus the *LMirror* does not store any data at all. However, it should be clear that this does not mean that the *LMaster* does not store any data. It would be highly inefficient to query the external data warehouse every time some information is needed (for instance, to send a message to any/all team member(s)). Therefore we have to divide available information into three categories. (a) Information copy of which is definitely stored by the *LMaster*, (b) kept a copy if possible (depending on its capabilities), and (c) information stored in the warehouse only.

In category (a) we include current information concerning: list of team members, their resource specification, contract details (both job contracts and *Worker* contracts). In category (c) we include all high volume data

discussed in section 4.4. The remaining information discussed here, such as, for example, cumulative trust information, market value of specific hardware, etc., may be stored by the *LMaster*, but can be also accessed from the data warehouse.

Let us now come back to the *LMirror*. It does not store any data, but has procedures for: (a) checking existence of the *LMaster*, and (b) becoming an *LMaster* in the case when it crashes. Note that this simplifies the overall situation of the *LMirror*. Thus far one of the important issues was, should the *LMirror* work in a similar way as any other *Worker* agent. It was assumed that the *LMaster* will not work as a *Worker*, but will be 100% involved in team management. It will also be paid for its efforts from the overhead imposed on all contracts (see, [29] for more details). However, in the case of the *LMirror* the situation was not this obvious. On the one hand, it had to store team data and regularly check that the *LMaster* is still alive. On the other hand, since these actions require much less effort that these of the *LMaster*, it could perform some tasks as other *Workers* do. However, as a *Worker* it would not be as efficient as the others. Recall, for instance, that it was to process digested and compressed packages containing information about negotiations that took place recently. This being the case the *LMirror* would have to stop processing the *User* request and *immediately* take care of mirroring-related operations (since, assumptions behind our system setup specify that these operations have precedence over regular work). Let us also recall the suggestion made above, that one of the functions of the *LMirror* could be mining the team data; as an add-on function that could be performed instead of being a regular *Worker*. Regardless of the final solution used, this lack of clarity as to functionalities and payment (how much should the *LMirror* be paid for its services in each of the above described approaches?) would make the design of the system unclear and maintenance more difficult. This particularly concerns the economic model behind the system that would become more / unnecessarily complicated.

Situation becomes much clearer when the outsourced data warehouse-based solution is used. Since the *LMirror* does not have to store data it can work as all other *Workers* (the only mirroring-related procedure will be checking existence of the *LMaster*—using the pinging-procedure, see [28]). Obviously, if the *LMaster* crashes, the *LMirror* will have to undertake emergency procedures and become the *LMaster* and re-create the *LMirror*. However, this being an emergency situation requires special measures (e.g. involving dealing with the job it was processing). As suggested above, mining data stored in the warehouse can be delegated to a special data mining agent. In the case that this agent cannot move to the node where the data is stored, this could be a natural work for the *LMirror*, as in this case the data mining task it is executing can be stopped at any time. Furthermore, the economic aspect of the system can be simplified, as the *LMirror* can be paid as any *Worker*.

4.6.1. Restoration of managerial agents. Let us now briefly summarize steps that take place when a crashed managerial agent is going to be re-created in the scenario when team data is persisted in the external data warehouse. First, let us consider re-creation of a crashed *LMirror*. In this situation the *LMaster* selects the *Worker* that is to become the next *LMirror*. For this it utilizes information about *Worker* resources, as well as contract and trust information (selected agent should not only be one of the best computers in the team, but have a contract with no expiration date and a very good performance track record). Next, the task that this *Worker* was working on is transferred to another *Worker*, while the selected *Worker* is upgraded by uploading appropriate modules (see, [18]. Among these modules the new *LMirror* obtains information where the team data is stored and details how to access this information. As soon as the new *LMirror* is fully operational, the *LMaster* undertakes the following steps: (a) informs the *CIC* about the fact that its team has the new *LMirror*, (b) informs all remaining *Workers* about the identity of the new *LMirror*, and (c) if necessary, informs the data storage facility that the new *LMirror* has access rights to the team data.

As noted above, the replacement of the crashed *LMaster* consists of two phases. First phase proceeds exactly as described in [28], and takes place within the node that hosts the *LMirror*. The only differences involve (i) access rights to the team information being established / confirmed with the data warehousing infrastructure, and (ii) copying necessary managerial data to the new *LMaster* (see, Section 4.6). The result of this phase is replacement of the *LMirror* (which at the end of the process self-destructs) by th new *LMaster*. This means that the team has now an *LMaster* and no *LMirror*. Therefore the, above described, procedure of re-creating an *LMirror* follows immediately, in the second phase.

5. Concluding remarks. The aim of this paper was to discuss issues involved in information mirroring in an agent-based Grid resource management system. We have focused our attention on information generated within the agent team and considered four important cases: (1) team data, (2) job contracts and their execution, (3) trust-related information, and (4) other sources of large volume information. We have established that we

have to deal with two main situations: (a) small-volume data that has to be mirrored immediately, and (b) large volume data that may be mirrored infrequently. Further analysis indicated that large volume data collection may be best achieved through utilization of a contracted data storage facility. This latter solution is our solution of choice and we plan to utilize it in our system.

Acknowledgments. Work of Maria Ganzha and Michal Drozdowicz was supported from the "Funds for Science" of the Polish Ministry for Science and Higher Education for years 2008-2011, as a research project (contract number N516 382434). Collaboration of the Polish and Bulgarian teams is partially supported by the *Parallel and Distributed Computing Practices* grant. Collaboration of Polish and French teams is partially supported by the PICS grant *New Methods for Balancing Loads and Scheduling Jobs in the Grid and Dedicated Systems*. Collaboration of the Polish and Russian teams is partially supported by the *Efficient use of Computational Grids* grant. Work of Marcin Paprzycki and Sofiya Ivanovska was supported in part by the National Science Fund of Bulgaria under Grant No. D002-146/16.12.2008

REFERENCES

[1] *Fast-start failover best practices: Oracle data guard 10g release 2*. http://www.oracle.com/technology/deploy/availability/pdf/MAA_WP_10gR2_FastStartFailoverBestPractices.pdf.
[2] *Paypal*. http://www.paypal.com.
[3] *Project: Minimum intrusion grid*. http://www.migrid.org/MiG/Mig/published_papers.html.
[4] *Sql server 2008 failover clustering*. http://download.microsoft.com/download/6/9/D/69D1FEA7-5B42-437A-B3BA-A4AD13E34EF6/SQLServer2008FailoverCluster.docx.
[5] *Welcome to the FIPA*. http://www.fipa.org/.
[6] A. Abraham, R. Buyya, and B. Nath, *Natures heuristics for scheduling jobs on computational grids*, in Proc. of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000, pp. 45–52.
[7] H.-J. Burckert, K. Fischer, and G. Vierke, *Holonic transport scheduling with teletruck*, Applied Artificial Intelligence, 14 (2000), pp. 697–725.
[8] S. Bussmann and K. Schild, *An agent-based approach to the control of flexible production systems*, in Proc. of the 8th IEEE Int. Conf. on Emergent Technologies and Factory Automation (ETFA 2001), vol. 2, IEEE CS Press, Los Alamitos, CA, 2001, pp. 481–488.
[9] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, *Economic models for resource management and scheduling in grid computing*, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 1507–1542.
[10] R. Buyya, J. Giddy, and D. Abramson, *An evaluation of economy-based resource trading and scheduling on computational power grids for parameter weep applications*, in Proceedings of the Second Workshop on Active Middleware Services (AMS 2000), Pittsburgh, USA, August 2000, Kluwer Academic Press.
[11] R. Cervenka and I. Trencansky, *Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS*, Whitestein Series in Software Agent Technologies and Autonomic Computing, A Birkhauser book, 2007.
[12] M. Dominiak, M. Ganzha, M. Gawinecki, W. Kuranowski, M. Paprzycki, S. Margenov, and I. Lirkov, *Utilizing agent teams in grid resource brokering*, International Transactions on Systems Science and Applications, 3 (2008), pp. 296–306.
[13] M. Dominiak, M. Ganzha, and M. Paprzycki, *Selecting grid-agent-team to execute user-job—initial solution*, in Proc. of the Conference on Complex, Intelligent and Software Intensive Systems, Los Alamitos, CA, 2007, IEEE CS Press, pp. 249–256.
[14] M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki, *Utilizing agent teams in grid resource management—preliminary considerations*, in Proc. of the IEEE J. V. Atanasoff Conference, Los Alamitos, CA, 2006, IEEE CS Press, pp. 46–51.
[15] M. Drozdowicz, M. Ganzha, W. Kuranowski, M. Paprzycki, I. Alshabani, R. Olejnik, M. Taifour, M. Senobari, and I. Lirkov, *Software agents in adaj: Load balancing in a distributed environment*, Applications of Mathematics in Engineering and Economics'34, (2008), pp. 527–540.
[16] M. Drozdowicz, M. Ganzha, M. Paprzycki, M. Gawinecki, and A. Legalov, *Information flow and usage in an e-shop operating within an agent-based e-commerce system*, in Journal of Siberian Federal Univercity, vol. 2 of Engineering and Technologies, 2009, pp. 3–22.
[17] M. Drozdowicz, M. Ganzha, M. Paprzycki, R. Olejnik, I. Lirkov, P. Telegin, and M. Senobari, *Ontologies, agents and the grid—an overview*, in Profeedings of the PARENG'2009 Conference, 2009. in press.
[18] M. Ganzha, M. Gawinecki, M. Szymczak, G. Frackowiak, M. Paprzycki, M.-W. Park, Y.-S. Han, and Y. Sohn, *Generic framework for agent adaptability and utilization in a virtual organization—preliminary considerations*, in Proceedings of the 2008 WEBIST Conference, J. et. al., ed., Setubal, Portugal, 2008, INSTICC Press.
[19] M. Ganzha, M. Paprzycki, M. Drozdowicz, M. Senobari, I. Lirkov, S. Ivanovska, R. Olejnik, and P. Telegin, *Information flow and mirroring in an agent-based grid resource brokering system*, in Proceedings of LLSC Meetings. to appear.
[20] M. Ganzha, M. Paprzycki, and I. Lirkov, *Trust management in an agent-based grid resource brokering system—preliminary considerations*, in Applications of Mathematics in Engineering and Economics'33, M. Todorov, ed., vol. 946 of AIP Conf. Proc., College Park, MD, 2007, American Institute of Physics, pp. 35–46.
[21] N. Garg, D. Grosu, and V. Chaudhary, *Antisocial behavior of agents in scheduling mechanisms*, IEEE Transactions on Systems, Man and Cybernetics, 37 (2007), pp. 946–954. Part A.

[22] M. Ji, A. Veitch, and J. Wilkes, *Seneca: remote mirroring done write*, in HP Labs publications, 2003.
[23] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, *Designing of disaster*, in HP Labs publications, 2004.
[24] R. Kimball and M. Ross, *The data warehouse toolkit: the complete gide to dimensional modeling*, Wiley, 2 ed., 2002.
[25] W. Kuranowski, M. Ganzha, M. Gawinecki, M. Paprzycki, I. Lirkov, and S. Margenov, *Forming and managing agent teams acting as resource brokers in the grid—preliminary considerations*, International Journal of Computational Intelligence Research, 4 (2008), pp. 9–16.
[26] W. Kuranowski, M. Ganzha, M. Paprzycki, and I. Lirkov, *Supervising agent team an agent-based grid resource brokering system—initial solution*, in Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems, F. Xhafa and L. Barolli, eds., IEEE CS Press, Los Alamitos, CA, pp. 321–326.
[27] ———, *Supervising agent team an agent-based grid resource brokering system—initial solution*, in Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems, F. Xhafa and L. Barolli, eds., Los Alamitos, CA, 2008, IEEE CS Press, pp. 321–326.
[28] ———, *Supervising agent team an agent-based grid resource brokering system-initial solution*, in Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems, F. Xhafa and L. Barolli, eds., Los Alamitos, CA, 2008, IEEE CS Press, pp. 321–326.
[29] W. Kuranowski, M. Paprzycki, M. Ganzha, M. Gawinecki, I. Lirkov, and S. Margenov, *Agents as resource brokers in grids—forming agent teams*, in Proceedings of the LSSC Meeting, LNCS, Springer, 2007.
[30] M.Drozdowicz, M.Ganzha, M.Paprzycki, R.Olejnik, I.Lirkov, P.Telegin, and M.Senobari, *Parallel, Distributed and Grid Computing for Engineering*, Computational Scientce, Engineering and Technology Series:21, Saxe-Coburg Publications, Stirligshire, UK, 2009, ch. Ontologies, Agents and the Grid: An Overview, pp. 117–140.
[31] M.Senobari, M.Drozdowicz, M.Ganzha, M.Paprzycki, R.Olejnik, I.Lirkov, P.Telegin, and N.M.Charkari, *Parallel, Distributed and Grid Computing for Engineering*, Computational Scientce, Engineering and Technology Series:21, Saxe-Coburg Publications, Stirligshire, UK, 2009, ch. Resource Management in Grids: Overview and a discussion of a possible approach for a Agent-Based Middleware, pp. 141–164.
[32] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, K. Krishnakumar, and A. Meisels, *A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing*, in Advances in Grid Computing—EGC 2005, vol. 3470/2005 of Lecture Notes in Computer Science, Germany, 2005, Springer Verlag, pp. 651–660.
[33] R. H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara, *Snapmirror: File-system-based asynchronous mirroring for disaster recovery*, in FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2002, USENIX Association, p. 9.
[34] D. A. Rajkumar Buyya and S. Venugopal, *The grid economy*, in Proceedings of the IEEE, vol. 93, 2005, pp. 698–714.
[35] M. Senobari, M. Drozdowicz, M. Paprzycki, W. Kuranowski, M. Ganzha, R. Olejnik, and I. Lirkov, *Combining an jade-agent-based grid infrastructure with the globus middleware—initial solution*, in Proc. of the CIMCA-IAWITC 2008 Conference, M. Mohammadian, ed., Los Alamitos, CA, 2008, IEEE CS Press, pp. 890–895.
[36] D. Ward and H. Hexmoor, *Deception as a means for power among collaborative agents*, in Proceedings of the Fifth International Symposium on Collaborative Technologies and Systems (CTS 2004), W. Smari and W. McQuay, eds., Society for Modeling and Simulation International, 2004, pp. 109–115.
[37] W. Xing, M. D. Dikaiakos, R. Sakellariou, S. Orlando, and D. Laforenza, *Design and development of a core grid ontology*, in Proc. of the CoreGRID Workshop "Integrated research in Grid Computing,", November 2005, pp. 21–31.

Edited by: Dana Petcu
Received: Oct 15th, 2009
Accepted: Nov 3rd, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 413–418. http://www.scpe.org
ISSN 1895-1767

MULTI-APPLICATION BAG OF JOBS FOR INTERACTIVE AND ON-DEMAND COMPUTING

BRANKO MAROVIĆ, MILAN POTOČNIK, AND BRANISLAV ČUKANOVIĆ*

Abstract. A generic infrastructural grid service for submission, management, and access to computing resource is described. It is suitable for interactive applications and on-demand computing with frequent usage of short jobs. The improved access to computing resources is achieved through allocation and maintenance of a pool of ready jobs, the size of which is dynamically adjusted to demand produced by applications. A set of APIs facilitates communication between the clients and worker jobs through a simple programming model.

Key words: cluster and grid computing, distributed computing, pilot job infrastructures, interactive applications, programming API

1. Introduction. The grid infrastructures provides access to enormous computing resources to its users. There are many grid sites dispersed throughout the world and they are grouped into several more or less independent infrastructures. Obtaining these resources can be difficult, especially for users that are not experts in the usage of grid, but are rather scientists or users of specific applications running on the grid. Various grid middlewares employ different operational schemes, authorisation policies and access interfaces. Aside from the complexity of the grid infrastructures, certain limitations are also present, notably the need for users to wait, sometimes for a significant time, until their requests for computing resources (also called job submissions) are processed and the lack of good support for interactive applications.

The use of pilot jobs [1] is gaining increasing popularity among various groups of grid users. With pilot-based infrastructures, users submit their jobs to a centralized queue or repository. These jobs are handled by grid computing resources executing asynchronously started pilot jobs. Pilot jobs communicate with a pilot aggregator, which allocates user jobs from the repository. Once started, the jobs get the actual work to be performed from the aggregator. This late binding of user jobs and resources greatly improves the user experience, as it hides broken grid resources and provides more accurate information about available resources.

On the other hand, there are many implementations of user interaction with jobs running on the grid, most of which provide communication between the user and job through text terminal-like user interface. However, none of them addresses one of key problems related to the interactive jobs—the startup time for grid jobs that is often unacceptably long for interactive use.

Work Binder is a generic service developed in Java programming language whose main purpose is to quickly allocate new jobs for grid users. Through its simple API, it also has support for interactivity between the end user and the job being run on the remote server located on the grid. With gLite middleware [2], this server is called a worker node (WN) and it is one computing node belonging to the computing element (CE), which is usually a cluster administered by some grid site. However, this service can be easily adapted to any other infrastructure that allow applications to submit code to computing resources.

Work Binder [3] is aimed to be used with three specific types of grid jobs:

- Interactive jobs—jobs executing on a remote worker node that require communication (interaction) with the user that started them (in gLite support for these types of jobs is very poor).
- Jobs with relatively short execution time that have a good chance of being resubmitted or participate in a workflow.
- Jobs with critical demand for startup time.

All of these job types have one thing in common, the need for short startup time. Work Binder is designed in such a way that it enables almost instant allocation of jobs to the users. In effect, Work Binder acts as a mediator between client and server components of an arbitrary application in the grid environment.

2. Design of work binder. Work Binder consists of software components distributed in three tiers, at the client, worker, and central service, as shown in Figure 2.1:

- Various application-specific client programs interact with end user and invoke remote processing. They request jobs from the Work Binder service and communicates with obtained worker processes.

*University of Belgrade Computer Centre

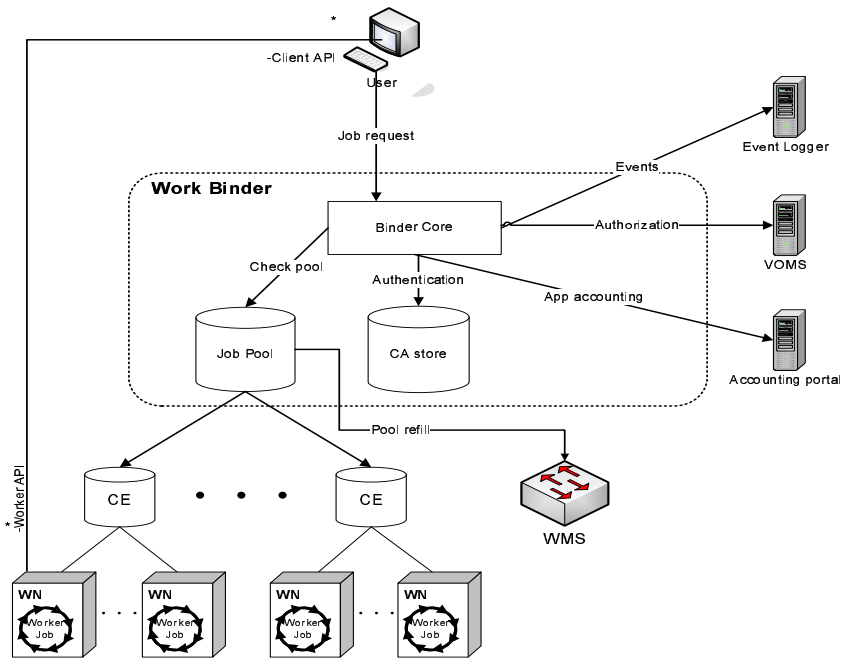


FIG. 2.1. *Components of Work Binder environment*

- Application-specific programs perform computation on grid worker nodes. Each program execution is called worker, and is being run within a grid job. It starts the actual application code (as a class or separate external program) after pairing with some client.
- Work Binder service is a mediator that maintains the pool of free workers and matchmakes them with clients. It is capable of simultaneous handling of several applications.

Key features that distinguish Work Binder, giving it a significant advantage over other related pilot infrastructure solutions [4] [5], include:

- Java API which allows quick & simple way to integrate new applications into the Binder & grid environment;
- Almost instant allocation of new jobs for users.
- Support for any number of different applications which will share the same pool of jobs;
- Adaptive dynamic allocation of new jobs;
- Work Binder hides the complexity of grid infrastructure from the user;
- Possibility to incorporate application-specific intercepting handlers into the mediator.

2.1. Submission of New Jobs. Work Binder submits worker jobs to the grid, maintains a pool of ready jobs, and mediates between clients and workers. It pairs clients and worker jobs from which it has received connections. A small dispatcher program started by the Worker Job then starts an application-dependent Java class or executable file installed on the targeted grid site. Further communication (what kind of communication will be used is specified by the client) between the client and the worker is continued using one of three possible methods:

- Via binder—All communication will go through the binder with the option to have a specific code on the binder for processing the data transferred between the client and the worker.
- Built-in direct communication—The worker connects directly to the client after the client has been matched to the worker and they continue their communication without going through the binder; network performance will probably be better, however not all clients are capable of accepting incoming connections.
- Custom communication—Defined using arbitrary application-specific mechanism; it is up to the client and the worker to implement it, the service only provides exchange of communication descriptors.

2.2. Management of the Job Pool. A pool of ready worker jobs is maintained using sophisticated algorithms with the goal of always having enough worker jobs in the pool for incoming clients as well as minimizing the stress on the grid infrastructure (the pool changes its size according to user demand). Having one pool of jobs for many different applications is an efficient way of conserving precious grid resources. When client connects to Work Binder and asks for a worker job, it will acquire a job almost instantly, providing there is one available. If not, all jobs in the pool are used up, and the client has an option to try again later.

The suitable size of the pool of jobs may vary significantly, depending on the current load and dynamics of user and job arrivals. For example, in order to handle interaction with individual end users, the interactive applications often need many processes for short periods of time. Such pattern of job allocation may be also attractive to some non-interactive applications that repeatedly run short jobs. However, such a behavior is extremely unsuitable for the current grid infrastructure. With Work Binder, a near-optimal usage of grid resources and high availability of ready jobs are archived by implementing an adaptive job submission and pool management policy. The policy shares jobs among several applications, reuses finished jobs, and adapts to the characteristics of grid sites and present workload.

Two modes of mediator operation, called idle and active, determine different job pool sizes:

- Idle mode of operation is used when there are no active clients. In this mode, pool of ready jobs is kept at a predefined minimum.
- Active mode of operation is used when the number of active clients is above zero. The desired pool size in this operation mode is determined using a number of estimation functions.

In order to calculate the desired pool size, it is necessary to take a few things into account. The first one is previously mentioned minimum pool size. The second is the amount of busy jobs in the pool—the more the pool is used, the more it is expected that it will be used in the near future. The third is the arrival frequency of new clients—if a lot of clients come in a short period of time, pool of jobs may be quickly used up.

In order to keep the pool of ready jobs at a desired size two job creation (job refill) strategies are used: regular and full throttle.

2.3. Regular Refill Strategy. Regular strategy is used for each CE individually and works over longer periods of time that are proportional to the response time of each particular CE. Response time in this case means the time in which the submitted job on the CE will become available to Work Binder.

The sum of available (ready) jobs for each CE used by Work Binder is the total number of ready jobs in the binder's pool. The goal of the regular refill strategy is to keep the number of ready jobs on each CE at the appropriate level by analyzing its current status.

The target pool size for each CE depends on the minimal and maximum pool size for the CE agreed between Work Binder administrator and the administrators of the particular CE and its load. The load preference is the measure of how heavily some CE is currently used by the binder.

The amount of jobs that will submitted on some CE is calculated from the target pool size for the CE and the current amount of ready jobs, but it also takes into account already submitted jobs that are expected to become ready (since it takes time for a submitted job to become available in the grid environment) and busy jobs that will finish soon and possibly become reusable (this is one of Work Binder's advantages, not available in the regular grid environment). It is important to note that the calculated number of jobs will not be submitted instantly, but rather gradually over time to compensate for the non-instant responsiveness of the computing elements.

2.4. Full Throttle Refill Strategy. Full throttle (panic) strategy is used to allocate jobs globally on Work Binder. Unlike the regular strategy which works slowly over periods of time, full throttle strategy submits jobs instantly as soon as it detects that the desired total amount of jobs is greater than the actual amount of ready jobs (and the jobs that are expected to become ready in a relatively short time). This can happen if the clients' arrival frequency becomes too high or if there are many busy jobs that saturate some of computing elements.

If panic refill strategy decides that some new jobs need to be submitted, it is necessary to make these jobs available as soon as possible. Work Binder keeps track of each CE's response time, and in order to provide new ready jobs quickly, it picks the group of computing elements with lowest response times and proportionally submits jobs to them. However, Work Binder will not submit and maintain more than the maximum of jobs agreed between the binder administrator and a CE administrator.

If fastest computing elements have reached their maximum pool sizes and there is a further need for panic refill, the amount of remaining refill jobs will be submitted to the rest of computing elements, by submitting as much as possible of new jobs on the fastest CEs first.

2.5. Work Binder Java API. The Work Binder Java API allows application users to easily integrate Work Binder with their existing applications. This integration can be achieved by using the appropriate API wrapper interfaces. Wrapper interfaces exist in all three tiers, and their purpose is to hide the complex internal communication from application users and provide them with simple interfaces and methods, so that they can focus on their own application specific problems and not bother with grid and binder infrastructure related issues. The API is designed in such a way, that the applications that already have some sort of client-server design can be easily adapted to use it (even if they were not gridified previously).

2.6. Client Interface. At the client side, *ClientConnector* is the wrapper interface through which users can connect to the Binder and obtain a Worker Job. Wrapper interface can be initiated with many options, some of them being:

- Binder Address—IP address of the Work Binder instance
- Binder Port—port on which the Work Binder instance is listening for incoming workers instances
- Application ID—unique name identifying which application supported by the binder will the client use
- Required Wall Clock Time—how long will the client use the Worker Job
- Access String—string describing the connection that will be used between the client and the worker
- Client's Certificate—user's grid certificate needed for authentication & authorization (every grid user has its own certificate that can be used to access various grid services)

Once a Worker Job has been obtained, the user can communicate to the Worker Job through *ClientConnector* interface methods or get the reference to the actual socket being used to communicate to the Worker job (this feature is not a preferred way of communication, but is provided in order to make life easier for applications that already have some sort of client-server communication implemented).

2.7. Binder Interface. Work Binder intercepts communication between the client and worker. One of the purposes of this intermediate level of communication, among other reasons, is to provide a way for clients and workers to connect to each other in environments that would not allow them to establish direct communication due to firewall or other network issues (Binder acts as a proxy server in this case). Another possible use is to add some application logic to the binder level where some processing can be performed on the Binder instead of on the remote worker job. The client chooses whether it wants to connect to the worker job via binder or directly (the details of direct connection can be exchanged via binder). If direct communication is chosen, this entire level of communication is skipped.

If communication via binder is used, the users have an option to implement application specific handlers that run at the binder by implementing the provided *BinderHandler* interface. If there is no application-specific implementation on Work Binder, a simple built-in proxy implementation is used where Work Binder acts as a proxy between the client and the Worker Job.

From within *BinderHandler* users are given access to an instance of the *BinderConnector* interface. This instance represents the wrapper interface at the binder. Similarly to the wrapper interface on the client, this class gives access to the communication primitives to the client on one and the worker on the other side, as well as direct access to the actual sockets used to communicate to the client and the worker.

2.8. Worker Interface. Worker level interface provides communication from within a worker job. Application users must implement their custom handlers by implementing the provided *WorkerHandler* interface. Users then get access to an instance of the *WorkerConnector* interface, which is also a wrapper interface. With this interface, as with the other two wrappers, users get access to communication primitives as well as low level access to the actual socket used for communication.

Application-specific *WorkerHandler* implementations are executed on remote grid sites, so they must be preinstalled and available to Work Binder in order to be executed successfully. Binder will detect the installed applications on each supporting site and keep track of them allowing application users to transparently add new applications to the Binder.

If an application user just wants to execute some external program located on the WN, instead of writing her own *WorkerHandler* implementation, there is an option to use one of built-in implementations. This way, an easy integration of programs written in various programming languages is possible, by providing execution

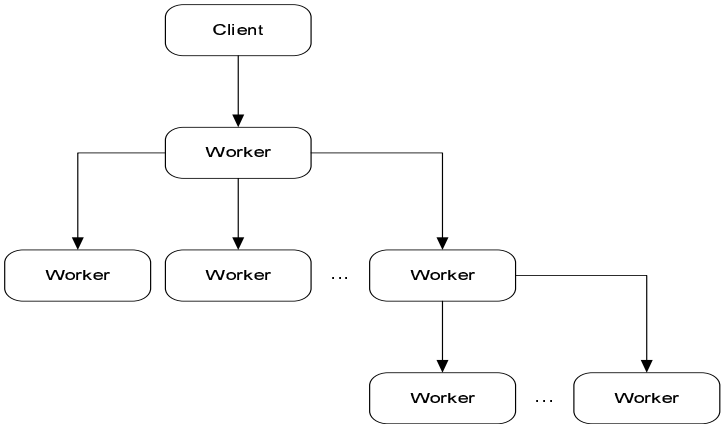


FIG. 2.2. *An example of dynamic job graph*

of application-defined programs or scripts, as well as assistance in establishing communication with the client. Two built-in implementations for two distinct use cases scenarios are available:

- *ExternalExecutor* is a *WorkerHandler* implementation that is intended for applications that want to establish direct communication between the worker and the client, where the worker is an external program located on the remote worker node. This implementation simply closes the connection to the binder and towards client, and executes a command line string on the worker with arguments provided by the client. It is up to the external program and the client to reestablish communication, if needed. For example, the client may inform the worker (using the program invocation parameters) about the address the worker should connect to.
- *ExternalListener* is the other *WorkerHandler* implementation intended for applications that want to use communication via binder, but the worker is an external program located on the remote WN. This implementation preserves the connection to the binder and towards client, starts listening on a local socket, and executes the external program to which it provides an address and the port on which it is listening as addition to command line arguments. The worker program is expected to connect to this socket (instead of connecting to the client) and communicate with the client normally using its own application-defined protocol as if there are no mediators between the worker and client. The client therefore does not notice any interruption of the communication and can start using the application-defined protocol after successful execution of *ClientConnector.connect()* method. Another benefit of this approach is that all communication will go through the binder. This scenario is useful in cases when the worker and the client cannot establish direct communication or when there is a need to have a specific code on the binder itself through which all the data transferred between the client and the worker will go.

An interesting feature is the ability of users to allocate several worker jobs by using the ClientConnector, not only from client machines, but also from workers. This way complex dynamic graph of jobs can be implemented, as shown in Figure 2.2. It is up to the application to orchestrate obtained workers that run in parallel.

3. Conclusion. The described approach will allow applications to use the grid in a new way, with a minimal additional load on the infrastructure. Users will have improved experience by having immediate availability to new jobs and the ability to uniformly access jobs from several grid sites using a simple programming model.

Three applications are currently being adapted to use this service, which will help its further evaluation and improvement. Also, a simplified interface for communication between the processes that use the described architecture is being implemented.

This research was supported by SEE-GRID-SCI project funded by the European Commission under the FP7 Research Infrastructures contract no. 211338.

REFERENCES

[1] *Pilot Jobs*, HEP Sysadmin Wiki, `http://www.sysadmin.hep.ac.uk/wiki/Pilot_Jobs`

[2] S. Burke, S. Campana, P.M. Lorenzo, C. Nater, R. Santinelli, A. Sciabí, *gLite 3.1 User Guide*, Tech. Rep. CERN-LCG-GDEIS-722398, Worldwide LHC Computing Grid, April 2008, https://edms.cern.ch/document/722398/
[3] *Work Binder Application Service*, EGEE SEE ROC and SEE-GRID Wiki http://wiki.egee-see.org/Work_Binder_Application_Service
[4] J. T. Moscicki, *DIANE—Distributed Analysis Environment for GRID-enabled Simulation and Analysis of Physics Data*, Nuclear Science Symposium Conference Record, 2003 IEEE Volume 3, 19–25 Oct. 2003, pp.1617–1620 Vol. 3.
[5] *DIANE: Distributed Analysis Environment*, http://it-proj-diane.web.cern.ch/it-proj-diane/

Edited by: Dana Petcu
Received: May 22nd, 2009
Accepted: Nov 3rd, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 419–428. http://www.scpe.org
ISSN 1895-1767

LOAD BALANCING METRICS FOR THE SOAJA FRAMEWORK

RICHARD OLEJNIK*, IYAD ALSHABANI*, BERNARD TOURSEL*, ERYK LASKOWSKI†, AND MAREK TUDRUJ†‡

Abstract. The paper describes system and program metrics used for load balancing algorithms for Java program execution in the SOAJA (Service Oriented Adaptative Java Applications) executive environment. This environment aims in maintaining design and execution of large scale computing tasks in complex networked Grid environments. SOAJA services provide means for static and dynamic load balancing with the use of special metrics obtained by Java object observation. SOAJA comprises mechanisms and algorithms for automatic placement and adaptation of application objects, in response to evolution of resource availability. Under control of SOAJA, parallel Java objects can be optimally allocated to Grid nodes before execution and next migrated at runtime to less loaded nodes to maintain the balance of loads of constituent JVMs. SOAJA mechanisms employ computation power metrics based on measurements of the idle time of processor nodes and communication bandwidth metrics for network resources based on statistical assessment of the existing traffic. Due to these mechanisms the granularity of computing and distribution of the application elements on the Grid platform can be optimally controlled.

Key words: service-oriented architecture, adaptive software, load balancing, grid computing, distributed computing

1. Introduction. The execution of an irregular distributed application usually generates a load imbalance, which brings down the overall effectiveness of the execution platform. Additionally, in a multi-user Grid context, the availability of computing resources can notably vary over time. Thus, an optimization subsystem, embedded in the execution environment (or in the distributed application) is essential.

Load balancing is one of important procedures applied to heuristically optimize execution time of parallel programs. A general classification and an overview of load balancing methods are presented in [4, 3]. The paper deals with an asynchronous approach to load balancing, in which the load balancing activities are performed in parallel with computations. Load balancing can be further divided into static load balancing where an computational load is partitioned among executive units by an algorithm executed before program execution and dynamic load balancing, where the load decomposition is adaptively changed during computations, following the system resources availability.

This paper is concerned equally with static and dynamic load balancing. In Java-based computing on Grid static load balancing has received relatively small attention. In this paper we present a two phase approach to load balancing of Java programs execution in Grid. The first phase consists of a static load balancing, which determines an initial deployment of application Java objects over the network of Java Virtual Machines. This static load balancing algorithm scenario includes execution of the application for a representative set of data to be able to detect some static properties concerning computational and communication aspects and to be able to use this properties for an initial deployment of program elements before execution. This phase of load balancing is based on tracing of the load of virtual machines and method invocations.

The second phase of the load balancing process is dynamically organized during program execution. It is based on three basic operations: JVM load observation, detection of the load imbalance and load migration if the imbalance exists. A dynamic agent approach is used to implement these operations. An observation system of the execution environment and the distributed application is an important part of dynamic load balancing. The system observes two areas:

a) the load of different nodes within the platform and the network,
b) the evolution of the applications.

Using the information provided by these observations, several metrics to detect and compute the load imbalance of processors have been proposed in the paper. They differ from standard measures known in the literature [2].

This paper describes SOAJA overall architecture and then deals with its internal concepts. The rest of the paper is composed of 5 parts. In the first part the general assumptions for the SOAJA framework are presented. Part 2 explains the use of web services in SOAJA. Part 3 discusses the relations between web services and functions of DG-ADAJ. Part 4 describes the load imbalance detection mechanisms in SOAJA. Part 5 describes the load imbalance correction mechanisms.

*Computer Science Laboratory of Lille (UMR CNRS 8022), University of Sciences and Technologies of Lille, France ({Richard.Olejnik, Iyad.Alshabani, Bernard.Toursel}@lifl.fr)
†Institute of Computer Science Polish Academy of Sciences, Warsaw, Poland ({laskowsk, tudruj}@ipipan.waw.pl)
‡Polish-Japanese Institute of Information Technology, Warsaw, Poland

2. SOAJA and ADAJ Frameworks. The SOAJA (Service Oriented Adaptative Java Applications) is a Java framework, which provides the infrastructure, components and services enabling a platform-independent execution of complex distributed data mining workflows [7]. It manages data access and resources sharing, including databases, information systems and hardware resources access. It supports resource discovery and provides context-aware recommendations for the dynamic composition of data mining operations and workflows. The underlying agent-based layer of the SOAJA infrastructure provides means to orchestrate very large, heterogeneous and dynamic workflows hardware with optimized load balancing across multiple nodes, constituting the Grid. The main services of the framework are observation, measuring of the JVM load, measuring of the physical processor load, load balancing service, and parallel data services.

The SOAJA environment is deployed in a Grid infrastructure, provided that an JVM is installed on each processor node. The SOAJA is the extension of the ADAJ environment to make it scalable on the Grid and to support service oriented architecture [5].

2.1. General Architecture of ADAJ. The design of distributed programs and optimization of their performance is not a simple task. Indeed, a programmer must consider construction of its application in the most effective way, while taking account of heterogeneity of the executive environment. To overcome these encumbrances, the ADAJ framework has been proposed [1].

ADAJ (Adaptative Distributed Application in Java) is a programming and execution environment for parallel and distributed Java applications. Design objectives of ADAJ are as follows: simplifying the programmer's work by hiding problems related to the management of parallelism, facilitating the development of applications and allowing their automatic or quasi-automatic deployment in heterogeneous environments, ensuring effective implementation of parallelism, by mechanisms of inter- and intra-applications load balancing.

The ADAJ provides mechanisms at middleware level that enable dynamic and automatic adaptation of computations to the structure of the executive platform and changes in resource availability. ADAJ includes four major features: a library containing the necessary tools to facilitate parallel programming, an observation system which scans the environment during its execution and retrieves the information necessary to optimize the program, a system that calculates the load of JVM and physical processors using the information gathered by the observation system, a system that allows correcting load imbalance by objects migration based on information from the observation and the calculation of loads.

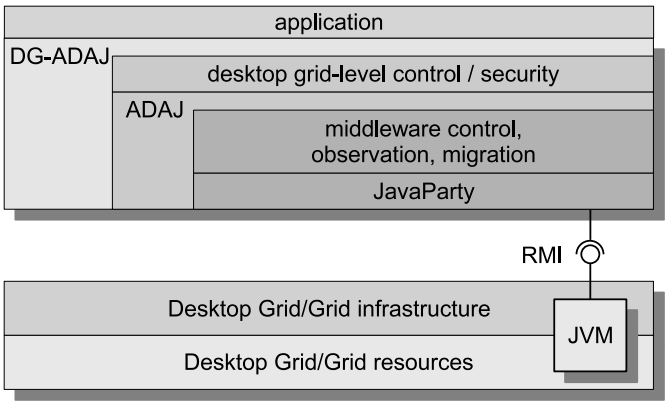


Fig. 2.1. *ADAJ layered architecture*

The ADAJ architecture is based on the multi-layered design pattern. The internal structure of the ADAJ consists of the following layers (Fig 2.1):

1. distributed programming layer (DPL) — JavaParty
   (a) provides a notion of global object (remotely accessible),
   (b) provides migration facility (used to move global objects between JVMs),
   (c) deploys an application on computing nodes.
2. load balancing layer — ADAJ core
   (a) constructs and maintains dynamic relationship graph, based on global objects' placement and observation data,
   (b) detects load imbalance and instructs local agents to equilibrate the load,

(c) depends on global objects, migration and deployment infrastructure provided by JavaParty.

3. application layer — Distributed Collections (Fragments)
   (a) provides an application construction framework, which supports programmers and eases distributed application development.

The current implementation of the ADAJ is based on JavaParty [14], which facilitates the usage of RMI protocol by application programmers. The JavaParty allows execution of distributed Java applications on workstations connected via a network. JavaParty has introduced the concept of remote objects that can be distributed in a transparent manner. It compensates for the drawbacks of the RMI protocol because it conceals the addressing and communication mechanisms. Using JavaParty, it is sufficient to annotate Java classes with the word `remote` that will give access to remote objects from any part of an application without publishing them explicitly in service names space as RMI.

DG-ADAJ is a version of the ADAJ environment implemented for Desktop Grids. Initially, DG-ADAJ was conceived as an extension of the ADAJ system (see [10]), built for cluster computing. It has been re-engineered to extend its for larger scale distributed computing and to introduce some special security mechanisms, which provide reliable application execution in Desktop Grids [12].

2.2. Web Services in SOAJA. SOAJA uses WSRF [9] standard and allows instantiation of statefull services for applications. The statefullness of services is exploited by the application clients that allows asynchronous communication among software components instantiated for the client on different platforms.

The Fig. 2.2 illustrates the main web services that constitutes the SOAJA environment.

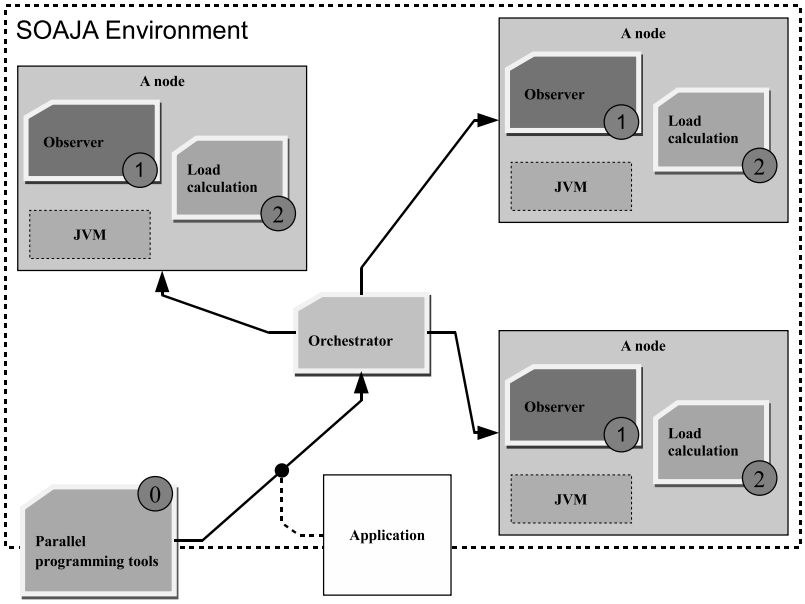


Fig. 2.2. *SOAJA environment*

Various architectural styles and middleware components can be used for the implementation of a SOA-based environment. The Enterprise Service Bus (ESB) is a middleware technology which provides the necessary features with which it is possible to implement SOA. In a typical ESB usage scenario, the ESB layer itself is deployed over the existing infrastructure. Based on this infrastructure, the ESB layer offers the necessary support for transport interconnections while it exposes the existing subsystems through a specific set of adapters. With the help of the ESB as a SOAJA implementation technology, services are exposed in a uniform manner, using open standards, so that any client, who is able to consume web services over a generic or specific transport, is able to access them. For example, the SOAJA has been used to implement data mining application in the WODKA project [11].

SOAJA is a platform developed on top of DG-ADAJ, by adding the web services layer and gaining SOA-specific properties, Fig 2.3. With computing grid as a main target, the SOAJA platform provides a uniform and transparent interface to the infrastructure of the DG-ADAJ platform. The SOAJA platform, through its

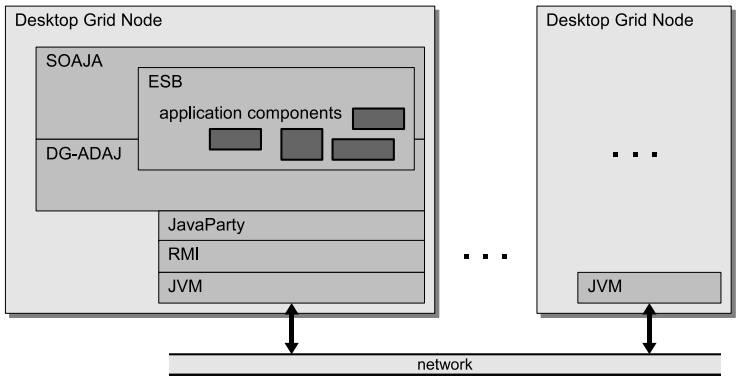


FIG. 2.3. *SOAJA layered architecture*

orchestration layer, provides also a possibility to execute single-cluster applications in the DG-ADAJ environment [1, 13]. With DG-ADAJ executions controlled via an ESB, the orchestration layer is able to offer both the support for execution of complex compositions, as well as elementary execution of applications in the underlying DG-ADAJ environment. This way, the SOAJA environment hides implementation details of different Grid environments behind the set of various web services, offering the necessary support for integration, interoperability and reliable messaging. As a side effect, by employing the orchestration layer, we enable a programming-in-the-large paradigm necessary to assure the development and support of long living, asynchronous processes.

The SOAJA platform is going to help efficient execution of heterogeneous applications enabled by DG-ADAJ by offering basic support for workflow deployment and enactment. The DG-ADAJ environment could thus be freed from some of the placement, distribution and execution tasks, by moving significant parts of these to the higher layer of the ESB and the enactment environment. With SOAJA, the ability to design component-based and service oriented applications, developed over the DG-ADAJ platform is extended with the help of web services open standards, by offering the possibility to access both local and remote components, eventually deployed on different DG-ADAJ environments.

3. Program execution optimization in SOAJA. Load balancing is the fundamental procedure applied in order to optimize execution time of parallel programs. In the case of the architecture of the SAOJA environment, the load balancing is achieved by such a distribution of the application components (objects) among active Grid nodes that guarantee a possibly high efficiency of the overall application execution. To make the load balancing effective during the whole run-time of an application, the SOAJA employs equally both static and dynamic load balancing. This is accomplished by the optimization of the following two aspects of application execution: initial objects deployment and dynamic load balancing.

3.1. Initial object deployment optimization. An optimization feature of SOAJA is an initial application objects deployment on JVMs, which results in a shorter execution time. The initial placement of application objects to JVMs is the service of the orchestrator shown in Fig. 2.2. The initial object deployment optimization algorithm follows the pattern of static parallelization method in multithreaded Java program. It defines decomposition of Java code into parallel threads distributed on a set of JVMs, so as to reduce program execution time [13]. The control decisions are taken to determine which of the classes should be distributed and what the mapping of objects and data components (fragments) should be to JVM nodes, to reduce direct inter-object communication and to balance loads of the JVMs. When applied to an application run under DG-ADAJ control, it will determine an initial distribution of its objects among Java Virtual Machines (JVMs) assigned to Grid active nodes, thus leading to a reduction of the total execution time.

The proposed optimization algorithm employs an image of application program, based on an analysis of the byte code generated by Java compiler. This analysis identifies control dependencies between byte code instructions. They are represented in adequate MDG (Method Dependence Graph) and MCG (Method Call Graph) graphs of the program [6, 13]. The number of mutual method calls and the number of thread spawns during program execution for representative input data are measured using observation mechanisms described later in the paper. Program behavior, including all created objects in each class, all called methods, as well as all spawned threads, are registered in trace files.

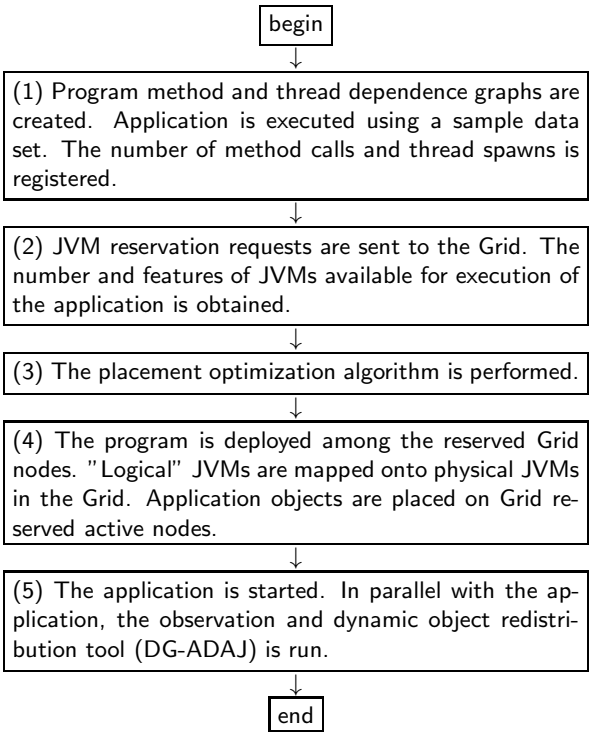


FIG. 3.1. *The control flow of an application execution.*

The flow of actions during application execution is shown in the diagram in Fig. 3.1. The first three blocks in the diagram determine an initial optimized placement of application objects and perform the respective objects distribution over Grid on JVMs nodes. This part of the algorithm starts with execution of the application using some representative sample data. For that, the number of available JVMs nodes on the Grid must be known. The number of method calls and spawned threads that have appeared during execution is recorded. Next, the program method and thread dependence graphs are annotated with the recorded data.

At the beginning of the object deployment optimization algorithm we treat all objects as remote objects (respectively all classes are distributed classes). Based on the recorded control data, the algorithm decides which classes should remain distributed and how the involved objects should be placed on a set of JVMs assigned to active nodes on the Grid. We use the heuristics based on the following principles: the strong locality of method calls has to be preserved inside each parallel thread and the number of inter-thread calls, which cross the boundaries of JVMs has to be reduced. To fulfill such object requirements we designate all calls inside a single thread to the same JVM and optimize distribution of threads across available sets of JVMs by applying load balancing methods.

The algorithm consists of two phases (see [6] for details). In the first phase the MCG graph is traversed in the DFS (Deep-First-Search) manner to agglomerate method calls executed in single thread. In this step, the algorithm finds the MCG subgraphs, which are constructed of vertices connected by edges connecting calls inside threads. We assume that each subgraph is executed on a dedicated JVM. Subgraphs are built at the level of single objects. In case when an object belongs to different subgraphs, the new subgraph is constructed and a unique JVM number is assigned to it. We expect that, in most cases, at the end of this phase, the number of found subgraphs is far bigger than the number of available JVMs in the system.

In the second phase of the algorithm, we clusterize the subgraphs obtained in the previous phase until the number of clusters is equal to the number of JVMs. The general outline of this phase is similar to Sarkar's [15] edge-zeroing clustering heuristics. At each clustering step, the algorithm finds the subgraphs, which are connected by edges with the biggest weight value and which connect nodes placed on different JVMs. All nodes of those subgraphs are assigned to the same JVM while the total number of remote calls decreases. The algorithm stops when the number of clusters is equal to the assumed number of JVMs.

The first phase of the algorithm makes that method calls inside programmer-declared threads are local to the JVM. This allows exploiting thread level parallelism without introducing large inter-JVM communication overheads. The gain of the second step of the heuristics comes from reduction of RMI calls to remote objects.

3.2. Load balancing strategy. The workstations used in the network are heterogeneous, but they have different and variable computing capabilities over time. The load imbalance occurs when the differences in workload between the workstations become too big. An application execution may not be optimal because some workstations have too much work and the others have not enough. Let's consider that the network works correctly and that the DG-ADAJ environment is running. We distinguish two main steps in load balancing: detection of imbalance and its correction, if necessary. The first step uses measurement tools to know the functional state of workstations. The second consists in migrating of the load from overloaded workstations to underloaded workstations in order to balance the workload.

The observation mechanism of applications in the DG-ADAJ environment aims at providing knowledge of the applications behavior during their execution. This knowledge is gathered by observing activity of constituent objects.

There are two types of objects in DG-ADAJ (Fig. 3.2):

global objects These are global objects that can be created remotely in any JVM. They are remote accessible. There is only one copy of a global object in all environment. The global objects are also migratable, i. e. they can be moved from one JVM to another.

local objects These objects are traditional Java objects. They can be used in only one JVM at the place where they reside. If another JVM needs such object, it will create of a new copy of the object concerned. Obviously, local objects cannot be migrated.

We observe only global objects as the observation and migration of all objects would generate a considerable work overhead compared to the profits brought by load balancing.

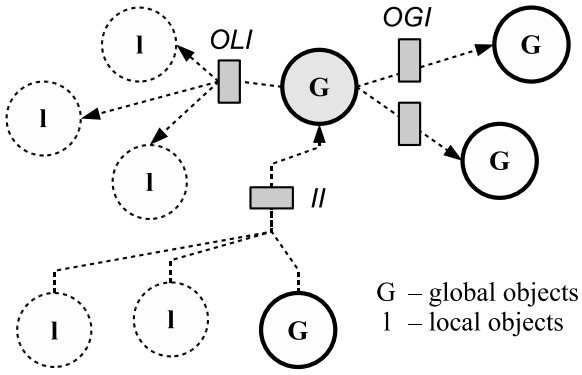


FIG. 3.2. *Relations between objects*

4. Detection of load imbalance. In this first phase, the purpose is to obtain knowledge about of the functional state of workstations composing the cluster. As the environment is heterogeneous, it is necessary to know not only the load of workstations but also their working capabilities. Such capability is directly related to the workstation computing power, so we have to estimate it. This measurement needs to be made only once when the workstation joins the system. We call it *the calibration* in this paper.

We then need the workstations workload measurement at a given time. For this purpose, we observe this part of the CPU time that is used by DG-ADAJ applications. However, such measurement is not normalized and cannot serve for comparisons between different workstations. It is thus necessary to balance it against different computing power of the workstations. After a series of measures, we compare the values found on different machines and we determine if there is a load imbalance.

The heterogeneity disallows us to compare measurements taken on workstations whose computing powers are different. Before we are able to compare the workstations load, we have to normalize the measurements. The normalization of the workloads is made by means of the power indications. After experiments to determine the workstation power, we found that the formula, which allows us to compare the workstations load is the

product of the power index and the CPU time use rate by a thread, which gives the availability index of a CPU:

$$Ind_{availability} = Ind_{power} * \%Time_{CPU}$$

At this point, we have availability indices of all workstations. By comparing these indices, we will be able to detect the load imbalance. An imbalance is characterized by too big dispersion of the availability indices of workstations composing the network. It is difficult to fix a threshold not to be overpassed to characterize a load imbalance. But we can define an interval in which the dispersion of the availability indices remains acceptable at a given moment. We are going to be interested in the gap between the minimal and maximal availability indices found during a series of measures. If the distance between these values is too big, we conclude that there is a load imbalance, which can to be considered using the following condition:

$$Stability = (\max(Ind_{availability}) \leq \alpha * \min(Ind_{availability}))$$

If this inequality is verified, the availability indices are close enough not to present a too big imbalance. Otherwise, the range of the values is too big and an imbalance of workload between workstations is detected. The central point of this inequality is the value taken by the coefficient α. We tried to clarify it first by using statistical tools then by using an experimental way. We cannot finally give a unique value for this coefficient. Nevertheless, we can restrict its value to the interval $[1.5 \ldots 2.5]$. These experimental values give good results because they are neither too restrictive nor too tolerant for the load imbalance.

The tolerance for the load imbalance among all workstations depends directly on the value of the coefficient α. The smaller this coefficient is, the more we are demanding as to the load balance between workstations. Indeed, a low coefficient allows only a small difference between loads of workstations. This can be seen as a guarantee of the quality of the balancing results. However, a low tolerance of the imbalance leads to much more frequent detection of the imbalance. The consequence is to activate the mechanism of load balancing more frequently and therefore make it more expensive in time. Not only the migrated objects can not do their computing, but they can also block the execution of others objects, which are awaiting results. In addition, the coefficient α is to be chosen according to the number of machines in the network. When this number is big, the coefficient must be increased and vise versa.

5. Correction of load imbalance. In this phase, we are in the state in which a load imbalance has been detected. To correct this imbalance, we have to classify the workstations into three categories according to their availability index:

1. **Overloaded** workstations: availability indices are low,
2. **Normally loaded** workstations: availability indices are medium,
3. **Underloaded** workstations: availability indices are high.

The purpose of the load balancing is to transform the workstations categories: overloaded and underloaded into the category normally loaded. To do it, we need to migrate the workstations load from the first category to the third one.

5.1. Classification of workstations. We use the K-Means algorithm [8] to build the categories of workstations based on the computed availability indices. The K-Means algorithm allows to classify a distribution of n values into k categories by choosing k centers for categories. We want to classify workstations into the categories: overloaded, normally loaded and underloaded. For this, we use this algorithm by taking the computed availability indices and $k = 3$, to obtain 3 categories finally.

The three centers that we choose are the minimum, average and maximum availability indices. The average index is simply the average of indices measured during the last series of measures over the whole network. By comparing the distances of workstations availability indices from the three centers, the three categories of workstations will be identified. The center represented by the minimum index builds the category of overloaded workstations, the center using the average one builds the category of normally loaded workstations and finally the center based on the maximum index is used to build the category of underloaded workstations. The important thing is therefore to have the overloaded and underloaded categories in order to be able to move the load from the overloaded workstations to the underloaded workstations.

5.2. Choice of candidates for migration. To correct load imbalance, we have to migrate the load from overloaded workstations to underloaded workstations. Firstly, we must identify the load that we want to migrate. The loads are represented by the activities of the objects which are running on the JVMs. We need

to select an object on each JVM in the overloaded nodes. Let's see how to choose such an object, so that its migration changes the load balance in the network.

The migrated entity is necessarily a global object because it is not intrinsically linked to the JVM on which it currently runs. Among the global objects in a JVM, some of them have more suitable characteristics to be migrated. These characteristics are related with the other computer objects and the load quantity carried out. Two relations are involved:

a) the attraction of a global object to the JVM,
b) the weight of the global object.

The attraction of a global object to a JVM is expressed in terms of communication links, which it shares with other global objects inside the same JVM on the computing node. A strong attraction involves frequent communication, which will be realized as remote communication after object migration. This communication will then have a higher cost than the current cost. A small attraction will permit to leave a global object the current computing node and to run it on another one, without introducing significant amount of additional remote communication. So, the less the object is attracted by the current JVM, the more interesting it is to be selected as a migration candidate.

The computational weight of migrated global object gives the quantity of load to be removed from the current machine. An object, whose quantity of work is big i. e. shows a continuous activity, should not be migrated. In addition, by migrating a too big quantity of work (load), we could reverse the role of the involved machines (the source will become target and vice-versa). In contrast, the migration of an object with small quantity of a work does not bring significant load variation improvements. Furthermore, the migration cost will not be compensated with the new generated load distribution. In conclusion, the decision should be to move an object whose quantity of work is neither too big, nor too small. Thus, the smaller the distance is to the average object loads, the more the object is interesting for migration.

The observation of the objects activity is done by counting the activation methods. These activations can be done by global or local objects. The observation of a global object (only these objects are observable), includes (see Fig. 3.2):

1. observation of object invocations to each global object, including him: **OGI** (OutputGlobalInvocation),
2. the observation of object invocations to all local objects: **OLI** (OutputLocalInvocation),
3. the observation of others objects invocations to the considered object: **II** (InputInvocation).

The attraction of the global object *obj* to the actual JVM:

$$attr(obj) = \sum_{o \in JVM} (OGI(obj, o) + OGI(o, obj))$$

Distance compared to the average quantity of work of the *obj*:

$$dist_{m_{WP}}(obj) = |WP_{obj} - m_{WP}|$$

Where $m_{WP} = \frac{\sum_{o \in JVM} WP_{obj}}{n}$ (n is the number of global objects on the JVM) and

$$WP_{obj} = OGI(obj, obj) + II(obj) + OLI(obj)$$

These formulas allow to compute the attraction of an object to the local JVM in order to compare it with the attractions of other objects of this JVM. The comparison formulas are:

1. Global attraction measure:

$$\%attr(obj) = \frac{attr(obj)}{\sum_{o \in JVM} attr(o)}$$

2. Migration distance, compared to the average quantity of work of the object:

$$\%dist_{m_{WP}}(obj) = \frac{dist_{m_{WP}}(obj)}{\sum_{o \in JVM} dist_{m_{WP}}(o)}$$

The most interesting object to migrate is selected based on the weighted sum of these relations:

$$Classification(obj) =$$
$$\alpha_{attr} * \%attr(obj) + (1 - \alpha_{attr}) * \%dist_{m_{WP}}(obj)$$

α_{attr} is a real between 0 and 1. Its choice remains experimental. Let us notice however that the bigger α_{attr} is, the bigger is the weight of the object attraction.

5.3. Selection of the target for migration. The migration of global objects reduces the load of a JVM running on an overloaded workstation and therefore the global load cost of this workstation. The question now is: where migrate these objects? Naturally, the potential destinations are one or more underloaded workstations. However, the choice of one of these computing nodes can be more or less convenient from the point of view of work and communication quantity of the object to migrate. For an object selected for migration, we must find the best target according to these criteria.

The first criterion to quality as a target is the attraction of the selected object to this workstation. We say that the attraction of an object—candidate for migration, is big when it communicates a lot with the global objects in the target JVM. A relationship of attraction of the global object *obj* to JVM_i is defined as follows:

$$attrext_i = \sum_{objext \in JVM_i} (OGI(objext, obj) + OGI(obj, objext))$$

The more the object is externally attracted by an underloaded machine, the more it is interesting that this machine is chosen as a migration destination for it. This criterion should not be the only one during selection of target for migration. In fact, it is possible that two underloaded workstations have the same amount of communication with the candidate for migration. In this case, the second criterion will be the workstations' availability indices. We naturally prefer the one whose availability index is the highest, because it is actually the least loaded.

To complete discussion of the criteria for choosing a target for migration, we should take into account the number of *(waiting)* threads in the JVM of the potential targets. We consider them, however, as potential load, which must be taken under consideration with the related load currently done on the machine.

These three points are obvious constraints to be met by the target machine:

1. the external attraction of the object is maximal to the target machine,
2. the quantity of work on the target machine is minimal,
3. the number of waiting Java threads is minimal.

These three conditions are gathered in a formula in order to designate the underloaded workstation, which is the most favorable for the selected object migration. Firstly, we have to normalize all the values related in the interval $[0 \dots 1]$. We then obtain:

$$\%attrext_i = \frac{attrext_i}{\sum_j attrext_j}$$

$$\%Ind^*_{availability_i} = \frac{Ind^*_{availability_i}}{\sum_j Ind^*_{availability_j}}$$

where

$$Ind^*_{availability} = Ind_{availability} - Ind_{availability} * \frac{NbThread_{wait}}{NbThread_{total}}.$$

The availability index was corrected by the potential work of the workstation, represented by the waiting threads. We have not considered this index during the classification of workstations into three categories, because we want to classify workstations according to their measured quantity of work and not a potential one. Indeed, threads waiting must be seen as supplementary work about which we know nothing. They may start executing in one second quite well as in one hour. Their consideration is thus justified only when we want to examine our measurements in perspective as it is the case here. We thus chose to decrease the raw indication of availability by means of the relationship between the number of waiting threads and the total number of threads staying in the machine.

The aggregation function should account for the two described components by giving them different weights. The balanced sum of them is:

$$Quality_i = \alpha_q * \%attrext_i + \beta_q * \%Ind^*_{availability_i}$$

with α_q and $\beta_q \in [0 \dots 1]$.

For an object which is a candidate for migration, this formula is applied to all JVM potential node targets. The workstation which maximizes this sum will be chosen as new location for the object. The choice of coefficients α_q and β_q is experimental but the sum of the two must be equal to 1 (it was therefore $\alpha_q = 1 - \beta_q$). The weight of one or the other value can be increased by changing these coefficients. The migration of the object allows to eliminate communication on the network and to reduce considerably the waiting time for replies. For that purpose, the coefficient α_q must be the most important to promote the machines for which the attraction of the object is maximal. For example, we can use the coefficients $\alpha_q = 0.6$ and $\beta_q = 0.4$.

6. Conclusions. The paper has described a run-time environment SOAJA for Java program execution optimization to provide load balancing of JVMs implemented on the Grid platform. The SOAJA mechanisms first adjust the initial parallelization granularity and placement of the application program elements on JVMs. Then, the objects distribution is adjusted to the evolution of the availability of system resources at run time by object migration. To optimize the initial object deployment and JVMs load balancing the SOAJA infrastructure applies system and program metrics obtained by its special observation mechanisms. They are used by components and services to enable static and dynamic load balancing of the nodes of a Grid. The system is currently under implementation inside the GRID 5000 project.

REFERENCES

[1] I. Alshabani, R. Olejnik and B. Toursel. *Parallel Tools for a Distributed Component Framework*. 1st International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA04). Damascus, Syria, April 2004.
[2] J. Cao et al., *Grid load balancing using intelligent agents*, Future Generation Computer Systems, 21 (2005), pp. 135—149.
[3] K. Devine et al., *New challenges in dynamic load balancing*, Applied Numerical Mathematics, 52 [2005], pp. 133—152, Elsevier.
[4] R. Diekmann, B. Monien, R. Preis, *Load Balancing Strategies for Distributed Memory Machines*, Karsch/Monien/Satz (ed.): "Multi-Scale Phenomena and their Simulation" World Scientific, pp. 255-266, 1997.
[5] T. Erl, *Service-oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River: Prentice Hall PTR, 2005. ISBN 0-13-185858-0.
[6] V. Felea, E. Laskowski, B. Toursel, M. Tudruj, *Optimizing Object Oriented Programs Based on the Byte Code-Defined Data Dependence Graphs*, Procs. of Concurrent Information Processing and Computing (CIPC NATO ARW), Sinaia, Romania, pp. 34–46, 2003.
[7] V. Fiolet, G. Lefait, R. Olejnik, B. Toursel, *Optimal Grid Exploitation Algorithms for Data Mining*, In Proc. of ISPDC 2006, IEEE Computer Society, July 2006, pp. 246-252.
[8] J. A. Hartigan et M. A. Wong, *A K-Means clustering algorithm*, Applied statistics, Vol. 28, pp. 100-108, 1979.
[9] OASIS Web Services Resource Framework (WSRF) `http://www.oasis-open.org/committees/wsrf/`
[10] R. Olejnik, A. Bouchi, B. Toursel. *An Object Observation for a Java Adaptative Distributed Application Platform.* Intl. Conference on Parallel Computing in Electrical Engineering PARELEC 2002, pp. 171-176, Warsaw, Poland, September 2002.
[11] R. Olejnik, F. Fortis, B. Toursel, *Webservices Oriented Datamining in Knowledge Architecture*, Accepted to publication in Future Generation Computer System (FGCS)—The International Journal of Grid Computing: Theory, Methods and Applications
[12] R. Olejnik, B. Toursel, M. Ganzha, M. Paprzycki, *Combining Software Agents and Grid Middleware*, Advanced in Grid and Pervasive Computing, C. Cerin and K.-C Li Editors, LNCS 4459, pp. 678-685, Springer Verlag, Berlin, Heidelberg, 2007.
[13] Olejnik R., Toursel B., Tudruj M., Laskowski E., *Byte-code scheduling of Java programs with branches for desktop grid*, Future Generation Computer Systems, Vol. 23, Issue 8, November 2007, pp. 977-982, ©Elsevier Science.
[14] M. Philippsen, M. Zenger. *JavaParty—Transparent Remote Objects in Java.* Concurrency; Practice & Experience, Vol. 9. No. 11. pp. 1225-1242. November 1997.
[15] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors.* The MIT Press, 1989.

Edited by: Dana Petcu
Received: September 30th, 2009
Accepted: November 3rd, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 429–441. http://www.scpe.org
ISSN 1895-1767

PERFORMANCE STUDY OF THE FIRST THREE INTEL MULTICORE PROCESSORS

AMI MAROWKA*

Abstract. The transition from sequential computing to parallel computing represents the next turning point in the way software engineers design and write software. This paradigm shift leads the integration of parallel programming standards for high-end shared-memory machine architectures into desktop programming environments. In this paper we present a performance study of these new systems. We evaluate the performance of an OpenMP shared-memory programming model that is integrated into Microsoft Visual Studio C++ 2005 and Intel C++ compilers on a multicore processor. We benchmarked using the NAS OpenMP high-level applications benchmarks and the EPCC OpenMP low-level benchmarks. We report the basic timings and runtime profiles of each benchmark and analyze the running results.

Key words: multicore, openMP, NPB, micro-benchmarks

1. Introduction. For many years parallel computers have been used only by an exclusive scientific niche. Only universities and research institutions backed by government budgets or funded by multi-billion-dollar companies could afford to purchase state-of-the-art parallel machines. Multiprocessor machines are very expensive and demand expertise in system administration and programming skills. Parallel computing therefore remains a specialized field of an exclusive community.

Now, two complementary technologies bring parallel computing to the desktop. On the hardware side is the multicore processor for desktop computers, and on the software side is the integration of the OpenMP parallel programming model into Microsoft Visual C++ 2005. These technologies promise massive exposure to parallel computing that nobody can ignore, thus making a technology shift unavoidable.

The dual-core processors first appeared on the market six years ago [1]. The chip makers Sun and IBM were the first: Sun introduced the Microprocessor Architecture for Java Computing and IBM launched the Power 4 dual-core processor. These processors, like their predecessors, were expensive and optimized for special-purpose computing with intensive tasks running on high-end servers. The greatest change came with the dual-core processors that AMD and Intel launched in 2005. Designed for desktop computers, these processors caused prices to drop; thus, a desktop computer with a dual-core processor can now be bought for less than $500. This price is affordable for students and computer science departments alike. But dual-core processors are only the beginning. The chip makers are now working on the next generation of multicore processors that will contain 4, 8, and 16 cores on a single die. Unfortunately, writing a parallel code is more complex than writing a serial code [2]. Parallel programming is extremely difficult. This is where the OpenMP programming model comes into the picture [3]. OpenMP helps developers to create multithreaded applications more easily while retaining the look and feel of serial programming.

The extra development effort and code complexity of parallel programming give rise to an obvious question– Is it worthwhile? The best way to answer this question is by benchmarking. This paper presents a performance study of OpenMP shared-memory programming model [3] that was integrated into Microsoft Visual Studio C++ 2005 and Intel C++ compilers on multicore processors. The benchmarking was conducted using the NAS OpenMP parallel benchmark suite [4] with different sizes of input classes, and the EPCC OpenMP directives benchmarks [7, 12]. We report the basic timings and runtime profiles of each benchmark and analyze the running results. A preliminary conference version paper of the study presented in this paper is [14]. This paper presents a completed study and includes details and materials that have been studied since the first version, such as related-works section; more tables and figures; more detailed tables and figures; benchmarking of Intel Quad-core machine; and clearer presentation of comparison between different architectures by clock cycles.

The rest of this paper is organized as follows. Section 2 presents related work. In Sections 3, 4, and 5 we provide brief overviews of the OpenMP, NPB benchmarks, and EPCC micro-benchmarks respectively. Section 6 is an in-depth analysis of the benchmarks results and Section 7 presents our conclusions.

2. Related Work. Multicore processors are ubiquitous and therefore they are studied intensively from many aspects. Many chipmakers offer today multicore processors with different architectures. The work presented in this paper covers the first generations of multicore processors from Intel. The performances of these

*Department of Computer Science, Shenkar College of Engineering and Design, Israel amimar2@yahoo.com

processors are compared by using OpenMP NPB benchmarks and the EPCC micro-benchmarks on MS Windows operating system.

Chunhua Liao et al [15] reported on experiments of OpenMP on Sun Fire V490 with Chip Multiprocessing (Solaris 10 operating system) and a Dell Precision 450 workstation with Simultaneous MultiThreading(SMT) technology (Linux kernel 2.6.3 SMP(Symmetric Multi-Processor) operating system). For this study they used EPCC Microbenchmark suite, subsets of the benchmarks in SPEC OMPM2001 and the NAS parallel benchmark 3.0 suites. The main conclusions of this study are that a straightforward OpenMP implementation for traditional SMP architecture may not achieve good scalability on the Xeon system because of memory bandwidth bottleneck and competition for the shared computing resources. They found that a HyperThreading-aware OS is important for maintaining load balance and efficiently utilizing the resources of an SMT system and that the EPCC micro-benchmarks results also indicated that the overhead of OpenMP synchronization implementation in a SMT system is higher than that in an SMP system.

Kent Milfeld et al [16] studied the performance of two dual-core processors, Intel dual-core Woodcrest (RedHat OS) and AMD dual-core Opteron (SuSE OS). In this work the costs of creating Pthreads and OpenMP thread were investigated and the performance of thread/process scheduling and affinity in multi-cache systems and the thread synchronization methods and costs were evaluated. Also, the L2 Cache Characteristics (latency) and the impact of a large-page memory on the performance of four NPB-MPI benchmarks were studied. The main conclusions of the authors are that applications need to be optimized for CLP, but it is still uncertain which cache system, independent or shared, will be best. Moreover, shared L2 cache suffers from contention and low level synchronization methods for multiple cores and thus will need to be implemented in thread control at the user-program level to assure locality of the data. Nevertheless, independent cache systems are contention free at the L2 cache level, and are probably more ideal for MPI codes, which employ non-shared paradigms.

Furlinger Karl et al [17] studied the scalability characteristics of medium and large variants of the SPEC OpenMP benchmarks on large-scale shared memory multiprocessor machines using their own OpenMP profiling tool, ompP. The experiments were conducted on 2 to 32 processors on a 32 processor SGI Alitx 3700 with the medium variant while the large variant were tested on 32 to 128 processors, with increments of 16, of a larger Altix 4700 machine. Four overheads categories used to evaluate the scalability: (S) corresponds to synchronization overhead, (I) represents overhead due to imbalance, (L) denotes limited parallelism overhead and (M) signals thread management overhead. The results show that 4 of 7 medium applications and 3 of 5 large applications achieved poor scalability due to load imbalance, thread management overhead, and small parallel loops.

Grant and Afsahi [18] studied the optimal operating configuration of Hybrid chip multithreaded SMPs for scientific applications and to identify the shared resources that might become a bottleneck to performance under the different hardware configurations. The authors investigated a two-way dual-core Hyper-Threaded (HT) Intel Xeon SMP server under single program and multi-program multithreaded workloads using the NAS OpenMP benchmark suite. The experiments were conducted on a Dell PowerEdge 2850 SMP server with Red Hat Linux Enterprise WS 4.1 distribution with Kernel 2.6.9-11 while LMbench tool used for measuring the L1, L2, and main memory latencies of the processor. The performance results indicate that in the single-program case, the CMP-based SMP and CMT-based SMP configurations have the highest average speedup across all of the applications. The most efficient architecture is a single HT-enabled dual-core processor that is almost comparable to the performance of a 2-way dual-core HT disabled system.

Curtis-Maury et al [19] evaluated the performance of OpenMP applications on simulated CMP and SMT architectures using Simics simulation platform. The simulations evaluated the performance of NAS Parallel Benchmarks suite. The authors found that the high level of resource sharing in SMTs results in performance complications, should more than 1 thread be assigned on a single physical processor. CMPs, on the other hand, are an attractive alternative. Their results show that the exploitation of the multiple processor cores on each chip results in significant performance benefits. Moreover, the execution of multiple threads on each processor is more efficient and predictable on CMPs than it is on SMTs due to the higher degree of resource isolation, which results in fewer conflicts between threads co-executing on the same processor. Although adaptive run-time techniques can improve the performance of OpenMP applications on SMTs, inherent architectural bottlenecks hinder the efficient exploitation of these processors.

3. OpenMP Programming Model. OpenMP is a tool for writing multi-threaded applications in a shared memory environment [3]. It consists of a set of compiler directives and library routines. The compiler

generates a multi-threaded code based on the specified directives. OpenMP is essentially a comparatively recent standardization SMP (Symmetric Multi-Processor) development and practice. By using OpenMP, it is relatively easy to create parallel applications in FORTRAN, C, and C++. Compiler and third party applications support is becoming more common.

An OpenMP program begins with a single thread of execution called the master thread. The master thread spawns teams of threads in response to OpenMP directives, which perform work in parallel. Parallelism is thus added incrementally: the serial program evolves into a parallel one. OpenMP directives are inserted at key locations in the source code. These directives take the form of comments in FORTRAN and pragmas in C and C++. The compiler interprets the directives and creates the necessary code to parallelize the indicated tasks/regions. The parallel region is the basic construct that creates a team of threads and initiates parallel execution. Most OpenMP directives apply to structured blocks, which are blocks of code with one entry point at the top and one exit point at the bottom. The number of threads created when entering parallel regions is controlled by the value of the environment variable OMP_NUM_THREADS. The number of threads can also be set by a function call from within the program, which takes precedence over the environment variable. It is possible to vary the number of threads created in subsequent parallel regions. Each thread executes the block of code enclosed by the parallel region.

In general, there is no synchronization between threads. Different threads may reach the end of the parallel region at different times. OpenMP does provide constructs for synchronization, but the code should not be written in such a way that its output depends upon different threads executing statements at particular times. OpenMP provides a number of constructs for thread synchronization and coordination, among them critical, atomic, barrier, and master. These are sufficient for many needs, but OpenMP also provides a set of runtime thread-locking functions that can be used for fine control. When all threads reach the end of the parallel region, all but the master thread go out of existence and the master continues alone. The OpenMP directive clauses—Private, Shared, and Default—control whether the listed variables are shared among different threads or are private (local) to each thread.

OpenMP provides several constructs for sharing work among threads in a team. These are: Parallel for/do, Parallel Sections, Workshare, and Single directive. These constructs are placed inside an existing parallel region. The result is to distribute execution of associated statements among the existing threads. A number of environmental variables may be set to control aspects of OpenMP execution; for example, the number of threads, loop scheduling, and the enabling of nested parallelism and dynamic adjustment of the number of threads.

OpenMP also provides a number of routines that may be called from within one's code. These may be used to get and set the number of threads, enable or disable dynamic thread allocation, check whether the code is executing in parallel, etc. Changes in the runtime environment made by these routines take precedence over the corresponding environment variables.

4. NAS Parallel Benchmark. We used the NPB OpenMP-C benchmark suite, based on NPB 2.3-serial version, to evaluate the OpenMP performance on our multicore machines. Since the official OpenMP version of the NBP benchmarks from NASA is written only in FORTRAN we used the NPB OpenMP-C version that was developed as part of the Omni project [6]. The NAS Parallel Benchmarks (NPB) [4, 5] was devised by the Numerical Aerodynamic Simulation Program of NASA for the performance analysis of highly parallel computers. The NPB are valuable since they are rigorous and close to real-life needed applications. The NPB consist of five kernels and three simulated applications. They all compute or simulate different algorithmic and computational aspects of aerodynamic applications. For most of the kernels it is possible to select the problem size. Sometimes the problem sizes are called: Class S or T (12x12x12), Class W (24x24x24), and Class A 64x64x64).

The following is a brief description of the five kernels we used in our work.

Kernel EP. In the embarrassing parallel benchmark, two-dimensional statistics are accumulated from a large number of Gaussian pseudo-random numbers, which are generated according to a particular scheme that is well suited for parallel computation.

Kernel MG. The MG (Multi-grid) benchmark is a simplified multi-grid kernel, which solves a 3-D Poisson PDE. The Class W problem uses the same size grid as Class S but has a greater number of inner loop iterations.

Kernel CG. In the CG (Conjugate Gradient) benchmark, a conjugate gradient method is used to compute an approximation to the smallest eigen value of a large, sparse, symmetric positive definite matrix. This kernel is typical of unstructured grid computations applications.

TABLE 5.1
The Tested Multicore Machines

Platform	No. of Cores	Clock(GHz)	L1 Cache	L2 Cache	Memory
Intel Pentium D 820	2	2.8	2x16KB	2x1MB	512MB
Intel Core 2 Duo E6300	2	1.86	2x32KB	1x2MB	1GB
Intel Core 2 Quad Q6600	4	2.4	4x32KB	2x2MB	512MB

Kernel FT. In the FT (3-D FFT PDE) benchmark, a 3-D partial differential equation is solved using FFTs. This kernel performs the essence of many spectral methods. This benchmark is somewhat unique in that computational library routines may be legally employed.

Kernel IS. The IS (Integer Sort) benchmark tests a sorting operation that is important in particle method codes. This type of application is similar to particle-in-cell applications of physics, wherein particles are assigned to cells and may drift out. The sorting operation is used to reassign particles to the appropriate cells.

The three simulated applications we used are as follows.

BT is a simulated CFD application that uses an implicit algorithm to solve 3-dimensional (3-D) compressible Navier-Stokes equations. The finite differences solution to the problem is based on an Alternating Direction Implicit (ADI) approximate factorization that decouples the *x, y,* and *z* dimensions. The resulting systems are Block-Tridiagonal of 5x5 blocks and are solved sequentially along each dimension.

SP is a simulated CFD application that has a similar structure to BT. The finite differences solution to the problem is based on a Beam-Warming approximate factorization that decouples the *x, y,* and *z* dimensions. The resulting system has Scalar Pentadiagonal bands of linear equations that are solved sequentially along each dimension.

LU is a simulated CFD application that uses the symmetric successive over-relaxation (SSOR) method to solve a seven-block-diagonal system resulting from finite-difference discretization of the Navier-Stokes equations in 3-D by splitting it into block Lower and Upper triangular systems.

5. EPCC Microbenchmarks. The EPCC micro-benchmark suite is a set of benchmarks that measure the overhead incurred by OpenMP compiler directives of a specific OpenMP implementation [7, 12]. Three classes of overhead can be measured by the EPCC micro-benchmark suite: synchronization, loop scheduling, and array operations. The current release supports the OpenMP 2.0 standard. The overhead cost incurred by a specific compiler directive is measured by comparing the sequential execution time of a section code containing the compiler directive, and the parallel execution time of the same code. The measurements are repeated a few times for statistical stability. By using the EPCC micro-benchmark, the developer is able to compare the relative efficiency of different implementations of OpenMP running on the same platform; choose the more efficient construct of two semantically equivalent; and predict the overall performance of an application. Although there are other tools in the market that were developed for similar purposes as EPCC, such as ompP [10] that performs overhead analysis, and Sphinx [13], EPCC software is considered the de-facto standard of its kind.

6. Experimental Results. We tested the performance of the EPCC micro-benchmarks and the NAS OpenMP kernels and applications on multicore machines. The list of the tested platforms is shown in Table 1. On the software side we used the OpenMP version 2.0 of Intel C++ OpenMP compiler 11.0 under the XP operating system. All the benchmarks were also compiled by Microsoft Visual Studio C++ 2005 and evaluated on the above multicore machines. However, the differences between the results obtained by using the Microsoft and Intel compilers are negligible, so they are not shown here. Only the results obtained by Intel compiler are shown here. All the measurements shown are in units of Kilo-Clock-Cycles (KCC) and Giga-Clock-Cycles (GCC) for better comparison between different machine architectures. KCC is thousands of clock-cycles per second and GCC is billions of clock-cycles per second.

First, we measured the overhead cost of the OpenMP directives by using the EPCC micro-benchmarks. In general, it is an important practice to start parallel application benchmarks by examining the overhead incurred by the primitive parallelism functions used by the parallel programming model on the testbed machine. This way the programmer has a priori knowledge of the effects of various possible overhead sources on the total performance of the applications. Tables 2-6 and figures 1-5 show results measured on the Intel Pentium D, Intel

TABLE 6.1
Synchronization overheads(in KCC) of 1, 2 and 4 threads on Pentium D, Core 2 Duo and Core 2 Quad machines

	Pentium D			Core 2 Duo			Core 2 Quad		
Num of Threads	1	2	4	1	2	4	1	2	4
Parallel	2.8	37.3	234	1.3	16.1	65	1.2	46.6	45.2
for	0.2	14.5	292	0.1	6.2	65	0.1	11.1	12.2
Parallel-for	2.9	39.4	221	1.3	15.1	64	1.3	48	45.5
Barrier	0.2	14.0	288	0.01	6.2	63	0.02	10.9	11.7
Single	0.1	9.6	266	0.09	5.3	2.0	0.07	11.9	11.4
Critical	0.2	1.0	2.0	0.13	0.93	0.5	1.2	1.0	14.4
Lock-Unlock	0.2	1.4	2.5	0.16	0.93	0.8	1.3	1.8	10.6
Ordered	0.5	27.1	27.1	0.24	10.7	15.0	0.25	23.5	20.3
Atomic	0.1	0.3	0.4	0.07	0.22	0.20	0.07	0.3	1.0
Reduction	3.0	40.2	260	0.13	16.1	17.0	1.2	50	52

Core 2 Duo and Intel Core 2 Quad machines while running EPCC micro-benchmarks compiled by Intel C++ compiler 9.1.

Table 2 shows the OpenMP synchronization overheads measured by running the EPCC micro-benchmarks with 1, 2 and 4 threads. Figure 1 is a bar chart of the OpenMP synchronization overheads measured by running the EPCC micro-benchmarks with two threads. First, it can be observed from figure 1 that the overhead cost is less than 50K cycles in all the cases. This low overhead has a negligible effect on the NAS applications performance that will be discussed later. Second, the OpenMP directive overheads on the Intel Core 2 Duo are up to 50% less than the overheads incurred by the directives on the Intel Pentium D and the Intel Core 2 Quad. This improvement is mainly due to the shared L2 cache memory architecture of the Intel Core 2 Duo processor compared to the distributed L2 cache memory of the Intel Pentium D. The Intel Core 2 Quad processor contains two separate pairs of dual-core with shared L2 cache each. However, when two threads are spawned, the operating system maps each one of them to a different pair of dual-core. The physical separation of four cores to two pair of dual-core is the cause of the increase in the synchronization costs as shown in figure 1 and Table 2. On the other hand, when four threads are invoked on a quad-core processor, the operating system maps each one of the four threads to a different core and thus there is no competition between the threads on the available cores. The result is less overhead due to better scheduling as can be shown in Table 2. For example, the barrier synchronization overheads on the Intel Pentium D and the Intel Core 2 Duo (which are dual-core processors) with four threads are 288 KCC and 63 KCC respectively, while the barrier synchronization overhead on the Intel Core 2 Quad with four threads is only 11.7 KCC. We will elaborate on these architectures later. Third, the overhead of a single core is relatively high. Thus, it is better for single threaded applications to be compiled with the OpenMP option set to off.

Further analysis of the results leads to the following conclusions: the combined directive Parallel-For is more efficiently used than the Parallel and the For directives, which are used separately; it costs less to use the Critical directive than to use the Lock-Unlock pair directives; the Barrier and the Single directives have a relatively low overhead; the Order and the Reduction clauses have relatively high costs, as can be expected; and, finally, the overhead of the Atomic directive is negligible and thus is recommended for use, where it is possible, instead of the Critical or the Lock-Unlock directives. We omit the discussion on scalability with respect to the number of cores because it is useless to do such an analysis when the machines have only two and four cores.

Table 3 shows the Array (or privatization) overheads of four clauses: Private, Firstprivate, CopyPrivate, and Copyin for 1, 2 and 4 threads on the tested machines and figure 2 is a bar chart of the results for the case of two threads. First, it can be observed again that the Intel Core 2 Duo processor presents lower overheads than the Pentium D and the Core 2 Quad processors when using two threads. However, when using four threads the Intel Core 2 Quad processor is more efficient. For example, to access a private array on the Intel Pentium D and the Intel Core 2 Duo with four threads costs 160 KCC and 98 KCC respectively, while it costs only 36.2 KCC on the Intel Core 2 Quad with four threads. Moreover, the results show that the Private, Firstprivate, and Copyin clauses are incurred acceptable overhead costs for the array allocation process. The CopyPrivate demonstrates excellent performance and a negligible overhead cost that enables super-efficient inter-threads communication.

TABLE 6.2
Array privatization overheads(in KCC) of 1, 2 and 4 threads on Pentium D, Core 2 Duo and Core 2 Quad machines.

	Pentium D			Core 2 Duo			Core 2 Quad		
Num of Threads	1	2	4	1	2	4	1	2	4
Private	2.85	37.9	160	2.0	22.8	98	1.4	52	36.2
FirstPrivate	2.8	44.4	274	2.0	22.4	100	1.4	53.9	35.7
CopyPrivate	0.2	0.28	0.28	0.05	0.08	0.05	0.05	0.08	0.05
CopyIn	2.8	51.1	211	2.0	24.0	102	1.5	52.9	100

TABLE 6.3
OpenMP Loop Scheduling overheads (in KCC) of 1, 2 and 4 threads on Intel Pentium D machine.

	Threads	Pentium D							
Chunk size		1	2	4	8	16	32	64	128
Static	1	0.8	0.2	0.05	0.2	0.03	0.06	0.05	0.05
	2	16.8	15.3	18.6	20.5	15.6	21	15.2	15
	4	294	290	296	294	292	292	292	292
Dynamic	1	25	12	8.6	6.3	3.7	3.1	2.6	1.8
	2	92	50	40	30	25	23	20	19
	4	425	352	327	308	300	294	288	282
Guided	1	0.22	2.5	2.4	2.2	2.3	1.8	2.3	2.4
	2	27	25	24	23	22	23	19	18
	4	305	302	300	300	297	296	295	294

TABLE 6.4
OpenMP Loop Scheduling overheads (in KCC) of 1, 2 and 4 threads on Intel Core 2 Duo machine.

	Threads	Core 2 Duo							
Chunk size		1	2	4	8	16	32	64	128
Static	1	0.07	1.54	1.8	1.9	2.0	2.5	3.0	4.0
	2	7.4	6.2	6.2	6.4	7.5	7.5	6.0	6.2
	4	74	74	74	74	74	74	74	74
Dynamic	1	16.0	8.0	4.0	2.0	1.2	2.7	0.65	0.55
	2	39	21	15	12	10	8.7	8.2	8.7
	4	138	104	87	81	79	74	72	72
Guided	1	0.5	0.5	0.5	1.3	0.5	0.5	0.5	0.94
	2	10	8.7	9.6	9.3	8.2	7.5	7.5	7.5
	4	80	80	80	79	80	80	80	80

TABLE 6.5
OpenMP Loop Scheduling overheads (in KCC) of 1, 2 and 4 threads on Intel Core 2 Quad machine.

	Threads	Core 2 Quad							
Chunk size		1	2	4	8	16	32	64	128
Static	1	0.2	0.3	0.4	0.5	0.5	0.6	0.6	2.4
	2	10.3	10.0	9.8	7.8	7.0	10.1	9.9	9.8
	4	12.8	11.6	11.7	11.0	10.8	11.0	11.0	26.4
Dynamic	1	15.7	9.5	6.3	4.7	1.4	3.4	3.2	0.6
	2	89	25	13.3	17.4	15.8	10.5	11.4	13.6
	4	391	182	88	28.4	20.2	18.5	17.1	16.6
Guided	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	2	16.4	10.5	12.5	14.3	15.7	8.8	13.5	12.0
	4	24.6	23.0	21.5	19.3	18.4	16.6	15.8	15.2

TABLE 6.6
Performance (in GCC) of EP, FT, MG, CG, BT, SP, and LU benchmarks for 1, 2 and 4 threads, and problem class W on Pentium D, Core 2 Duo and Core 2 Quad machines.

	Pentium D						
Threads	EP	FT	MG	CG	BT	SP	LU
1	33.68	2.57	2.54	2.35	33.76	99.96	66.05
2	16.66	2.52	1.7	2.18	29.7	146.46	37.12
4	46.64	2.04	2.85	3.19	37.91	1304	3967
	Core 2 Duo						
1	15.14	1.24	1.48	1.45	14.32	47.69	35.4
2	7.75	1.19	0.98	1.39	13.24	64.43	20.68
4	7.7	0.98	1.17	1.56	14.48	292	1266
	Core 2 Quad						
1	15.12	1.22	1.41	1.46	22.03	47.54	33.93
2	7.53	1.15	0.79	1.0	19.96	56.61	19.89
4	3.96	0.81	0.64	0.6	19.96	76.34	12.14

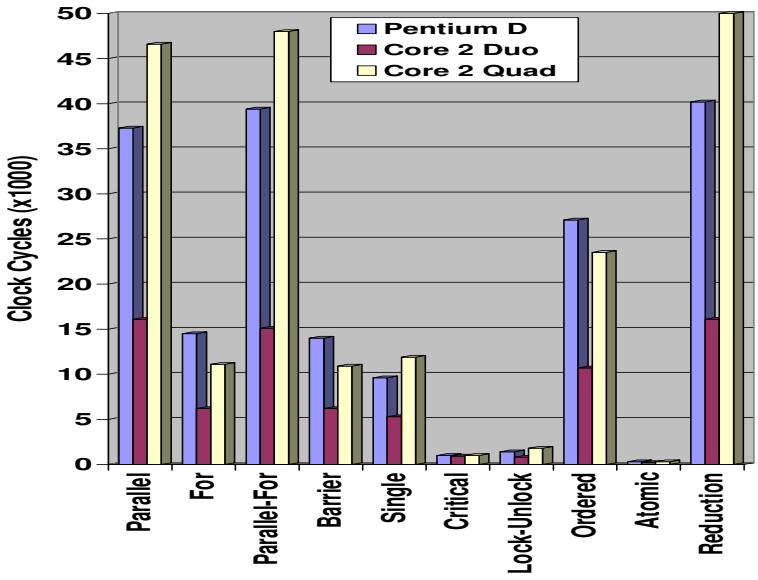


FIG. 6.1. *OpenMP Synchronization overheads of two threads on Intel Pentium D, Intel Core 2 Duo and Intel Core 2 Quad machines.*

Tables 4, 5 and 6 show the OpenMP loop scheduling overheads of the Static, Dynamic, and Guided clauses for chunk sizes of 1 to 128 when running 1, 2 and 4 threads, for Intel Pentium D, Intel Core 2 Duo and Intel Core 2 Quad machines respectively. Figures 3-5 are bar charts of the results for the case of two threads.

It can be observed that each clause has a different pattern. In the case of Intel Pentium D and two threads (Figure 3), the block cyclic scheduling (Static) presents similar overhead for all chunk sizes ($\sim$ 15K cycles). The Dynamic scheduling overhead decreases rapidly from the maximum point at a chunk size of one (92K cycles) to the minimum point at a chunk size of 128 (19K cycles). The Guided scheduling overhead has a similar pattern but with a more moderate decreasing curve. The maximum point is at a chunk size of one (92K cycles) and the minimum point is at a chunk size of 128 (14K cycles). The conclusion is that by increasing the chunk size the loop scheduling overhead is minimized.

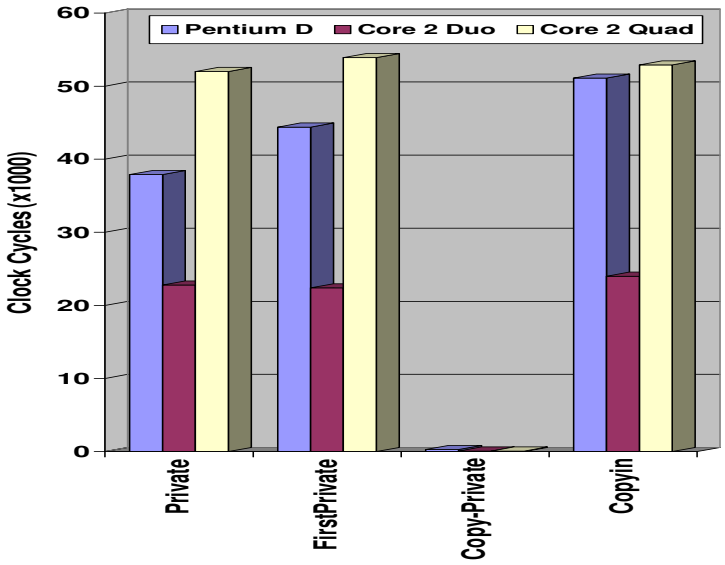


FIG. 6.2. *OpenMP Array Scheduling (privatization) overheads of two threads on Intel Pentium D, Intel Core 2 Duo and Intel Core 2 Quad machines.*

In the case of Intel Core 2 Duo and two threads (Figure 4), the loop scheduling overheads present similar pattern compared to the Intel Pentium D, but the overheads are up to 50% lower than those of the Intel Pentium D. The block cyclic scheduling (Static) is reach its optimal overhead at a chunk size of 64 (6K cycles). The Dynamic scheduling overhead decreases rapidly from the maximum point at a chunk size of one (48K cycles) to the minimum point at a chunk size of 64 (8K cycles). The Guided scheduling overhead has a similar pattern but with a more moderate decreasing curve. The maximum point is at a chunk size of one (10K cycles) and the minimum point is at a chunk size of 128(7K cycles).

In the case of the Intel Core 2 Quad and two threads (Figure 5), the loop scheduling overheads present similar pattern compared to the Intel Pentium D, but the overheads are more fluctuated. The block cyclic scheduling (Static) is reach its optimal overhead at a chunk size of 16 (7K cycles). The Dynamic scheduling overhead decreases rapidly from the maximum point at a chunk size of one (88K cycles) to the minimum point at a chunk size of 16 (10K cycles). The Guided scheduling overhead has a similar pattern but with a more moderate decreasing curve. The maximum point is at a chunk size of one (17K cycles) and the minimum point is at a chunk size of 32(8K cycles). It can be observed again that in the case of the Intel Core 2 Quad and four threads (Table 6), the scheduling overhead is up to twice compared to the case of two threads but still reasonably, in contrast to the scheduling overhead associate with four threads on the Intel Pentium D and the Intel 2 Core Duo where the overhead costs, due to less cores, are unacceptable (Table 4 and 5).

The bottom line of the EPCC micro-benchmarks results is that the overhead incurred by the OpenMP directives and the clauses are low and will not harm the performance of the NBP application benchmarks.

The NPB benchmarks were conducted with three different input sizes: S, W, and A for 1, 2 and 4 threads. The total running time of each benchmark was measured by wall-clock time. The speedup, efficiency, and the overhead of each run were calculated as follows:

Speedup $= T_1/T_{p,k}$ where T_1 is the time measured for running with a single core and $T_{p,k}$ the time measured with p cores and k threads.

Efficiency $= T_1/(p \cdot T_p)$ where T_1 is the time measured for running with a single core and T_p the time measured with p cores.

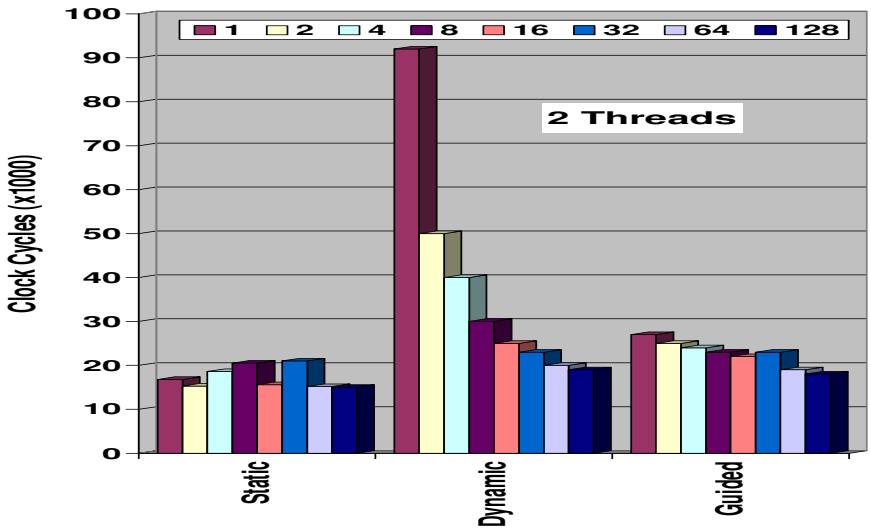


FIG. 6.3. *OpenMP Loop Scheduling overheads of two threads on Intel Pentium D machine.*

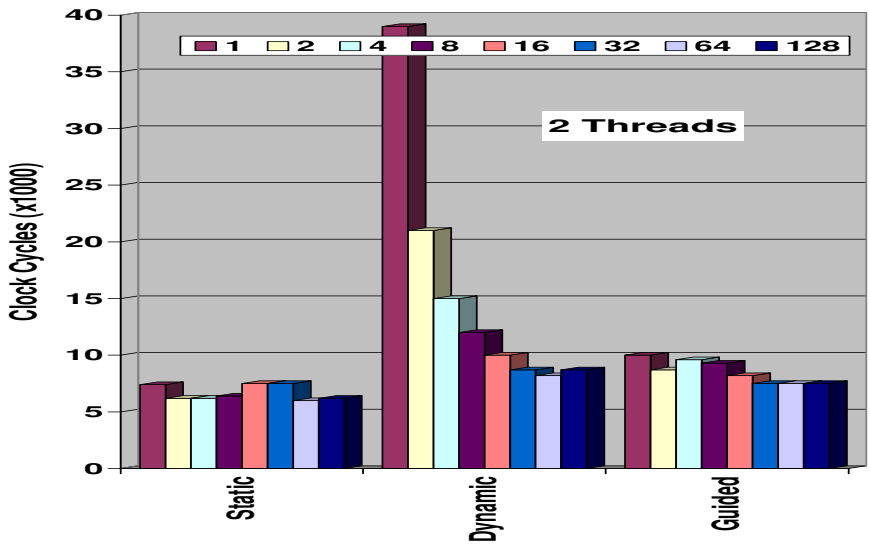


FIG. 6.4. *OpenMP Loop Scheduling overheads of two threads on Intel Core 2 Duo machine.*

$Overhead = T_{p,k} - T_1/p$ where T_1 is the time measured for running with a single core and $T_{p,k}$ the time measured with p cores and k threads.

We ran the original benchmarks that appear in [6] without any modification. Table 7 presents the performance (in Giga clock cycles) of class W (24x24x24) of seven different benchmarks in the NAS OpenMP suite for 1, 2 and 4 threads on the tested machines. Figure 6 is a bar chart of the calculated speedup of the NPB benchmarks, for the case of two threads, on the Intel Pentium D, the Intel Core 2 Duo and the Intel Core 2

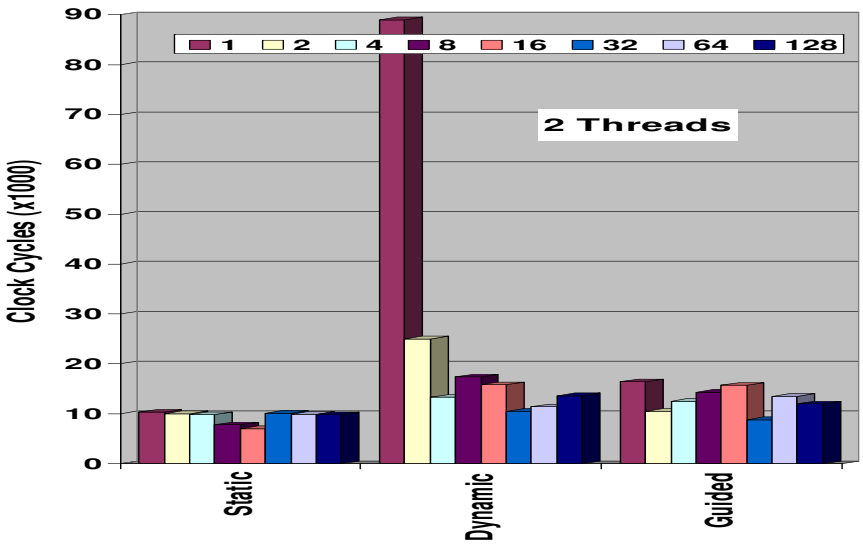


FIG. 6.5. *OpenMP Loop Scheduling overheads of two threads on Intel Core 2 Quad machine.*

Quad machines. However, the behavior of the benchmarks results for input classes S and A are similar to the input class W. The results of the IS kernel are omitted due to a problem known to the developers of the benchmarks suite , which they will fix in the next release. Figure 7 is a bar graph that depicts the percentage of the computation time and the overhead time as part of the total execution time, in the case of the Intel Pentium D machine.

Analysis of these results leads to the following findings.

The EP Kernel falls into the category of applications termed "embarrassingly parallel" based on the trivial partition ability of the problem, while incurring no data or functional dependencies, and requiring little or no communication between processors. It is included in the NPB suite to establish the reference point for peak performance on a given platform. Therefore, it is not surprising that the EP kernel achieved perfect speedup (~ 2.0) for two threads.

As can be observed from Figure 6, the speedups of the Intel Pentium D and the Intel Core 2 Duo processors are similar, with slight improvement in the case of Intel Core 2 Quad processor. The LU decomposition application, in case of Intel Pentium D, shows good speedup and efficiency for two cores, 1.77 and 0.88 respectively, but in the case of four threads the speedup drops drastically to 0.01 due to fewer cores than threads. The Intel Core 2 Duo processor achieves speedup of 1.71 and efficiency of 0.85 for two threads but drops to speedup of 0.02 for the case of four threads. On the other hand, the Intel Core 2 Quad achieves speedup 1.7 (0.85 efficiency) and 2.73 (0.68 efficiency) for two and four threads respectively (Table 7). The MG kernel shows more modest speedups (1.49 and 1.51) and efficiency (0.74 and 0.75) for two threads while for four threads the speedups drop to 0.89 and 0.79 on the Intel Pentium D and the Intel Core 2 Duo respectively. In the case of Intel Core 2 Quad the speedups are 1.78 (0.89 efficiency) and 2.20 (0.55 efficiency) for two and four threads respectively.

The rest of the benchmarks (FT, CG, BT, and SP) show poor speedups and efficiencies. These results can be explained by the logical structure of the applications, which do not match the underlying architectures of the multicore processors. For example, the FT kernel uses FFT on a complex array to solve a three-dimensional partial differential equation. Communication patterns in this kernel are structured and long distance in nature. This benchmark represents the essence of many "spectral" codes or eddy turbulence simulations. The CG kernel is used in conjugate gradient methods to approximate the smallest eigen-value of a symmetric, positive definite, sparse matrix with a random pattern of non-zeros. The communication patterns in this kernel are long- distance and unstructured.

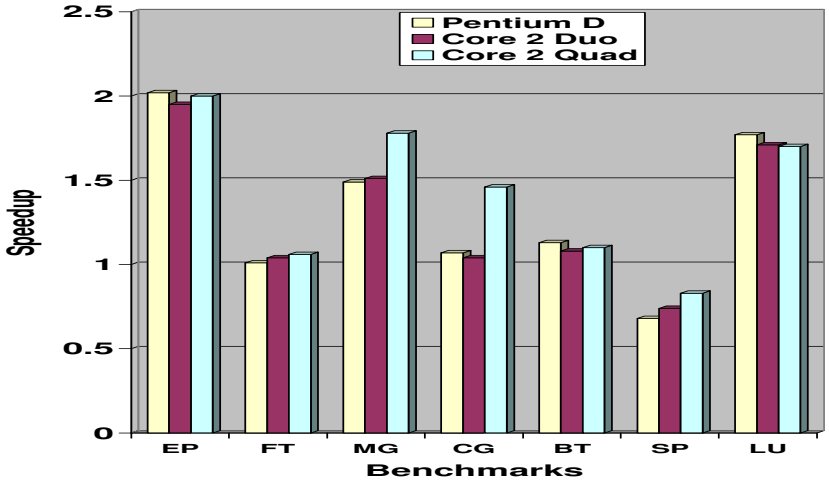


FIG. 6.6. *NPB speedup results of EP, FT, MG, CG, BT, SP, and LU benchmarks for 2 threads; problem class W; on Pentium D, Core 2 Duo and Core 2 Quad machines.*

These observations reveal the following conclusions. First, all the benchmarks, except EP, achieve very poor efficiency when the number of threads (4) is greater than the number of cores (2) which occurs in the cases of Intel Pentium D and Intel Core 2 Duo processors. In the case of Intel Core 2 Quad processor and four threads there is improvement in the speedups when changing the number of threads from two to four threads, except to the SP benchmark that exhibits poor performance in all the cases. It happens because the overhead caused by context-switch operations of the competing threads on the CPU resources is high. Moreover, two threads sharing a single core lead to cache conflicts that decrease the hit rate and thus degrade the performance.

The above poor performance of the NPB applications brought us, on the one hand, to extend our study further and to look for possible solutions to improve the performance but without restructuring the application programs, and on the other hand, to extend our understanding of how the underlying hardware works.

The Intel Pentium D processor has a different cache-memory architecture than Intel Core 2 Duo and Intel Core 2 Quad processors [8, 9]. The Pentium D 820 processor is a "distributed" cache-memory with two separately L1 caches of 16KB each and two separately L2 caches of 1MB each. On the other side, the Intel Core 2 Duo E6300 is a "shared" cache-memory with two separate L1 caches of 32KB each and a shared L2 cache of 4MB and the Intel Core 2 Quad Q6600 has two separate pairs of dual-core with 4MB shared L2 cache, and 2x32KB L1 caches. To understand the implications of these three different architectures on the performance, lets look at the following example.

Let A[100] be a shared array used by two threads running on two different cores. The threads are writing the array at the same time. One thread accesses the first part of the array, A[0-49], and the second one accesses the second half of the array, A[50-99]. In the case of the Intel Pentium D, array A will be copied into the L1 and the L2 caches of each core. Now, each time one of the threads completes a write operation, there is a need to update the copy of the array A in the neighboring core in order to maintain the caches consistency. This update is costly in terms of CPU cycles, as can be seen in Table 8. However, we expected to an improvement in the case of Intel Core 2 Duo processor because the L2 cache is shared, but we were disappointed to discover that the speedups of NBP benchmarks showed only $\sim 4\%$ improvment (in the case of FT, MG and SP) compared to the Intel Pentium D, and $\sim 4\%$ worsening (in the case of CG, BT and LU) when running with two threads.

So, we continued to explore further and we found another obstacle that we were not aware of: the shared L2 cache of the Intel Core 2 Duo is not banked. The L2 cache serves only one core at any given clock cycle, so a banked organization will not help. A round robin scheme is used to allocate L2 cache services to the cores for

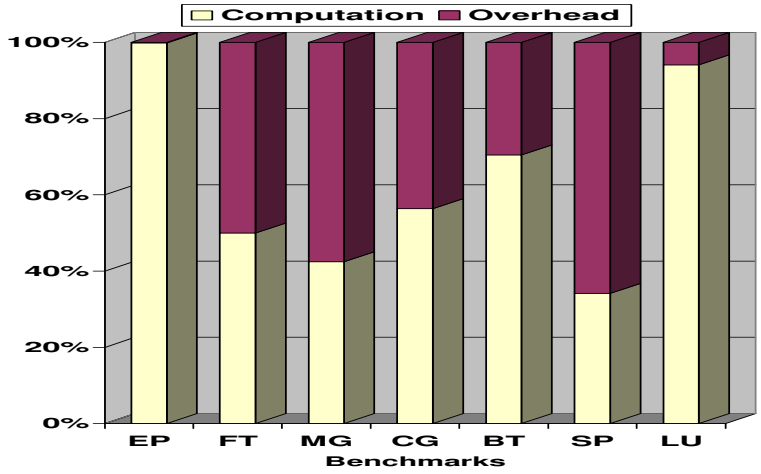


FIG. 6.7. *NPB computation time vs. overhead time of EP, FT, MG, CG, BT, SP, and LU benchmarks for 2 threads and class W on Pentium D machine.*

TABLE 6.7
False sharing penalties.

Case	Data Location	Latency(cycles/nsec)
L1 to L1 Cache	L1	14 core + 5.5 bus
Through L2	L2	14 core
Through Memory	Main Memory	14 core + 5.5 bus + 40-80 nsec

scenarios when both cores request L2 service. The false sharing penalty of the Intel Core 2 Duo is depicted in Table 8 and was taken from [9].

We looked further for optimization possibilities for improving the performance of the applications but without need to rewritten the programs.

First, we used a thread affinity option to tie a thread to its data to improve data locality [11]. Since OpenMP does not support thread affinity capabilities we used the Windows operating systems *SetThreadAffinityMask* option. Monitoring the threads scheduling by the Intel VTune performance analyzer confirmed that each thread was tied to one core during the program execution. Unfortunately, we did not observe any improvement in the performance of the applications. Second, we changed the loop iterations scheduling by using the OpenMP schedule clause. The fact that most of the parallelism of the NPB applications is done by *for* work-sharing, encouraged us to find the optimal scheduling. We tried the Static, Dynamic, and Guided options with 1, 4, 8, 16, 32, 64, and 128 chunk sizes. Unfortunately, we cannot report any significant improvement in the applications performance.

To summarize this section we list the key observations of our study.

- The impact of the parallel overheads of the OpenMP parallel mechanisms on the performance of NPB benchmarks is negligible.
- The new Intel multicore processors reduce the parallel overheads incurred by the OpenMP mechanisms compared to their predecessors.
- It is recommended to turn off the OpenMP compiler option when running single threaded applications because the relatively high overhead incurred by OpenMP.

- It is recommended to use one thread per processor-core to prevent the competition of two threads on a single core.
- Using the OpenMP directives Parallel and For together rather than separately incur less overhead.
- It is recommended to use the low-overhead OpenMP Atomic directive, where it is possible, instead of other locking mechanisms.
- OpenMP Loop scheduling incurs less overhead when the chunk size increases.
- The poor performance of five NPB benchmarks (FT, MG, CG, BT, and SP) is due to lack of matching between the logical structures of the applications and the underlying architecture of the multicore processors.

7. Conclusions. Multicore processors will dominate scientific computing, and commercial computing as well, in the near future. Understanding their performance characteristics is essential for design scalable and efficient applications. In this paper, we presented the efficiency of applications from NPB OpenMP-C suite and the overhead measurements of OpenMP directives and clauses running on Intel Pentium D, Intel Core 2 Duo and Intel Core 2 Quad machines using MS Visual studio C++ 2005 and Intel C++ compilers.

The benchmarking results show that most of the applications achieved poor performance, not because of the overhead incurred by the OpenMP directives, but because the NPB applications induced computation and communication patterns which are not cache friendly and result in a lot of false sharing situations. The diversified cache architectures of multicore processors call for new parallel programming languages and compilers that can use the hierarchy of cache memory systems in an efficient manner.

REFERENCES

[1] Geer D. (2005), "Chip makers turn to multicore processors," IEEE Computer.
[2] Marowka A. (2007), "Parallel Computing on Any Desktop," Communication of ACM, Vol. 50, Issue 9, pp. 74-78.
[3] "OpenMP Application Program Interface," http://www.openmp.org.
[4] Bailey D. H., Harsis T., Saphir W., Wijngaart R. V., Woo A., and Yarrow M., (1995) "The NAS Parallel Benchmarks 2.0," Report NAS-95-020, Nasa Ames Research Center.
[5] Jin H., Frumkin M., and Yan J., (1999) "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance," Report NAS-99-011, Nasa Ames Research Center.
[6] "The Omni Project," http://phase.hpcc.jp/Omni/home.html
[7] Bull M. (1999), "Measuring Synchronisation and Scheduling Overheads in OpenMP," Proceeding of First European Workshop on OpenMP (EWOMP '99) Lund, Sweden.
[8] Doweck J., (2006) "Inside Intel® Core™ Micro architecture and Smart Memory Access," A White Paper, Intel.
[9] Mendelson A., Mandelblat J., Gochman S., Shemer A., Chabukswar R., Niemeyer E. and Kumar A., (2006) "ICMP Implementation in Systems Based on the Intel® Core™ Duo Processor," Intel® Technology Journal, Vol. 10, Issue 02.
[10] Furlinger K., Gerndt M., and Dongarra J., (2007) "Scalability Analysis of the SPEC OpenMP Benchmarks on Large-Scale Shared Memory Multiprocessors," Proceeding of ICCS.
[11] Tian T., (2007) "Tips for effective usage of the shared cache in multicore architectures," Embedded magazine, http://embedded.com/showArticle.jhtml?articleID=196902691
[12] Bull M. and O'Neill D., (2001) "Microbenchmark Suite for OpenMP 2.0," Proceedings of the Third European Workshop on OpenMP (EWOMP'01), Pages: 41–48, Barcelona, Spain.
[13] "Sphinx Micro-benchmark Suite," http://www.llnl.gov/CASC/RTS_Report/sphinx.html
[14] Marowka A., (2008) "Performance of OpenMP Benchmarks on Multicore Processors," 8th International Conference on Algorithms and Architectures for Parallel Processing(ICA3PP), Agia Napa, Cyprus, LNCS proceeding Vol. 5022 pp. 208-219.
[15] Liao C., Liu Z., Huang L., and Chapman B. (2005) "Evaluating OpenMP on Chip MultiThreading Platforms," Proceeding of first international workshop on OpenMP (IWOMP 2005), Eugene, Oregon USA.
[16] Milfeld K., Goto K., Purkayastha A., Guiang C. and Schulz K. (2007) "Effective Use of Multi-Core Commodity Systems," The 8th LCI International Conference on High-Performance Clustered Computing, Lake Tahoe, California, 2007.
[17] Furlinger K., Gerndt M. and Jack Dongarra J. (2007) "Scalability Analysis of the SPEC OpenMP Benchmarks on Large-Scale Shared Memory Multiprocessors," Proceeding of ICCS, LNCS Volume 4488, pp. 815–822.
[18] Grant E. R. and Afsahi A. (2007) "A Comprehensive Analysis of OpenMP Applications on Dual-Core Intel Xeon SMPs," IEEE Proceeding of Parallel and Distributed Processing Symposium (IPDPS 2007).
[19] Curtis-Maury M., Ding X., Antonopoulos C. D. and Nikolopoulos D. S. (2008) "An Evaluation of OpenMP on Current and Emerging Multithreaded/Multicore Processors," M. S. Mueller et al. (Eds.): IWOMP 2005/2006, LNCS 4315, pp. 133–144, 2008.

Edited by: Marcin Paprzycki
Received: May 25th, 2009
Accepted: November 25th, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 443–444. http://www.scpe.org
ISSN 1895-1767

BOOK REVIEWS

EDITED BY SHAHRAM RAHIMI

Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies
by D. Floreano and C. Mattiussi

This is a book that bridges biological systems and computer science. For digital-based researchers, having this book which details the biological components of natural life and seamlessly integrates that knowledge into our digital realm is an essential asset. Each chapter is systematically introduces the reader to a biological system while easing them into the its computational counterpart. There are seven chapters covering evolution, cellular, neural, developmental, immune, behavioral, and collective systems.

Chapter 1 introduces the fundamental concept of computational evolution as related to biological systems. This chapter starts with the basic concepts of evolutionary theory and progresses, covering everything from fitness functions to analog circuits. The following chapter presents the next logical step upwards in biology, cellular structures and systems. Again introducing the basics of life and progressing towards cellular automata. Chapter 3 covers Neural Networks by introducing the Biological Nervous System, then the Artificial Neural Network. The core concepts to Neural Networks are detailed in a systematic and common-sense manner, introducing unsupervised learning, supervised learning, and reinforce learning, then progressing onto neural hardware and hybrid systems. In Chapter 4, the authors detail developmental systems, explaining how nature utilizes the cellular structures to how engineers can mimic nature. This theme of progression from biological introduction to digital computation is reproduced as a single voice through out each chapter. The fundamentals of Bio-Inspired Artificial Intelligence are well demonstrated, allowing for a novice researcher in this area to develop the necessary skills and have a firm grasp on this topic.

Once the reader has a solid grasp of the building blocks of life, the authors present chapters related to larger systems. Of particular interest to my research is the chapter on Immune Systems. This chapter provides a fundamental understanding of the Human Immune System, detailing the finer points of immunological cellular structures, while introducing a slightly more than generalized immune response concept. After a lengthy introduction of human immunology, we are introduced to the core of Artificial Immune Systems, the Negative Selection Algorithm and Clonal Selection Algorithm. Each one of these algorithms is covered enough so that the reader is capable of understanding each respective algorithms strengths and limitations. For new researchers to Artificial Immune Systems, days of reading journal articles is summarized in these sections, allowing for intelligent and efficient decision making in choosing your next step of research.

Chapter 6 and 7 provides the audience with behavior systems and collective systems, respectively. The behavioral systems covered in this book relate to aspects of AI, robots, and some machine learning. Once behavior is understood, collective and cooperative systems are covered. Optimization techniques of particle swarms, ant colonies, and topics derived for robotics are detailed and well explained.

While this is not a textbook, is does cover the fundamental concepts required to research Bio-Inspired Artificial Intelligence. For myself, the quality of this book can simply be noted by the publishers, MIT Press. Many of the best books I have encountered in my studies have been published by MIT, and here is another. Floreano and Mattiussi have not let me down in their quality, albeit I do have some complaints.

First, while the topics cover a solid breadth, the depth on detailing the computation side is limited. I would like to have seen either more depth in each chapter or a broader look at each chapters algorithms, but the book falls somewhere in the middle. My current research involves Danger Signals and their relationship to preventing Epidemic Attacks, so I would have like to seen more detail about Polly Matzinger's Danger Theory rather than one short paragraph saying that it is not universally accepted. While Immunologists may debate Danger Theory, novel algorithms have been developed off of the concept of Danger Theory and deserve a place in this book. Yet to counter my own argument, the authors do finish off each chapter with a Suggested Readings section outlining a series of excellent supplement papers to the chapters topics that would eventually lead the reader to these novel topics.

Overall, if you are interested in this field, buy this book. You can find it online at MIT Press for a discounted price. This book will make an excellent addition to any computer researchers library.

Anthony Kulis,
Department of Computer Science,
Southern Illinois University

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2$_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.