Numerical Integration of the Differential Riccati Equation: A High Performance Computing Approach

Enrique Arias[1] and Vicente Hernández[2]

[1] Departamento de Informática,
Universidad Castilla- La Mancha,
Avda. España s/n,02071-Albacete, Spain
earias@info-ab.uclm.es
Telephone number: +34-967-599200 Ext: 2597
Fax number: +34-967-599224

[2] Departamento de Sistemas Informátios y Computación,
Universidad Politécnica de Valencia,
Camino de Vera s/n, 46022-Valencia, Spain
vhernand@dsic.upv.es
Telephone number: +34-96-3877350 Ext: 73560
Fax number: +34-96-3877359

Abstract. This paper presents a **High Performance Computing Approach (HPC)** to the code appeared in [1] called DRESOL, for the **Numerical Integration of the Differential Riccati Equation**. This equation arises in the application of quadratic optimization for motion control to the feedback control of robotic manipulators. In this paper the main changes carried out in the DRESOL package and the new block oriented subroutines for computing the Sylvester and Lyapunov equations in order to obtain a sequential HPC implementation are described. From this new sequential implementation parallel algorithms for distributed memory platforms have been also carried out.
Keywords: Numerical Methods, Parallel Computing.

1 Introduction

During the last years, the availability of more powerful computers at a low cost has allowed great advances in the theory and application of non linear control [2–5].

This paper is concerned with the application of parallel computing for solving the differential Riccati equation (DRE). This equation arises on several applications of different fields of science and engineering, such as the quadratic optimization for motion control to the feedback control of robotic manipulators.

It is possible to find in the literature different methods for solving the DRE [6, 7]: direct integration methods [8], methods based on Bernoulli substitutions [9, 10], methods based on superposition rule of Riccati solutions [11, 12], numerical integration of the Riccati equation based on a Newton iteration method [13, 1], symplectic methods [15, 16], methods based on Magnus series [17], the Mobius scheme [18], etc.

The work presented in this paper has been based on the implementation developed by Luca Dieci [1] for solving the DRE by means of numerical integration.

This paper is organized as follows. First, an exhaustive study of the DRESOL package has been carried out in order to identify the main computational kernels to be enhanced. Section 2 describes the DRESOL package and several changes are proposed from a High Performance Computing (HPC) point of view. These changes can be clasified into minor changes and important changes. Minor changes are related to intensive use of standard linear algebra libreries such as Basic Linear Algebra Subroutines (BLAS) [20] and Linear Algebra PACKage (LAPACK) [21]. Important changes consist in new block oriented algorithms. In particular, new routines to solve the Sylvester and Lyapunov equations have been carried out under a block-oriented philosophy in order to improve performance.

Once a better sequential implementation was available, a distributed implementation has been carried out using Parallel Basic Linear Algebra Subroutines (PBLAS) [22] and Scalable Linear Algebra PACKage (ScaLAPACK) [23] for computations and Message Passing Interface (MPI) [24] for comunications and synchronizations purposes. Details on the parallel implementation are introduced in Section 4. Section 5 presents the cases of study and some experimental results. Finally, some conclusions and future work are outlined in Section 6.

2 Numerical Integration of the Differential Riccati Equation

In this Section, the methodology introduced in [1] by Luca Dieci for solving the Differential Riccati Equation (DRE) by numerical integration is presented.

DRE means the expression of a time-dependent change of variables which decouples a linear system of Ordinary Differential Equations (ODEs). Then, let $A(t)$ be the matrix of the underlying linear system. This matrix is a continuous matrix function and all the quantities are time dependent. Let's consider the two-point boundary value problems (TPBVPs) with separated boundary conditions (BCs), where the partitioning conforms to the BCs, and the matrix B_{12} is invertible

$$\dot{y} = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \ 0 < t < T,$$

$$[B_{11}, B_{12}] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_{t=0} = \beta_1, \ [B_{21}, B_{22}] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_{t=T} = \beta_2,$$

where $A_{11} \in R^{n\times n},\ A_{22} \in R^{m\times m},\ y_1 \in R^n,\ y_2 \in R^m$, and the constant matrices $B_{12} \in R^{m\times m},\ B_{21} \in R^{n\times n},\ B_1 \in R^m, B_2 \in R^n$. It is possible to consider the time-dependent change of variable

$$S(t) = \begin{pmatrix} I_n & 0 \\ X(t) & I_n \end{pmatrix}, \quad X(t) \in R^{n\times m}.$$

Assuming a new variable $w = S^{-1}(t)y$, the system is partly decoupled resulting the following structure

$$\dot{w} = \hat{A}(t)w, \tag{1}$$

$$\hat{A}(t) = S^{-1}AS - S^{-1}\dot{S} = \begin{pmatrix} A_{11} + A_{12}X & A_{12} \\ 0 & A_{22} - XA_{12} \end{pmatrix} = \begin{pmatrix} \hat{A}_{11} & A_{12} \\ 0 & \hat{A}_{22} \end{pmatrix}, \tag{2}$$

plus BCs for w. This problem has solution if and only if $X(t)$ satisfies the DRE

$$\dot{X} = A_{21} + A_{22}X - XA_{11} - XA_{12}X = F(t, X),\ \ X(0) = X_0, \tag{3}$$

and initial values X_0. By requiring

$$X_0 = -B_{12}^{-1}B_{11}, \tag{4}$$

then the BCs in (1) are also decoupled

$$(0, B_{12})w(0) = \beta_1, \quad (B_{21} + B_{22}X(T), B_{22})w(T) = \beta_2. \tag{5}$$

So, the original BVP (Boundary Value Problem) can be solved according to the following strategy:

1. Solve equations (3,4) and the $w_2 - \mathit{equation}$ in (2,5) from 0 to T.
2. Solve $w_1 - \mathit{equation}$ in (2,5) from T to 0 and recover $y = S(t)w$.

All the possible strategies need to integrate (3), and the numerical integration of this equation is the main purpose of this section, presented in [1].

A particular case of the Riccati equation (3), is the *symmetric* DRE

$$\dot{X} = A_{21} - A_{11}^T X - XA_{11} - XA_{12}X, \quad X(0) = X_0. \tag{6}$$

This kind of equation (6) appears when the matrix $A(t)$ is the particular Hamiltonian

$$A(t) = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{11}^T \end{pmatrix}$$

being $n = m,\ A_{12} = A_{21}^T,\ A_{21} = A_{21}^T$ semidefinite positive, and the solution is $X = X^T$. This *symmetric* DRE can be found in several application such as optimal control.

A common way to deal with the numerical integration of DRE is to rewrite it as an nm-vector differential equation, and then use any available initial value problem (IVP) software.

In [1] implicit schemes are used to solve the discrete version of the DRE according to the following property:

Property Discretization by an implicit scheme of a polynomial differential equation reduces it to a polynomial algebraic equation of the same degree.

In particular, the DRE reduces to an Algebraic Riccati Equation (ARE) after discretization. Then, this ARE is solved at each integration step by means of Newton's type iteration.

The implementation appeared in [1] is referred as DRESOL package. This implementation is based on the IVP software package LSODE developed by Hindsmarsh [14].

Several methods have been implemented for solving the AREs. However, in the context of stiff DREs, the best choice for solving the associated ARE obtained by using implicit schemes are Newton's or quasi-Newton's methods.

In both cases, at each iteration step a Sylvester equation has to be solved

$$G_{11}Y - YG_{22} = H, \tag{7}$$

where Y and $H \in R^{n\times m}$, $G_{11} \in R^{n\times n}$, and $G_{22} \in R^{m\times m}$. H, G_{11} y G_{22} change at each iteration if Newton's method is used, while G_{11} and G_{22} stay constant if quasi-Newton's method is used.

A standard solution for solving the equation (7) is the Bartels-Stewart algorithm [19]. By using this approach, the matrices G_{11} and G_{22} are both reduced, via orthogonal U and V, to Schur form, obtaining the equivalent system

$$(U^T G_{11} U)(U^T Y V) - (U^T Y V)(V^T G_{22} V) = U^T H V, \tag{8}$$

Once the quasi-triangular system (8) is solved for $U^T Y V$, then Y is easily recovered.

3 DRESOL package: A High Performance Computing version

In this Section, the changes carried out in the DRESOL package are described. The aim of these changes is to adapt, from a implementation point of view, the original DRESOL package to the current architectures, taking into account the memory hierarchy. Also, the changes introduced keep the following two important features:

- **Portability:** Standard Linear Algebra libraries have been used to achieve portable algorithms.
- **Efficiency:** Good performance has been achieved by improving certain routines. Enhancements in the routines consist on developing block oriented algorithms. In this way, the memory hierarchies of the modern computers can be efficiently exploited.

The DRESOL Package consists on the following subroutines and functions:

- **dresol and drequs**: They both constitute merely an interface between the user's driver and the differential Riccati equation solver, subroutine *drequ*.
- **intdy**: It computes an interpolated value of the y vector (which is the unrolled Riccati matrix x) at $t = tout$.

- **stode**: This is the core integrator, which does one step of the integration and the associated error control and eigenvalues checking.
- **cfode**: It sets all method coefficients and tests constants.
- **f, doblks, and foftex**: They compute the right hand side of the Riccati equation by maximizing efficiency.
- **prepj, jac**: They manage computation and preprocess of the Jacobian matrix and the Newton iteration matrix $p = i - hl_0 j$. There are several important technical details here. These routines call the routines *axpxbd, atxxad, orthes, ortran, hqr3, exchng, qrstep* and *split* to perform an ordered eigenvalue decomposition.
- **solsy**: It manages solution of linear system in the Chord iteration and in the quasi-Newton iteration. It calls the routines *axpxbs, shrslv, atxxas, symslv and sysslv* to solve the decomposed Sylvester equations.
- **ewset**: It sets the error weight vector *ewt* before each step.
- **vnorm**: It computes the weighted max norm of a vector.
- **srcom**: It is a user-callable routine to save and restore the contents of the internal common blocks.
- **r1mach**: It computes the unit roundoff in a machine-independent manner.
- **xerrwv, xsetun, and xsetf**: They handle the printing of all error messages and warnings. *xerrwv* is machine-dependent.

These routines do not use any linear algebra library in order to improve performance, such as BLAS [20] or LAPACK [21]. Only, some EISPACK [25] routines were used. But nowadays, there are new implementations, more robust and more efficient.

Two types of changes can be considered:

- The first one is related to the intensive use of BLAS and LAPACK.
- The second one consists on new block oriented algorithms.

As an example of the first type of changes, consider the following piece of code that belongs to **subroutine f()**

```
     do 20 j=1,nnq
         do 20 i=1,nnp
            prec=0.e0
            do 15 k=1,nnq
               prec=prec-xsave(i,k)*como(k,j)
15          continue
            effe(i,j)=prec
20    continue
```

This block can be replaced by a BLAS subroutine

```
      alpha=-1.0
      beta=0.0
      call sgemm("N","N",nnp,nnq,nnq,alpha,xsave,nnp,como,n,
     $           beta,effe,nnp)
```

Also, in the following piece of code, which belongs to **subroutine prepj()**, an important change can be carried out

```
call orthes(na,n,nlow,nup,a,ort)
call ortran(na,n,nlow,nup,a,ort,u)
call hqr3(a,u,n,nlow,nup,eps,er,ei,itype,na,nu)
```

This piece of code can be replaced by a LAPACK subroutine

```
   call sgees('V','N',dummy,n,a,na,sdim,er,ei,u,nu,
  $           ort,lort,dummy,iflag)
```

More significant changes have been introduced in **subroutine atxxas()** and **subroutine axpxbs()**. In these routines, a Lyapunov and a Sylvester equation of quasi-triangular systems is respectively solved, by using the so-called Bartels-Stewart algorithm [19].

Let's consider the solution of the quasi-triangular Sylvester equation

$$AX + XB = C, \tag{9}$$

where $A \in R^{m\times m}$, $B \in R^{n\times n}$, X, $C \in R^{m\times n}$, being A and B in real Schur form The real Schur form is a triangular form by blocks of dimension 1×1 o 2×2, according to the eigenvalues. That is, if the eigenvalue is real, then the algorithm works with a 1×1 block. On the other hand, when the eigenvalue is complex, the algorithm works with a 2×2 block. The matrices A, B, X and C are partitioned in blocks of dimension $r \times s$, r, $s \in 1, 2$. Then, the following partition is obtained

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1p} \\ & \ddots & \vdots \\ 0 & & A_{pp} \end{pmatrix}, \; B = \begin{pmatrix} B_{11} & \cdots & B_{1q} \\ & \ddots & \vdots \\ 0 & & B_{qq} \end{pmatrix},$$
$$X = \begin{pmatrix} X_{11} & \cdots & X_{1q} \\ \vdots & \ddots & \vdots \\ X_{p1} & \cdots & X_{pq} \end{pmatrix}, \; C = \begin{pmatrix} C_{11} & \cdots & C_{1q} \\ \vdots & \ddots & \vdots \\ C_{p1} & \cdots & C_{pq} \end{pmatrix}.$$

So, matrix A is partitioned into $p \times p$ blocks, matrix B is partitioned into $q \times q$ blocks and matrices X and C are partitioned into $p \times q$ blocks, where all the diagonal blocks have dimension 1×1 or 2×2.

A non-diagonal block A_{ij} has dimension $r \times s$, being the block A_{ii} of dimension $r \times r$ and the block A_{jj} of dimension $s \times s$.

Matching the block (i, j), $1 \le i \le p$, $1 \le j \le q$, at both sides of the equation (9) the following expression is obtained

$$\sum_{k=i}^{p} \sum_{l=1}^{j} \Big(A_{ik} X_{kl} + X_{kl} B_{lj} \Big) = C_{ij}, \tag{10}$$

from which the following $p \times q$ Sylvester equations are obtained

$$A_{ii}X_{ij} + X_{ij}B_{jj} = \hat{C}_{ij}, \tag{11}$$

$\forall i = 1, \ldots, p,\ \forall j = 1, \ldots, q$, with

$$\hat{C}_{ij} = C_{ij} - \sum_{\substack{k=i,l=1 \\ (k,l)\neq(i,j)}}^{p,j} \Big(A_{ik}X_{kl} + X_{kl}B_{lj} \Big). \tag{12}$$

The Sylvester equations (11) are at most of order 2, and then they can be solved using the *trsyl* LAPACK subroutine. Before solving these Sylvester equations, one needs to update the block C_{ij} according to the expression (12). The algorithm to solve the quasi-triangular Sylvester equation is shown in Figure 1

subroutine ***stsyl-version1*** **(** m**,** n**,** A**,** B**,** X**,** C **)**
    for $i = p : -1 : 1$
        for $j = 1 : q$
            $C_{1:i,j} = C_{1:i,j} - \Big(A_{1:i,i}X_{i,1:j-1} + X_{i,1:j-1}B_{1:j-1,j} \Big)$
            trsyl($|A_{ii}|$, $|B_{jj}|$, A_{ii}, B_{jj}, X_{ij}, C_{ij})
            $C_{1:i-1,j} = C_{1:i-1,j} - \Big(A_{1:i-1,i}(X_{ij}) + X_{ij}B_{jj} \Big)$
        end_for
    end_for
end stsyl-version1

Fig. 1. Algorithm to solve the quasi-triangular Sylvester equation. Version 1.

Although the *stsyl-version1* algorithm is block-oriented, the size of these blocks is at most 2×2. This block size is too reduced in order to take benefits of the memory hierarchy of current computers, in particular, the cache memory. From the *stsyl-version1* algorithm is possible to derive the block oriented algorithm of Figure 2.

subroutine ***stsyl-version2*** **(** m**,** n**,** A**,** B**,** X**,** C **)**
    for $i = m/ib : -1 : 1$
        for $j = 1 : n/jb$
            $C_{1:i,j} = C_{1:i,j} - \Big(A_{1:i,i}X_{i,1:j-1} + X_{i,1:j-1}B_{1:j-1,j} \Big)$
            trsyl($|A_{ii}|$, $|B_{jj}|$, A_{ii}, B_{jj}, X_{ij}, C_{ij})
            $C_{1:i-1,j} = C_{1:i-1,j} - \Big(A_{1:i-1,i}X_{ij} + X_{ij}B_{jj} \Big)$
        end_for
    end_for
end stsyl-version2

Fig. 2. Block-oriented algorithm to solve the quasi-triangular Sylvester equation. Version 2.

The block-oriented algorithm *stsyl-version2* is similar to the algorithm *stsyl-version1*, except that the block sizes in the first case are ib and jb, and in the second case either 1×1 or 2×2, depending upon whether the eigenvalues are real or complex.

A similar analysis can be done to obtain a block-oriented algorithm to solve the quasi-triangular Lyapunov equation.

4 Parallel Implementations

The parallel implementation has been done in a multicomputer or distributed memory platform (cluster of PC's).

Basically, clusters are different computers interconnected by means of a network. Each computer has its own memory. Communications between processors are done by message passing through the network. Thus, in these platforms, the bottleneck is the network.

The features of the platform have been taken into account in order to make appropriate and efficient implementations, keeping the good features in all software implementations such as portability, efficiency and robustness. To accomplish these features, standard libraries have been used for computation, communication and synchronization purposes.

In the distributed memory implementation, PBLAS [22] and ScaLAPACK [23] have been used for computation. The message passing interface is provided by the environment MPI [24]. Then, the distributed memory implementation consists on replacing calls to BLAS in the sequential implementation, with calls to PBLAS, and calls to LAPACK, with calls to ScaLAPACK. However, there are not parallel routines to compute the Sylvester (*pstsyl*) or Lyapunov (*pstlyc*) equations by blocks, as noted in the previous section. So, new parallel block oriented routines have been developed for these cases.

Figure 3 shows the parallel block oriented algorithm to solve a quasi-triangular Lyapunov equation. In this code, the updating steps are done in parallel. Meanwhile, the solution of the Lyapunov equation is done by only one processor due to the fact that only one processor has all data blocks without additional communications. This circumstance has been motivated by two factors. The first one is the block oriented implementation. The other one is the initial communications of the diagonal blocks of matrix A involved in the Lyapunov equation.

The parallel implementation carried out can be considered incomplete because a distributed memory implementation of a Schur factorization routine (with reordering facilities) has not been developed, due to the fact that this task was out of the scope of this work.

5 Experimental Results

The experimental results have been obtained in a Cluster of PC's. This platform belongs to the High Performance Networking and Computing Group (GRyCAP) [26] from the Universidad Politécnica de Valencia. The cluster consists on 12

```
Subroutine pstlyc ( n, A, X, C )
    To communicate diagonal blocks of matrix A
    for i = 1 : p
        X_{i,1:i-1} = (X_{1:i-1,i})^T
        for j = i : p
            /* Update I in parallel: block C_{i:j,j} by psgemm */
            C_{i:j,j} = C_{i:j,j} - X_{i,1:j-1} A_{1:j-1,j})
            /* To solve Lyapunov equation by processor (i,j) */
            if idfila==i and idcolumna==j then
              trsyl( |A_ii|, |A_jj|, A_ii^T, A_jj, X_ij, C_ij )
            end if
            /* Update II en parallel: block C_{i+1:j,j} by psgemm*/
            C_{i+1:j,j} = C_{i+1:j,j} - A_{i,i+1:j}^T X_ij
        end for
    end for
end subroutine
```

Fig. 3. Parallel block oriented algorithm for solving the quasi-triangular Lyapunov equation.

nodes (one is used as a front end) IBM xSeries 330 SMP under the RedHat 7.1 operating system. Each node has 2 Pentium III processors at 866 Mhz with 512 MB of main memory and 256 KB of cache memory. The nodes are interconnected by means of a Gigabit Ethernet. In order to consider a true distributed memory platform, only one of the two processors at each node has been used, due to the fact that the communications between processors at each node use threads and not message passing. So, 11 processors can be used in fact for test purposes.

The case of study has been extracted from [1], and consists on the following matrices:

$$A11 = A22 = 0;$$
$$A12 = A21 = \alpha I_n.$$

where n is the variable dimension of the problem and $\alpha > 0$, $0 \leq t \leq T = 10$. Initial values for vector X_0 are ones. The relative and absolute tolerance has the value $RTOL = ATOL = 10^{-3}$.

In our tests the value of n could be 128, 256, 512 and 1024. The value of the variable α is 10.

Another parameter to take into account is *icontr* which allows to solve a symmetric ($icontr = 0$) or non-symmetric ($icontr = 1$) Differential Riccati Equation. This parameter permits to take benefits of the symmetry of the Differential Riccati Equation. In terms of computation, it means to solve a Lyapunov equation (symmetric case) or a Sylvester equation (non-symmetric case).

Due to the fact that the DRESOL package has a lot of routines, it gets difficult to test all of them. So, the following measures have been considered:

(a) Time to perform one iteration of the DRESOL package to solve a symetric Differential Riccati Equation.

(b) Time to perform one iteration of the DRESOL package to solve a non-symetric Differential Riccati Equation.
(c) Time to solve a quasi-triangular Lyapunov equation.
(d) Time to solve a quasi-triangular Sylvester equation.

The graphics in Figure 4 show the execution time obtained by the original DRESOL package and the optimized package in a single processor (sequential version).

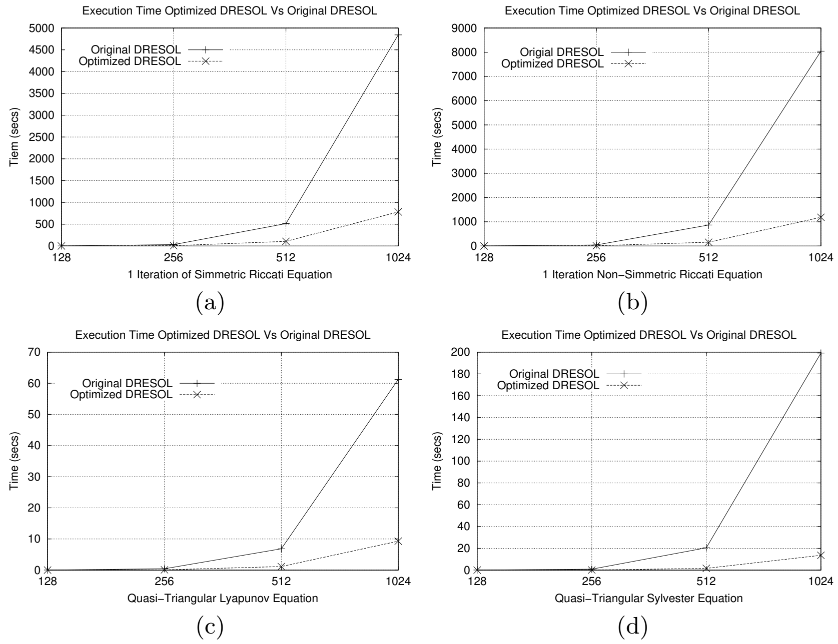

Fig. 4. Comparison between execution times of original DRESOL package and the optimized one. Sequential version. (a) Test a. (b) Test b. (c) Test c. (d) Test d.

The experimental results show a spectacular reduction in the execution time between the original DRESOL package and the optimized package developed in this work. As the dimension of the problem increases, the improvement of the execution time gets better and better. This is due to the fact that the new implementation is block-oriented. Thus, it is able to take benefits of the memory hierarchy of the current computers. In particular, the data flow between main memory and cache memory decreases drastically, thus obtaining better performance. It is also remarkable the fact that better performance is obtained by exploiting the symmetry of the problem, being almost twice faster.

The following parameters have been considered to run the test set into the parallel architecture:

– Block size: This parameter is very important in order to take benefits of the cache memory. The following block sizes have been considered: 8, 16, 32, 64 and 128.

– Number of processors: The values considered are 2, 4, and 8 processors.
– Topology of the grid in ScaLAPACK:
  • 2 processors: 2×1 and 1×2.
  • 4 processors: 4×1, 1×4 and 2×2.
  • 8 processors: 8×1, 1×8, 4×2 and 2×4.
  • 16 processors: 16×1, 1×16, 4×4, 8×2 and 2×8.

The performace obtained in the parallel implementations are evaluated in terms of:

– Execution time: It is the time spent for solving the problem.
– Speed-up: It is defined as the ratio of the time taken to solve a problem on a processor to the time required to solve the same problem on a parallel computer with p identical processors.
– Efficiency: It is a measure of the fraction of time for which a processor is usefully employed; it is defined as the ratio of the speed-up to the number of processors.

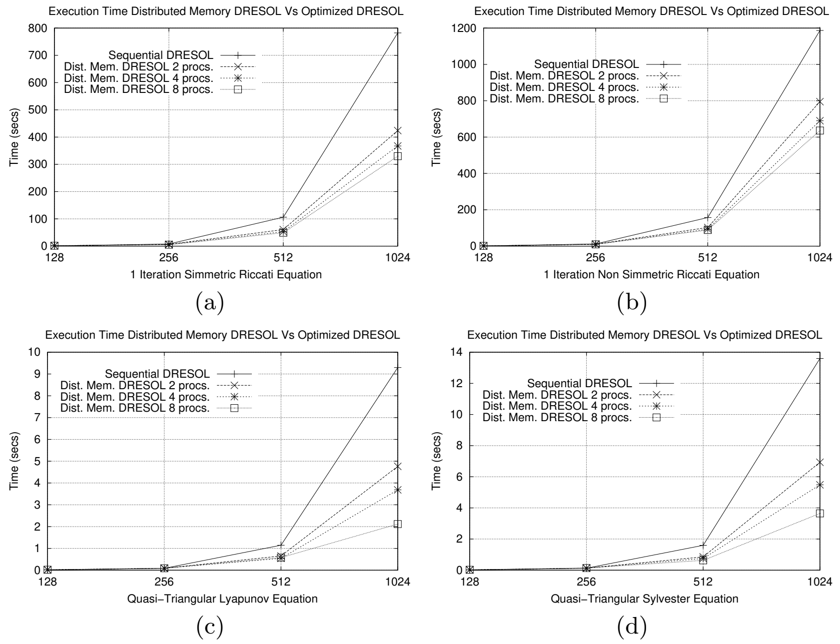

Fig. 5. Comparison between execution times of the optimized DRESOL package and the distributed memory version. (a) Test a. (b) Test b. (c) Test c. (d) Test d.

The graphics in Figure 5 show the execution time obtained by the optimized DRESOL package and the distributed memory version. The execution time has been dramatically reduced by using 2 processors. This reduction becomes less significant by using 4 and 8 processors when the time to perform one iteration is considered. This circumstance is due to the fact that the Schur factorization is sequentially computed, thus performance decreases. Considering the time to solve a Sylvester or Lyapunov equation, it is possible to appreciate a significant reduction in the execution time by using also 4 and 8 processors, because there is not influence of the Schur factorization. Again, better times are obtained by exploting the structure of the problem.

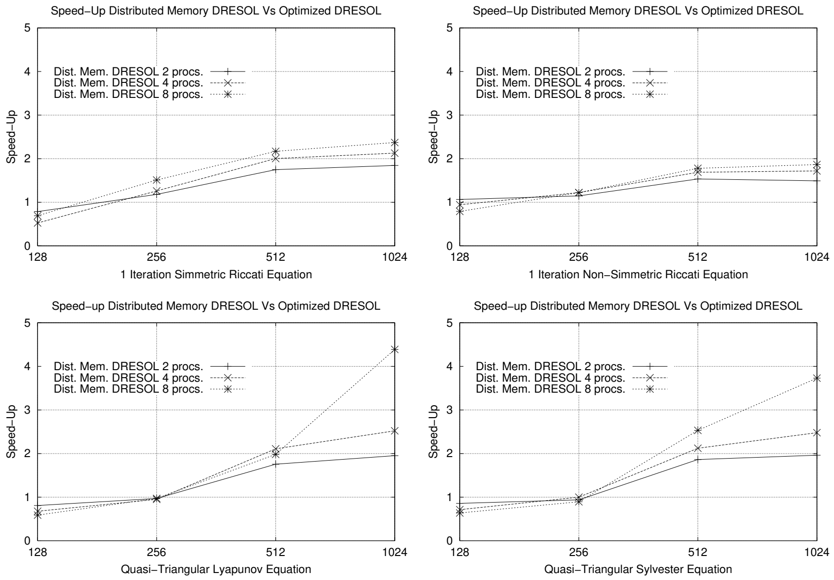

Fig. 6. Speed-Up: distributed memory DRESOL vs optimized DRESOL.

Figure 6 and 7 show the comparison between the optimized DRESOL package and the distributed memory version, in terms of speed-up and efficiency, respectively. In the cases of considering the time to perform one iteration of the code, good speed-up and efficiency are obtained for 2 processors, but not for 4 and 8 processors, due to the circsumstance previously explained. If it is considered the case of solving a Sylvester or Lyapunov equation, performance, in terms of speed-up and efficiency, increases with the size of the problem till almost the theorical values. Also, performance for 4 and 8 processors increases with respect to the size of the problem.

In any case, the times obtained by the sequential optimized implementation are very small, so any reduction of execution time by using the parallel implementation involves a huge amount of work. This amount of work is justified when high dimension problems have to be dealt with in real time.

6 Conclusions

In this paper a High Performance Computing Approach for solving the Differential Riccati Equation by Numerical Integration has been introduced, based on the original package called DRESOL by Luca Dieci [1].

This HPC approach has been developed in two phases. The first one consists on an exhaustive study of the original DRESOL package in order to identify

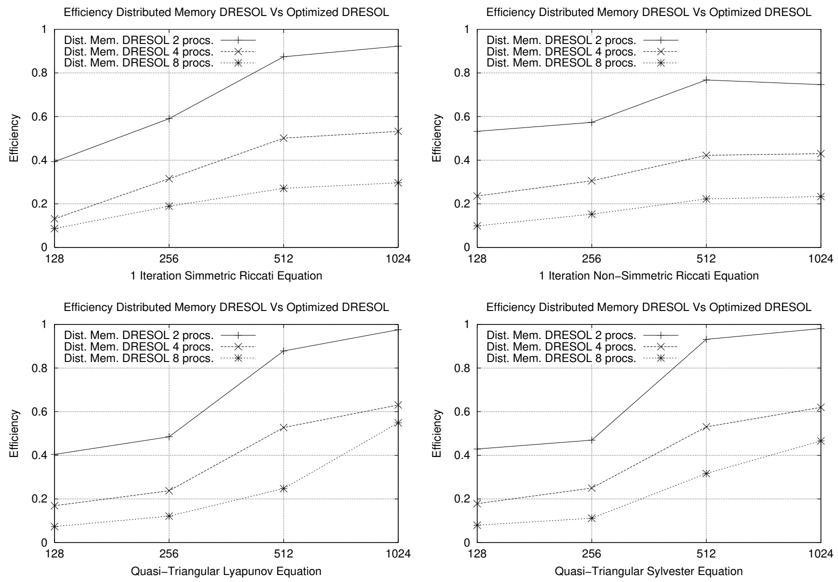

Fig. 7. Efficiency: distributed memory DRESOL vs optimized DRESOL.

the main computational kernels. From this study, a new sequential optimized version of DRESOL has been carried out. From this sequential optimized version, a new parallel implementation has been developed on a distributed memory architecture.

The sequential and the parallel implementations have been carried out taking into account three desirable features in high quality software: Portability, Efficiency and Robustness. Portability has been obtained by using standard libraries for computations (BLAS, LAPACK, PBLAS and ScaLAPACK) and for communications and synchronizations (MPI). The block oriented implementations improve the efficiency of the implementation by taking benefits of the memory hierarchy. Finally, the numerical methods employed in the implementations guarantee the robustness of the software.

With respect to the experimental results, the sequential optimized version dramatically reduces the execution time obtained by the original version. The parallel implementations improve the performance of the new sequential optimized version, achieving speed-up nearly to the theorical maximum.

References

1. Luca Dieci, *Numerical Integration of the diferential Riccati equation and some related issues*, SIAM J. on Mumerical Anaysis, v. 29, n. 3, pp. 781-815, 1992.
2. Isidori,A.. *Nonlinear Control Systems: An Introduction*. Springer-Verlag. 1989.
3. Spong, M.W. and Vidyasagar, M.. *Robot dynamics and control*. John Whiley and Sons. 1989.

4. Slotine, J. J. and W. Li. *Applied Nonlinear Control.* PrenticeHall. 1984.
5. Astrom, K. J. and Wittenmark, B.. *Computer-Controlled Systems: Theory and Design.* Prentice-Hall, Upper Saddle River, New Jersey, 1997.
6. C. S. Kenney and R. B. Leipnik. *Numerical Integration of the Differential Matrix Riccati Equation.* IEEE Transactions on Automatic Control, pp. 962-979, 1985.
7. Chiu H. Choi. *A survey of numerical methods for solving matrix Riccati differential equations.* Southeastcon'90 Proceedings. pp. 696–700, 1990.
8. Lainiotis, D. G.. *Partitioned Riccati solutions and integration-free doubling algorithms.* IEEE Trans. on Automatic Control, 21, pp. 677–689, 1976.
9. Leipnik, R. B.. *A canonical form and solution for the matrix Riccati differential equation.* Bull Australian Math. Soc., 26, pp. 355–261, 1985.
10. Davison, E. J. and Maki, M. C.. *The numerical solution of the matrix Riccati differential equation.* IEEE Trans. on Automatic Control, 18, pp. 71–73, 1973.
11. Rand, D. W. and Winternitz, P.. *Nonlinear Superposition Principles: a new numerical method for solvinf matrix Riccati equations.* Comp. Phys. Commun., 33, pp. 305-328, 1984.
12. Sorine, M and Winternitz, P.. *Superposition laws for the solution of differential Riccati equations.* IEEE Trans. on Automatic Control, 30, pp. 266–272, 1985.
13. Chiu H. Choi and Alan J. Laub. *Efficient Matrix-Valued Algorithms for Solving Stiff Riccati Differential Equations.* IEEE Trans. on Automatic Control, 35-7, pp. 770-776, 1990.
14. A.C. Hindmarsh. *Brief Description of ODEPACK – A Systematized Collection of ODE Solvers*, http://www.netlib.org/odepack/doc
15. Feng Kang and Qin Meng-zhao. *The sumplectic methods for the computation of Hamiltonian equations.* Proceedings of the 1st Chinese Conference for Numerical Methods for PDE's, pp. 1–37, 1987.
16. J. M. Sanz-Serna, *Symplectic integrators for Hamiltonian Problems: an overview*, Acta Numerica, v. 1, pp. 243-286, 1992.
17. Iserles, A. and Norsett, S. P.. *On the solution of linear differential equations in Lie groups.* Technical report, Cambrigde University, 1997.
18. Jeremy Schiff and S. Shnider, *A natural approach to the numerical integration of riccati differential equations*, SIAM, v. 36, n. 5, pp. 1392-1413, 1999.
19. R.H. Bartels and G.W. Stewart, *Solution of the Matrix Equation $AX+XB = C$: Algorithm 432*, CACM, v. 15, pp. 820-826, 1972.
20. Dongarra, J. and Du Croz, J. and Hammarling, S. and Hanson, R. J.. *An Extended Set of FORTRAN Basic Linear Algebra Subroutines.* ACM Trans. Math. Software, 1988. http://www.netlib.org/blas.tgz
21. Anderson, E. and others. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, 1994. http://www.netlib.org/lapack.
22. J. Choi and J. Dongarra and S. Ostrouchov and A. Petitet and D. Walker, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Department of Computer Science, University of Tennessee, Technical Report UT-CS-95-292, May, 1995.
23. Blackford et al., *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, 1997.
24. W. Gropp and E. Lusk and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1994.
25. B. T. Smith, et al, *Matrix Eigensystems Routines - EISPACK Guide*, Springer Lecture Notes in Computer Science, 1976.
26. High Performance Networking and Computing Group. *(GRYCAP).* http://www.grycap.upv.es.