Solving Discrete-Time Periodic Riccati Equations on a Cluster*

Peter Benner[1], Rafael Mayo[2], Enrique S. Quintana-Ortí[2], and Vicente Hernández[3]

[1] Zentrum für Technomathematik, Fachbereich 3/Mathematik und Informatik, Universität Bremen, D-28334 Bremen, Germany;
benner@math.uni-bremen.de

[2] Departamento de Informática, Universidad Jaume I, 12080–Castellón, Spain;
{mayo,quintana}@inf.uji.es

[3] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 46071–Valencia, Spain;
vhernand@dsic.upv.es

Abstract. This paper analyzes the performance of a parallel solver for discrete-time periodic Riccati equations based on a sequence of orthogonal reordering transformations of the monodromy matrices associated with the equations. A coarse-grain parallel algorithm is investigated on a Myrinet cluster.

Key words: Discrete-time periodic Riccati equations, periodic linear control systems, parallel algorithms, multicomputers, cluster computing.

1 Introduction

Consider the discrete-time periodic Riccati equation (DPRE)

$$X_k = Q_k + A_k^T X_{k+1} A_k - A_k^T X_{k+1} B_k (R_k + B_k^T X_{k+1} B_k)^{-1} B_k^T X_{k+1} A_k, \quad (1)$$

where $A_k \in \mathbb{R}^{n\times n}$, $B_k \in \mathbb{R}^{n\times m}$, $C_k \in \mathbb{R}^{r\times n}$, $Q_k \in \mathbb{R}^{n\times n}$, $R_k \in \mathbb{R}^{m\times m}$, and p is the period of the system, i.e., $A_{k+p} = A_k$, $B_{k+p} = B_k$, $C_{k+p} = C_k$, $Q_{k+p} = Q_k$, and $R_{k+p} = R_k$. Under mild conditions, the periodic symmetric positive semidefinite solution $X_k = X_{k+p} \in \mathbb{R}^{n\times n}$ of (1) is unique [3]. DPREs arise, e.g., in the solution of the periodic linear-quadratic optimal control problem, model reduction of periodic linear systems, etc. [3].

Consider now the periodic symplectic matrix pencil, associated with the DPRE (1),

$$L_k - \lambda M_k = \begin{bmatrix} A_k & 0 \\ -Q_k & I_n \end{bmatrix} - \lambda \begin{bmatrix} I_n & B_k R_k^{-1} B_k^T \\ 0 & A_k^T \end{bmatrix} \equiv L_{k+p} - \lambda M_{k+p}, \quad (2)$$

* Supported by the Consellería de Cultura, Educación y Ciencia de la Generalidad Valenciana GV99-59-1-14, the DAAD Programme *Acciones Integradas Hispano-Alemanas*, and the Fundació Caixa-Castelló Bancaixa.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 824–828, 2000.

where I_n denotes the identity matrix of order n. If all the A_k are invertible, the solution of the DPRE is given by the d-stable invariant subspace of the periodic monodromy matrices [5,8]

$$\Pi_k = M_{k+p-1}^{-1} L_{k+p-1} \cdots M_k^{-1} L_k, \quad \Pi_k = \Pi_{k+p}. \tag{3}$$

Note that the monodromy relation still holds if (some of) the A_k are singular and the algorithm presented here can still be applied in this case [3,5,8]. A numerically sound DPRE solver relies on an extension of the generalized Schur vectors method [5,8]. However, the parallel implementation of this QR-like algorithm renders an efficiency and scalability far from those of traditional matrix factorizations [4].

In this paper we follow a different approach, described in [2], for the solution of DPREs based on a reliable swapping of the matrix products in (3). In section 2 we briefly review the "swapping" method for solving DPREs and present a coarse-grain DPRE solver. A medium-grain parallel DPRE solver was investigated in [9]. In section 3 we report the performance of our approach on a cluster of Intel Pentium-II processors.

2 Parallel Solution of DPREs

In [2] an algorithm is introduced for solving DPREs without explicitly forming the monodromy matrices. The approach relies on the following lemma.

Lemma. Consider $Z, Y \in \mathbb{R}^{n \times n}$, with Y invertible, and let

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} Y \\ -Z \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{4}$$

be a QR factorization of $[Y^T, -Z^T]^T$; then $Q_{22}^{-1} Q_{21} = ZY^{-1}$.

By $(Y, Z) \leftarrow swap(Y, Z)$ we denote the application of the lemma to the matrix pair (Y, Z), where Y and Z are overwritten by Q_{22} and Q_{21}, respectively.

Applying the swapping procedure to (3), we obtain reordered monodromy matrices of the form

$$\Pi_k = \hat{M}_k^{-1} \hat{L}_k = (\bar{M}_k \cdots \bar{M}_{k+p-1})^{-1} (\bar{L}_{k+p-1} \cdots \bar{L}_k), \tag{5}$$

without computing any explicit inverse. The solution of the corresponding DPRE is then computed by solving the discrete-time algebraic Riccati equation (DARE) associated with the corresponding matrix pair $\hat{M}_k^{-1} \hat{L}_k$ [10].

The algorithm can be stated as follows [2]:

```
Input: p matrix pairs (L_k, M_k), k = 0, 1, ..., p-1
Output: Solution of the p DPREs associated with the matrix pairs
for k = 0, 1, ..., p-1
    Set L̂_k = L_k, M̂_{(k+1) mod p} = M_k
end
for t = 1, 2, ..., p-1
```

```
    for k = 0, 1, ..., p − 1
        (L_{(k+t) mod p}, M_k) ← swap(L_{(k+t) mod p}, M_k)
        L̂_k ← L_{(k+t) mod p} L̂_k
        M̂_{(k+t+1) mod p} ← M̂_{(k+t+1) mod p} M_k
    end
end
for k = 0, 1, ..., p − 1
    Solve the DARE associated with (L̂_k, M̂_k)
end
```

The procedure is only composed of QR factorizations and matrix products. The computational cost of the reordering algorithm is $\mathcal{O}(p^2n^3)$ flops (floating-point arithmetic operations) and $\mathcal{O}(pn^2)$ for workspace. The cost of the solution of the p DAREs at the final stage is $\mathcal{O}(pn^3)$ flops and $\mathcal{O}(pn^2)$ for workspace.

Consider a parallel distributed-memory architecture, composed of n_p processors, P_0, P_1,..., P_{n_p-1}, and, for simplicity, assume that $p = n_p$. A coarse-grain parallel algorithm can be stated as follows:

```
Input: p matrix pairs (L_k, M_k), k = 0, 1, ..., p − 1
Output: Solution of the p DPREs associated with the matrix pairs
for k = 0, 1, ..., p − 1 in parallel
    Assign (L_k, M_k) to processor P_k
    Set L̂_k = L_k, M̂_{(k+1) mod p} = M_k
    Send L_k to P_{(k+p−1) mod p}
    Receive L_{(k+1) mod p} from P_{(k+1) mod p}
end
for t = 1, 2, ..., p − 1
    for k = 0, 1, ..., p − 1 in parallel
        (L_{(k+t) mod p}, M_k) ← swap(L_{(k+t) mod p}, M_k)
        Send M̂_{(k+t) mod p} to P_{(k+p−1) mod p}
        L̂_k ← L_{(k+t) mod p} L̂_k
        Receive M̂_{(k+t+1) mod p} from P_{(k+1) mod p}
        Send L_{(k+t) mod p} to P_{(k+p−1) mod p}
        M̂_{(k+t+1) mod p} ← M̂_{(k+t+1) mod p} M_k
        Receive L_{(k+t+1) mod p} from P_{(k+1) mod p}
    end
end
for k = 0, 1, ..., p − 1 in parallel
    Solve the DARE associated with (L̂_k, M̂_k)
end
```

This parallel algorithm only requires efficient serial kernels for the QR factorization and the matrix product, like those available in LAPACK and BLAS [1], and a serial DARE solver based, e.g., on the matrix disk function [2]. Moreover, the algorithm presents a highly regular and local communication pattern as, at each iteration of the outer loop t, the only communications neccesary are the left circular shifts of L_k and $\hat{M}_k$. Computation and communication are overlapped in the algorithm, and the computational load in the algorithm is perfectly balanced.

In case $p > n_p$, we can assign the matrix pairs (L_k, M_k) cyclically to the processors, and proceed in the same manner.

3 Experimental Results

All the experiments were performed on a cluster of Intel Pentium-II processors connected via a *Myrinet* switch, using IEEE double precision floating-point arithmetic ($\epsilon \approx 2.2 \times 10^{-16}$). BLAS and MPI implementations specially tuned for this architecture were employed [7]. Performance experiments with routine `DGEMM` achieved 200 Mflops (millions of flops per second) on one processor.

Figure 1 evaluates the efficiency of our coarse-grain parallel algorithm for n=100 and 200, and $n_p = p = 2, 4, \ldots, 10$ processors. We obtain efficiencies higher than 1 due to the better management of the memory achieved in the parallel algorithms, which have to deal with a smaller number of matrices. In this figure we also report the scalability of the parallel algorithm. For this purpose, we evaluate the Mflops per processor, with $n^2 p/n_p$, $p = n_p$, fixed at 200. As the figure shows, the scalability is close to optimal.

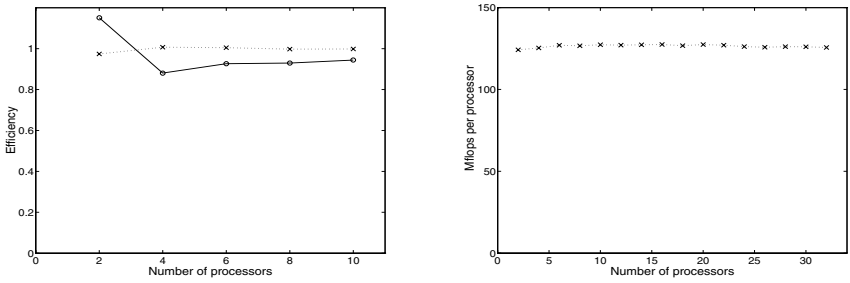

Fig. 1. Efficiency (left) and scalability (right) of the parallel algorithm for n=100 (solid line) and 200 (dotted line).

References

1. E. Anderson et al. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, 1994.
2. P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems*. Dissertation, Fak. f. Mathematik, TU Chemnitz–Zwickau, Chemnitz, FRG, 1997.
3. S. Bittanti, P. Colaneri, and G. De Nicolao. The periodic Riccati equation. In S. Bittanti, A.J. Laub, and J.C. Willems, editors, *The Riccati Equation*, pp. 127–162. Springer-Verlag, Berlin, 1991.
4. L. S. Blackford et al. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

5. A. Bojanczyk, G.H. Golub, and P. Van Dooren. The periodic Schur decomposition; algorithms and applications. In *Proc. SPIE Conference, 1770*, pp. 31–42, 1992.
6. G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, 1989.
7. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.
8. J.J. Hench and A.J. Laub. Numerical solution of the discrete-time periodic Riccati equation. *IEEE Trans. Automat. Control*, 39:1197–1210, 1994.
9. R. Mayo, E.S. Quintana-Ortí, E. Arias, and V. Hernández. *Parallel Solvers for Discrete-time Periodic Riccati Equations*. Lecture Notes in Computer Science. Springer–Verlag, 2000. To appear.
10. V. Mehrmann. *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution*. Number 163 in Lecture Notes in Control and Information Sciences. Springer-Verlag, Heidelberg, July 1991.