Zentrum für Technomathematik

Fachbereich 3 – Mathematik und Informatik

Parallel Algorithms for LQ Optimal Control of Discrete-Time Periodic Linear Systems

Peter Benner **Ralph Byers**
Rafael Mayo **Enrique S. Quintana-Ortí**
Vicente Hernández

Report 01–02

Berichte aus der Technomathematik

Report 01–02 **Februar 2001**

Parallel Algorithms for LQ Optimal Control of Discrete-Time Periodic Linear Systems[1]

Peter Benner

Zentrum für Technomathematik, Fachbereich 3/Mathematik und Informatik, Universität Bremen, 28334 Bremen, Germany. E-mail: benner@math.uni-bremen.de.

Ralph Byers

Dept. of Mathematics, University of Kansas, Lawrence, Kansas 66045, USA. E-mail: byers@math.ukans.edu.

Rafael Mayo, Enrique S. Quintana-Ortí

Depto. de Informática, Universidad Jaume I, 12.080–Castellón, Spain. E-mails: {mayo,quintana}@inf.uji.es.

Vicente Hernández

Depto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 46.071–Valencia, Spain. E-mail: vhernand@dsic.upv.es.

This paper analyzes the performance of two parallel algorithms for solving the linear-quadratic optimal control problem arising in discrete-time periodic linear systems. The algorithms perform a sequence of orthogonal reordering transformations on formal matrix products associated with the periodic linear system, and then employs the so-called matrix disk function to solve the resulting discrete-time periodic algebraic Riccati equations needed to determine the optimal periodic feedback. We parallelize these solvers using two different approaches, based on a coarse-grain and a medium-grain distribution of the computational load.

The experimental results report the high performance and scalability of the parallel algorithms on a Beowulf cluster.

1. INTRODUCTION

In this paper we analyze the parallel solution of the linear-quadratic (LQ) optimal control problem for periodic control systems on parallel computers with distributed memory. Specifically, we consider the discrete-time linear control system

$$\begin{aligned} x_{k+1} &= A_k x_k + B_k u_k, \quad x_0 = x^0, \\ y_k &= C_k x_k, \end{aligned} \qquad k = 0, 1, \ldots, \tag{1}$$

where $A_k \in \mathbb{R}^{n \times n}$, $B_k \in \mathbb{R}^{n \times m}$, and $C_k \in \mathbb{R}^{r \times n}$. The system is said to be periodic if for some integer period p, $A_{k+p} = A_k$, $B_{k+p} = B_k$, and $C_{k+p} = C_k$. The aim in the LQ

[1] Peter Benner and Enrique S. Quintana-Ortí were partially supported by the DAAD programme *Acciones Integradas Hispano-Alemanas*. Ralph Byers was partially supported by National Science Foundation awards CCR-9732671, MRI-9977352, by the NSF EPSCoR/K*STAR program through the Center for Advanced Scientific Computing, and by the National Computational Science Alliance under DMS990005N.

optimal control problem is to find a periodic feedback $\{u_k\}_{k=0}^{\infty}$ which minimizes

$$\frac{1}{2}\sum_{i=0}^{\infty}\left(x_i^T Q_i x_i + u_i^T R_i u_i\right)$$

[10, 11]. Here, $Q_k \in \mathbb{R}^{n\times n}$, $Q_k = Q_k^T \geq 0$, $Q_k = Q_{k+p}$, and $R_k \in \mathbb{R}^{m\times m}$, $R_k = R_k^T > 0$, $R_k = R_{k+p}$.

Under certain assumptions [10, 11], the unique optimal periodic feedback is given by

$$u_k^* = -(R_k + B_k^T X_{k+1} B_k)^{-1} B_k^T X_{k+1} A_k x_k,$$

where $X_k \in \mathbb{R}^{n\times n}$, $X_k = X_{k+p}$, is the unique symmetric positive semidefinite solution of the discrete-time periodic Riccati equation (DPRE)

$$\begin{aligned} 0 \;=\; & C_k^T Q_k C_k - X_k + A_k^T X_{k+1} A_k \\ & - A_k^T X_{k+1} B_k (R_k + B_k^T X_{k+1} B_k)^{-1} B_k^T X_{k+1} A_k; \end{aligned} \tag{2}$$

see [11] for details. In case $p = 1$, the DPRE reduces to the well-known discrete-time algebraic Riccati equation (DARE) [29].

Periodic linear systems naturally arise when performing multirate sampling of continuous linear systems [18]. Large state-space dimension n and/or large period appear, e.g., in the helicopter ground resonance damping problem and the satellite attitude control problem; see, e.g., [9, 22, 26, 37]. The analysis and design of these class of systems has received considerable attention in recent years (see, e.g., [10, 11, 13, 26, 33, 32, 36, 37]).

The need for parallel computing in this area can be seen from the fact that (2) represents a non-linear system with pn^2 unknowns. Reliable methods for solving these equations have a computational cost in flops (floating-point arithmetic operations) of $\mathcal{O}(pn^3)$.

In this paper we analyze the parallelization of two DPRE solvers, introduced in [5, 6], following two different approaches. First, we present a coarse-grain approach which only requires efficient point-to-point communication routines and a few high-performance numerical serial kernels for well-known linear algebra computations. A version of this coarse-grain algorithm with computational cost of $\mathcal{O}(p^2n^3)$ flops was reported in [7, 8]. Here we extend the theoretical analysis of the parallel properties of the algorithm and include a second variant, suggested in [6], with computational cost of $\mathcal{O}(p\log_2(p)n^3)$. Second, we investigate a medium-grain parallel approach, based on the use of parallel linear algebra libraries; in particular, we employ ScaLAPACK [12] to obtain scalable and portable implementations of the solvers. This approach is applied to both algorithms mentioned above.

The paper is structured as follows. In section 2 we briefly review three numerical DPRE solvers based on a reordering of a product of matrices associated with (2). Coarse-grain and medium-grain parallelizations of the solvers are described and analyzed in sections 3 and 4, respectively. In section 5 we report the performance of the algorithms on a Beowulf cluster of Intel$^{\mathrm{TM}}$ Pentium-II processors. This class of parallel distributed computer systems presents a better price/performance ratio than traditional parallel supercomputers and has recently become a cost-effective, widely-spread approach for solving large applications [15]. Finally, some concluding remarks are given in section 6.

2. SOLVING DISCRETE-TIME PERIODIC RICCATI EQUATIONS

In order to solve the LQ optimal control problem for discrete-time perioidc systems, we need to solve the DPRE (2). Here we consider the $2n \times 2n$ periodic matrix pairs associated with this DPRE,

$$L_k = \begin{bmatrix} A_k & 0 \\ -C_k^T Q_k C_k & I_n \end{bmatrix}, \quad M_k = \begin{bmatrix} I_n & B_k R_k^{-1} B_k^T \\ 0 & A_k^T \end{bmatrix}, \quad k = 0, 1, \ldots, p-1,$$

where I_n denotes the identity matrix of order n. In case all the A_k are non-singular, the solution matrices X_k of the DPRE in (2) are given via the invariant subspaces of the periodic matrices [23]

$$\Pi_k \;=\; M_{k+p-1}^{-1} L_{k+p-1} M_{k+p-2}^{-1} L_{k+p-2} \cdots M_k^{-1} L_k, \quad k = 0, 1, \ldots, p-1, \qquad (3)$$

corresponding to the eigenvalues inside the unit disk. Under mild control-theoretic assumptions, the Π_k have exactly n of these eigenvalues. If the columns of $\begin{bmatrix} U_k^T & V_k^T \end{bmatrix}^T$, $U_k, V_k \in \mathbb{R}^{n \times n}$, span this invariant subspace, and the U_k are invertible, then $X_k = -V_k U_k^{-1}$. Note that these relations still hold in a generalized sense if any of the A_k are singular [6, 11, 23, 35]. and all algorithms presented here can still be applied in that case.

The periodic QZ algorithm is a numerically sound DPRE solver which relies on an extension of the generalized Schur vector method [13, 23]. This is a QR-like algorithm with a computational cost of $\mathcal{O}(pn^3)$ flops (it has to deal with p eigenproblems). Several experimental studies report the difficulties in parallelizing this type of algorithms on parallel distributed systems (see, e.g., [24]). The algorithms present a fine granularity which introduces performance losses due to communication start-up overhead. Besides, traditional data layouts (column/row block scattered) lead to an unbalanced distribution of the computational load. These drawbacks can partially be solved by using a block Hankel distribution to improve load balancing [24] and multishift techniques to increase the granularity [25, 14]. Nevertheless, the parallelism and scalability of these algorithms are still far from those of traditional matrix factorizations [12].

In this paper we follow a different approach described in [5, 6] for solving DPREs without explicitly forming the matrix products in (3). The approach relies on the following lemma.

Lemma 1. *Consider $Z, Y \in \mathbb{R}^{q \times q}$, with Y invertible, and let*

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

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

The application of this lemma to a $q \times q$ matrix pair requires $40q^3/3$ flops and storage for $6q^2$ real numbers [6]. Hereafter, we denote by C_{swap} and C_{store} the computational and storage costs, respectively, of applying the above swapping procedure to a matrix pair of size $q = 2n$.

We next describe three different algorithms, based on the swapping lemma, for solving DPREs [5, 6].

2.1. Reordering a matrix product

The basic idea in this first algorithm is to apply the swapping procedure to reorder the matrix products in Π_k in order to obtain

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

without computing any explicit inverse. We describe the procedure by means of an example of period $p = 4$. First, apply the swapping procedure to (L_1, M_0), to obtain a reordered matrix pair $(L_1^{(1)}, M_0^{(1)})$ such that $L_1 M_0^{-1} = (M_0^{(1)})^{-1} L_1^{(1)}$. Then,

$$\begin{aligned}\Pi_0 &= M_3^{-1}L_3M_2^{-1}L_2M_1^{-1}L_1M_0^{-1}L_0\\ &= M_3^{-1}L_3M_2^{-1}L_2M_1^{-1}(M_0^{(1)})^{-1}L_1^{(1)}L_0\\ &= M_3^{-1}L_3M_2^{-1}L_2M_{1:0}^{-1}L_{1:0}.\end{aligned}$$

Here, we borrow the colon notation from [6] to indicate, e.g., that $L_{1:0}$ is obtained by collapsing (computing) the matrix product $L_1^{(1)}L_0$. Next, apply the swapping lemma to $(L_2, M_{1:0})$, to obtain a reordered matrix pair $(L_2^{(1)}, M_{1:0}^{(1)})$ such that

$$\begin{aligned}\Pi_0 &= M_3^{-1}L_3M_2^{-1}L_2M_{1:0}^{-1}L_{1:0}\\ &= M_3^{-1}L_3M_2^{-1}(M_{1:0}^{(1)})^{-1}L_2^{(1)}L_{1:0}\\ &= M_3^{-1}L_3M_{2:0}^{-1}L_{2:0}.\end{aligned}$$

By applying a last time the swapping procedure, to $(L_3, M_{2:0}^{-1})$, we obtain the required reordering in (5).

This stage requires applying the swapping procedure $p-1$ times. The solution of the each DPRE, X_k, is then obtained from the associated matrix pair $(\hat{L}_k, \hat{M}_k)$, using any method that computes the deflating subspace corresponding to the generalized eigenvalues inside the unit disk [28, 30, 31].

In case all A_k, $k = 0, 1, \ldots, p-1$, are non-singular, the remaining $p-1$ matrix products can be obtained from Π_k using the relation

$$\Pi_{k+1} = (M_k^{-1}L_k)^{-1}\Pi_k(M_k^{-1}L_k).$$

Thus, e.g.,

$$\Pi_{k+1} = (M_k^{-1}L_k)^{-1}\Pi_k(M_k^{-1}L_k) = (M_k^{-1}L_k)^{-1}\hat{M}_k^{-1}\hat{L}_k(M_k^{-1}L_k),$$

and this matrix can be obtained from Π_k by applying twice the swapping procedure.

Overall, the computation of Π_k, $k = 0, 1, \ldots, p-1$, requires $3(p-1)C_{comp}$ flops, where $C_{comp} = C_{swap} + 4(2n)^3$, and workspace for $2(p+1)n^2 + C_{store}$ real numbers [6]. The cost of solving the DPRE depends on the method employed to compute the corresponding deflating subspace and will be given in section 2.4.

The previous algorithm requires all A_k to be non-singular. Furthermore, a single moderately ill-conditioned matrix A_k may affect negatively the accuracy of the computed solutions. This algorithm is not considered any further.

2.2. Reordering p matrix products

Next we describe an algorithm that deals with the case of singular A_k matrices, at the expense of increasing the computational cost of the algorithm in the previous subsection

by a factor of p. We illustrate the idea in this algorithm by means of an example of period $p = 4$. Consider the matrices

$$\begin{aligned}
\Pi_0 &= M_3^{-1}L_3M_2^{-1}L_2M_1^{-1}L_1M_0^{-1}L_0,\\
\Pi_1 &= M_0^{-1}L_0M_3^{-1}L_3M_2^{-1}L_2M_1^{-1}L_1,\\
\Pi_2 &= M_1^{-1}L_1M_0^{-1}L_0M_3^{-1}L_3M_2^{-1}L_2,\\
\Pi_3 &= M_2^{-1}L_2M_1^{-1}L_1M_0^{-1}L_0M_3^{-1}L_3,
\end{aligned}$$

and apply the swapping procedure to the matrix pairs (L_3, M_2), (L_2, M_1), (L_1, M_0), and (L_0, M_3). Then, we obtain $(L_3^{(1)}, M_2^{(1)})$, $(L_2^{(1)}, M_1^{(1)})$, $(L_1^{(1)}, M_0^{(1)})$, and $(L_0^{(1)}, M_3^{(1)})$, which satisfy

$$\begin{aligned}
L_3M_2^{-1} &= (M_2^{(1)})^{-1}L_3^{(1)},\\
L_2M_1^{-1} &= (M_1^{(1)})^{-1}L_2^{(1)},\\
L_1M_0^{-1} &= (M_0^{(1)})^{-1}L_1^{(1)}, \quad \text{and}\\
L_0M_3^{-1} &= (M_3^{(1)})^{-1}L_0^{(1)}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\Pi_0 &= M_3^{-1}(M_2^{(1)})^{-1}L_3^{(1)}(M_1^{(1)})^{-1}L_2^{(1)}(M_0^{(1)})^{-1}L_1^{(1)}L_0,\\
\Pi_1 &= M_0^{-1}(M_3^{(1)})^{-1}L_0^{(1)}(M_2^{(1)})^{-1}L_3^{(1)}(M_1^{(1)})^{-1}L_2^{(1)}L_1,\\
\Pi_2 &= M_1^{-1}(M_0^{(1)})^{-1}L_1^{(1)}(M_3^{(1)})^{-1}L_0^{(1)}(M_2^{(1)})^{-1}L_3^{(1)}L_2, \quad \text{and}\\
\Pi_3 &= M_2^{-1}(M_1^{(1)})^{-1}L_2^{(1)}(M_0^{(1)})^{-1}L_1^{(1)}(M_3^{(1)})^{-1}L_0^{(1)}L_3.
\end{aligned}$$

Repeating the swapping procedure with the matrix pairs $(L_3^{(1)}, M_1^{(1)})$, $(L_2^{(1)}, M_0^{(1)})$, $(L_0^{(1)}, M_2^{(1)})$, and $(L_1^{(1)}, M_3^{(1)})$, we obtain $(L_3^{(2)}, M_1^{(2)})$, $(L_2^{(2)}, M_0^{(2)})$, $(L_0^{(2)}, M_2^{(2)})$, and $(L_1^{(2)}, M_3^{(2)})$ such that

$$\begin{aligned}
\Pi_0 &= M_3^{-1}(M_2^{(1)})^{-1}(M_1^{(2)})^{-1}L_3^{(2)}(M_0^{(2)})^{-1}L_2^{(2)}L_1^{(1)}L_0,\\
\Pi_1 &= M_0^{-1}(M_3^{(1)})^{-1}(M_2^{(2)})^{-1}L_0^{(2)}(M_1^{(2)})^{-1}L_3^{(2)}L_2^{(1)}L_1,\\
\Pi_2 &= M_1^{-1}(M_0^{(1)})^{-1}(M_3^{(2)})^{-1}L_1^{(2)}(M_2^{(2)})^{-1}L_0^{(2)}L_3^{(1)}L_2, \quad \text{and}\\
\Pi_3 &= M_2^{-1}(M_1^{(1)})^{-1}(M_0^{(2)})^{-1}L_2^{(2)}(M_3^{(2)})^{-1}L_1^{(2)}L_0^{(1)}L_3.
\end{aligned}$$

A last reordering of the matrix pairs $(L_3^{(2)}, M_0^{(2)})$, $(L_0^{(2)}, M_1^{(2)})$, $(L_1^{(2)}, M_2^{(2)})$, and $(L_2^{(2)}, M_3^{(2)})$, provides the required reordering in (5).

The algorithm can be stated as follows [5]. In the algorithm we denote by $(\bar{Y}, \bar{Z}) \leftarrow swap(Y, Z)$ the application of the swapping lemma to the matrix pair (Y, Z), where $\bar{Y}$ and $\bar{Z}$ are overwritten by Q_{22} and Q_{21}, respectively, using the notation of Lemma 1.

Algorithm 2.1

Input: p matrix pairs (L_k, M_k), $k = 0, 1, \ldots, p-1$.
Output: Solution of the p DPREs associated with the matrix pairs.
for $k = 0, 1, \ldots, p-1$
    $\hat{L}_k \leftarrow L_k$, $\hat{M}_{(k+1) \bmod p} \leftarrow M_k$
end
for $t = 1, 2, \ldots, p-1$
    for $k = 0, 1, \ldots, p-1$
        $(L_{(k+t-1) \bmod p}, M_{(k+p-1) \bmod p}) \leftarrow swap(L_{(k+t-1) \bmod p}, M_{(k+p-1) \bmod p})$

$\hat{M}_{(k+t) \bmod p} \leftarrow M_{(k+p-1) \bmod p} \hat{M}_{(k+t) \bmod p}$
$\hat{L}_k \leftarrow L_{(k+t) \bmod p} \hat{L}_k$
end
end
for $k = 0, 1, \ldots, p-1$
Solve the DPRE using $(\hat{L}_k, \hat{M}_k)$
end

The algorithm is only composed of QR factorizations and matrix products [20]. The computational cost of the reordering procedure is $p(p-1)C_{comp}$ flops and $p(2n^2 + C_{store})$ for workspace [6].

2.3. Reducing the computational cost

In this subsection we describe an algorithm which reduces the computational cost of the previous algorithm to $\mathcal{O}(p \log_2(p) n^3)$ flops, while maintaning a similar numerical behavior.

We use an example of period $p = 4$ to describe the algorithm. Consider the matrices

$$\begin{aligned}
\Pi_0 &= M_3^{-1} L_3 M_2^{-1} L_2 M_1^{-1} L_1 M_0^{-1} L_0, \\
\Pi_1 &= M_2^{-1} L_2 M_1^{-1} L_1 M_0^{-1} L_0 M_3^{-1} L_3, \\
\Pi_2 &= M_1^{-1} L_1 M_0^{-1} L_0 M_3^{-1} L_3 M_2^{-1} L_2, \\
\Pi_3 &= M_0^{-1} L_0 M_3^{-1} L_3 M_2^{-1} L_2 M_1^{-1} L_1,
\end{aligned}$$

and apply the swapping procedure to reorder the matrix products $L_3 M_2^{-1}$, $L_2 M_1^{-1}$, $L_1 M_0^{-1}$, and $L_0 M_3^{-1}$ into $(M_2^{(1)})^{-1} L_3^{(1)}$, $(M_1^{(1)})^{-1} L_2^{(1)}$, $(M_0^{(1)})^{-1} L_1^{(1)}$, and $(M_3^{(1)})^{-1} L_0^{(1)}$, respectively. Note that these matrix products appear twice in $\Pi_0, \ldots, \Pi_3$. Thus, we obtain reordered matrix products

$$\begin{aligned}
\Pi_0 &= M_3^{-1} (M_2^{(1)})^{-1} L_3^{(1)} L_2 M_1^{-1} (M_0^{(1)})^{-1} L_1^{(1)} L_0 = M_{3:2}^{-1} L_{3:2} M_{1:0}^{-1} L_{1:0}, \\
\Pi_1 &= M_2^{-1} (M_1^{(1)})^{-1} L_2^{(1)} L_1 M_0^{-1} (M_3^{(1)})^{-1} L_0^{(1)} L_3 = M_{2:1}^{-1} L_{2:1} M_{0:3}^{-1} L_{0:3}, \\
\Pi_2 &= M_1^{-1} (M_0^{(1)})^{-1} L_1^{(1)} L_0 M_3^{-1} (M_2^{(1)})^{-1} L_3^{(1)} L_2 = M_{1:0}^{-1} L_{1:0} M_{3:2}^{-1} L_{3:2}, \\
\Pi_3 &= M_0^{-1} (M_3^{(1)})^{-1} L_0^{(1)} L_3 M_2^{-1} (M_1^{(1)})^{-1} L_2^{(1)} L_1 = M_{2:1}^{-1} L_{2:1} M_{2:1}^{-1} L_{2:1}.
\end{aligned}$$

Now, we only need to apply the swapping procedure again, to reorder the matrix products $L_{3:2} M_{1:0}^{-1}$, $L_{2:1} M_{0:3}^{-1}$, $L_{1:0} M_{3:2}^{-1}$, $L_{2:1} M_{2:1}^{-1}$, and thus obtain the required reordered matrix products in (5).

The algorithm can be stated as follows. For simplicity, we present the algorithm for p equal to a power of 2.

Algorithm 2.2
Input: p matrix pairs (L_k, M_k), $k = 0, 1, \ldots, p-1$.
Output: Solution of the p DPREs associated with the matrix pairs.
for $t = 1, 2, \ldots, \log_2(p)$
$l \leftarrow 2^{t-1}$
for $k = 0, 1, \ldots, p-1$
$(Y, Z) \leftarrow swap(L_k, M_{(k+p-l) \bmod p})$
$\hat{L}_k \leftarrow Y L_{(k+p-l) \bmod p}$
$\hat{M}_k \leftarrow Z M_k$
end
for $k = 0, 1, \ldots, p-1$

```
        L_k ← L̂_k
        M_k ← M̂_k
    end
end
for k = 0, 1, ..., p − 1
    Solve the DPRE using (L̂_k, M̂_k)
end
```

The computational cost of the reordering procedure in this case is $p\lceil \log_2(p) \rceil C_{comp}$ flops and $2p(2n^2 + C_{store})$ for workspace [6].

2.4. Computing X_k

At the final stage of the three reordering algorithms described above, it is necessary to solve the DPRE (2). As outlined in section 2, these solutions can be obtained from certain invariant subspaces of the formal matrix products Π_k. As these subspaces are exactly the deflating subspaces of the matrix pairs $(\hat{L}_k, \hat{M}_k)$, the X_k are computed from the right deflating subspace of $(\hat{L}_k, \hat{M}_k)$ corresponding to the eigenvalues inside the unit disk.

Here we propose to compute the X_k using the so-called matrix disk function [5]. This matrix function can be computed using an inverse-free iteration, composed of QR factorizations and matrix products, which employs the rationale behind the swapping lemma.

The inverse-free iteration for the matrix disk function was introduced in [27] and made truly inverse-free in [4]. The iterative procedure can be stated as follows.

Algorithm 3.1
Input: A matrix pair (L, M).
Output: Disk function of the matrix pair
$j \leftarrow 0$
$L_0 \leftarrow L$, $M_0 \leftarrow M$
$R_0 \leftarrow 0$.
repeat
Compute the QR factorization

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} M_j \\ -L_j \end{bmatrix} = \begin{bmatrix} R_{j+1} \\ 0 \end{bmatrix}$$

$L_{j+1} \leftarrow Q_{21} L_j$
$M_{j+1} \leftarrow Q_{22} M_j$
$j \leftarrow j + 1$
until $\|R_{j+1} - R_j\|_F < \tau \|R_{j+1}\|_F$

In the algorithm, τ is a user-defined tolerance threshold for the stopping criterion. The disk function of the matrix pair (L, M) is defined from the matrix pair at convergence, denoted by (L_∞, M_∞), as $\text{disk}(L, M) = (L_\infty + M_\infty)^{-1} M_\infty$ [5]. Note that the QR factorization computed at each iteration of the algorithm is exactly the same used as in the swapping lemma.

If we apply Algorithm 3.1 to $(L, M) := (\hat{L}_k, \hat{M}_k)$, then $X := X_k$ can be computed directly from disk (L, M) without computing a basis for the corresponding deflating subspace explicitly. Partition L_∞ into $n \times n$ blocks as

$$L_\infty = \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix};$$

then X is obtained by solving

$$\begin{bmatrix} L_{12} \\ L_{22} \end{bmatrix} X = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix};$$

see [5] for details.

The cost of computing the matrix disk function of a $q \times q$ matrix pair using the inverse-free iteration is $40q^3/3$ flops per iteration. Solving the linear-least square (LLS) system for X adds $13q^3/3$ flops more.

3. COARSE-GRAIN PARALLEL ALGORITHMS

Our coarse-grain parallel algorithms employ a logical linear array of p processes, P_0, P_1,..., P_{p-1}, where p is the period of the system. These parallel algorithms require efficient point-to-point communication routines, high-performance numerical serial kernels for the matrix product and the QR factorization, like those available in BLAS [16] and LAPACK [2], and a serial subspace extraction method based in our case on the matrix disk function [5].

Consider first the parallelization of Algorithm 2.1. In this algorithm, each matrix pair (L_k, M_k) is initially stored in a different process P_k, $k = 0, 1, \ldots, p-1$. During the swapping stage, each swapping of a matrix pair is carried out locally in the process where it is stored. Thus the coarse grain parallelism is obtained by performing each iteration of loop k in Algorithm 2.1 (a swap of a matrix pair) in a different process. By the end of this stage, a reordered matrix pair $(\hat{L}_k, \hat{M}_k)$, as in (5), is stored in process P_k, and the solution of the corresponding DPRE can be obtained locally.

Parallel Algorithm 4.1
Input: p matrix pairs (L_k, M_k), stored in P_k, $k = 0, 1, \ldots, p-1$.
Output: Solution of the p DPREs associated with the matrix pairs and stored in P_k, $k = 0, 1, \ldots, p-1$.
In process P_k:
  $\hat{L}_k \leftarrow L_k$, $\hat{M}_{(k+1) \bmod p} \leftarrow M_k$
  Send M_k to $P_{(k+1) \bmod p}$
  Receive $M_{(k+p-1) \bmod p}$ from $P_{(k+p-1) \bmod p}$
  for $t = 1, 2, \ldots, p-1$
    $(L_{(k+t-1) \bmod p}, M_{(k+p-1) \bmod p}) \leftarrow swap(L_{(k+t-1) \bmod p}, M_{(k+p-1) \bmod p})$
    Send $L_{(k+t-1) \bmod p}$ to $P_{(k+p-1) \bmod p}$
    $\hat{M}_{(k+t) \bmod p} \leftarrow M_{(k+p-1) \bmod p} \hat{M}_{(k+t) \bmod p}$
    Receive $L_{(k+t) \bmod p}$ from $P_{(k+1) \bmod p}$
    Send $\hat{M}_{(k+t) \bmod p}$ to $P_{(k+p-1) \bmod p}$
    $\hat{L}_k \leftarrow L_{(k+t) \bmod p} \hat{L}_k$
    Receive $\hat{M}_{(k+t+1) \bmod p}$ from $P_{(k+1) \bmod p}$
  end
  Solve the DPRE using $(\hat{L}_k, \hat{M}_k)$

The algorithm presents a regular, local communication pattern as the only communications necessary are the left circular shifts of L_k and $\hat{M}_k$.

Assume our system consists of n_p physical processors, $\mathcal{P}_0, \ldots, \mathcal{P}_{n_p-1}$, with $n_p \leq p$. (Using a number of processors larger than p does not produce any benefit in this algorithm.) In the ideal distribution a group of $\lceil (p-r-1)/n_p \rceil + 1$ consecutive processes are mapped

onto processor $\mathcal{P}_r$, $r = 0, \ldots, n_p - 1$. As the communication pattern only involves neighbour processes, this mapping only requires the communication of n_p matrix pairs between neighbour processors at each iteration of loop t. The remaining $p - n_p$ transferences are actually performed inside a processor, and can be implemented as local memory matrix copies.

To derive a theoretical model for our parallel algorithm we use a simplified variant of the "logGp" model [1], where the time required to transfer a message of length l between two processors is given by $\alpha + \beta l$. (Basically, the latency and overhead parameters of the logGp model are combined into α, while no distinction is made between the bandwidth, β^{-1}, for short and long messages.) We also define γ as the time required to perform a flop. Finally, for simplicity, we do not distinguish between the cost of sending a square matrix of size n and that of performing a matrix copy between two processes in the same processor. We use $\alpha + \beta n^2$ in both cases, though we recognize that the matrix copy can be much faster.

An upper bound for the parallel execution time of the previous algorithm is given by

$$\begin{aligned} T^{\max}(n,p,n_p) &= \sum_{t=1}^{p-1} \sum_{k=0}^{\lceil p/n_p \rceil - 1} \left(\gamma(C_{comp} + 2(2n)^3) + 2(\alpha + \beta(2n)^2) \right) \\ &= (p-1)\lceil \tfrac{p}{n_p} \rceil \left(\gamma(C_{comp} + 2(2n)^3) + 2(\alpha + \beta(2n)^2) \right). \end{aligned}$$

The transference of the matrix pairs can be overlapped with the computation of the matrix products using, e.g., a non-blocking (buffered) communication `Send` routine; the execution time will then be

$$T^{\mathrm{ovl}}(n,p,n_p) = (p-1)\lceil \frac{p}{n_p} \rceil \left(\gamma C_{comp} + 2 \ \max\{\gamma(2n)^3, \alpha + \beta(2n)^2\} \right).$$

The actual degree of overlap depends on the efficiency of the communication subsystem and the computational performance of the processors. Usually, $\beta \gg \gamma$ and from a certain "overlapping" threshold $\hat{n}$ communication will be completely overlapped with computation for all $n > \hat{n}$. For a particular architecture, this threshold can be derived as the value of n which satisfies

$$\gamma(2n)^3 \geq \alpha + \beta(2n)^2.$$

In the optimal case communication and computation will be completely overlapped and

$$T^{\mathrm{opt}}(n,p,n_p) = (p-1)\lceil \frac{p}{n_p} \rceil \ \gamma(C_{comp} + 2(2n)^3).$$

The optimal speed-up is then

$$S^{\mathrm{opt}}(n,p,n_p) = \frac{T^{\mathrm{opt}}(n,p,1)}{T^{\mathrm{opt}}(n,p,n_p)} = \frac{p}{\lceil \frac{p}{n_p} \rceil}.$$

This model will surely deviate from the experimental results obtained in section 5. We point out two reasons for this behavior. First, in general α and β depend on the message length. Second, the computational cost of a flop depends on the problem size and the type of operation; e.g., the so-called Level 3 BLAS operations (basically, matrix products) exploit the hierarchical structure of the memory to achieve a lower γ.

Let us consider now the parallelization of Algorithm 2.2. For simplicity, again we only present the algorithm for a period $p = 2^i$ for some positive integer i.

```
Parallel Algorithm 4.2
Input:  p matrix pairs (L_k, M_k), stored in P_k, k = 0, 1, ..., p-1
Output:  Solution of the p DPREs associated with the matrix pairs and
         stored in P_k, k = 0, 1, ..., p-1.
In process P_k:
    Send M_k to P_{(k+1) mod p}
    Receive M_{(k+p-1) mod p} from P_{(k+p-1) mod p}
    for t = 1, 2, ..., log_2(p)
        l ← 2^{t-1}
        (Y, Z) ← swap(L_k, M_{(k+p-l) mod p})
        Send L_k to P_{(k+l) mod p}
        M_k ← Z M_k
        Receive L_{(k+p-l) mod p} from P_{(k+p-l) mod p}
        if (t ≠ log_2(p)) Send M_k to P_{(k+2l) mod p}
        L_k ← Y L_{(k+p-l) mod p}
        if (t ≠ log_2(p)) Receive M_{(k+p-2l) mod p}
    end
    Solve the DPRE using (L̂_k, M̂_k)
```

The analysis on the execution time of this parallel algorithm, the overlapping between communication and computation, and the maximum speed-up attainable follow closely those of Algorithm 4.1.

In the parallel coarse grain algorithms each X_k is computed on a single processor, so only a serial implementation of Algorithm 3.3 is required.

4. MEDIUM-GRAIN PARALLEL ALGORITHMS

The coarse-grain algorithms lose part of their efficiency when the parallel architecture consists of a number of processors n_p larger than the period p of the system. Specifically, in such a case, there are $n_p - p$ idle processors, and the maximum speed-up attainable using n_p processors is limited by p.

To overcome this drawback we propose to use a different parallelization scheme, with a finer grain, where all the processors of the system cooperate to compute each of the matrix operations in the algorithm. Another advantadge of the medium-grain approach in case $p < n_p$ is that the distribution of each matrix among several processors may allow the solution of larger problems (in n) than the coarse-grain approach.

The development of medium-grain parallel algorithms, which work "at the matrix level", is supported by parallel matrix algebra libraries like ScaLAPACK [12] and PLAPACK [34]. Both public libraries rely on the message-passing paradigm and provide parallel routines for basic matrix computations (e.g., matrix-vector product, matrix-matrix products, matrix norms), and solving linear systems, eigenproblems and singular value problems. ScaLAPACK closely mimics the functionality of the popular LAPACK [2], and is used as a black box. PLAPACK basically offers parallel routines for the BLAS [16] and provides the user with an environment for easy development of new, user-tailored codes.

In this paper we propose to use the kernels in ScaLAPACK. This library employs BLAS and LAPACK for serial computations, PB-BLAS (parallel block BLAS) for parallel basic matrix algebra computations, and BLACS (basic linear algebra communication subprograms) for communication. The efficiency of the ScaLAPACK kernels depends on the efficiency of the underlying computational BLAS/PB-BLAS routines and the communica-

tion BLACS library. BLACS can be used on any machine that supports either PVM [19] or MPI [21], thus providing a highly portable environment.

In ScaLAPACK, the user is responsible for distributing the data among the processes. Access to data stored in a different process must be explicitly requested and provided via message-passing. The implementation of ScaLAPACK employs a block-cyclic distribution scheme [12]. The data matrices are mapped onto a logical $p_r \times p_c$ grid of processes. Each process owns a collection of blocks of dimension $m_b \times n_b$, which are locally and contiguously stored in a two-dimensional array in "column-major" order.

For scalability purposes, we map each process onto a different physical processor (i.e., $n_p = p_r \times p_c$), and we use a 2-dimensional block-scattered layout for all our matrices. We employ square blocks of size n_b for the layout, with n_b experimentally determined to optimize performance.

The parallelization of a numerical algorithm using this library consists of distributing the matrices, and identifying and calling the appropriate parallel routines.

The reordering algorithms for the DPRE perform the following matrix operations based on Lemma 1: QR factorization, forming Q_{12} and Q_{22} (these matrices can be computed by applying from the right the transformations computed in the QR factorization to a matrix of the form $[0_n, I_n]$), and matrix product. ScaLAPACK provides routines `PDGEQRF`, `PDORMQR`, and `PDGEMM` for these purposes.

The computation of X_k requires basically the same matrix operations plus the solution of a consistent overdetermined LLS problem at the final stage. This problem can be solved by performing the QR factorization of the coefficient matrix, applying these transformations to the right-hand side matrix, $[L_{11}^T, L_{21}^T]^T$, and solving a triangular linear system. The two first steps can be performed using routines `PDGEQRF`, `PDORMQR`, while the last step is done using `PDTRSM`.

Table 1 reports the number of subroutine calls required by the reordering procedure in the DPRE solvers (swapping stage in Algorithms 2.1 and 2.2) and solving the DPRE from the reordered matrix pair (Algorithm 3.1). In the table, "iter" stands for the number of iterations necessary for convergence in the inverse-free iteration for the matrix disk function.

Table 1
Number of calls to different parallel routines from ScaLAPACK required by the reordering procedure in the DPRE solvers and solving the DPRE from the reordered matrix pair.

	`PDGEQRF`	`PDORMQR` Form Q_{12}, Q_{22}	`PDGEMM`	`PDORMQR` Apply to $[L_{11}^T, L_{21}^T]^T$	`PDTRSM`
Alg. 2.1	$p(p-1)$	$p(p-1)$	$2p(p-1)$	—	—
Alg. 2.2	$p\log_2(p)$	$p\log_2(p)$	$2p\log_2(p)$	—	—
Alg. 3.1	iter $+ 1$	iter	2 iter	1	1

In general, predicting execution time is a complex task due to the large number of factors that have an influence on the final results (system software such as compilers, libraries, operating system; layout block size, processor grid size, actual implementation of collective communication routines, etc.). This is also true for ScaLAPACK, where proposed models

Table 2
Performance of the communication libraries.

Library	α (μs.)	β^{-1} (Mbps.)	
		Short messages	Long messages
MPI/GM API	37.4	213.1	254.4

for the execution time of the routines are oversimplistic and neglect many of these factors; see, e.g., [3, 17]. We therefore do not pursue this goal any further.

5. EXPERIMENTAL RESULTS

Our experiments are performed on a cluster of Intel Pentium-II processors at 300 MHz, with 128 MB of RAM each, using IEEE double-precision floating-point arithmetic ($\epsilon \approx 2.2 \times 10^{-16}$). An implementation of BLAS specially tuned for this architecture was employed. Performance experiments with the matrix product routine in BLAS (`DGEMM`) achieved 180 Mflops (millions of flops per second) on one processor; that roughly provides a parameter $\gamma \approx 5.5$ ns.

The cluster consists of 32 nodes connected by a Myrinet multistage interconnection network[1]. Myrinet provides 1.28 Gbps, full-duplex links, and employs cut-through (wormhole) packet switching with source-based routing. The nodes in our network are connected via two M2M-OCT-SW8 Myrinet crossbar SAN switches. Each node contains a M2M-PCI-32B Myrinet network interface card, with a LANai 4.3 processor, a 33 MHz PCI-bus interface, and 1 MB of RAM.

In our coarse-grain parallel algorithms we employed basic `Send` and `Receive` communication routines in an implementation of MPI, specially tuned for the Myrinet, which makes direct use of the GM API. This is a native application programming interface by Myricom$^{\mathrm{TM}}$ for communication over Myrinet which avoids the overhead of using MPI on top of the TCP/IP protocol stack.

Our medium-grain parallel algorithms are implemented using ScaLAPACK. The communications in this case are performed using BLACS on top of MPI/GM API.

Table 2 reports the communication performance of these libraries measured using a simple *ping-pong* test, both for short (20 KB) and long (500 KB) messages.

5.1. Performance of the coarse-grain parallel reordering

In these algorithms we are interested in finding the threshold from where the communications will be overlapped with the computations. For this purpose, we use data in table 2 and $\gamma \approx 5.5$ ns. to determine the theoretical threshold at $n = 9$.

In practice, the resolution of the timer that we used did not allow to obtain accurate measurements for $n < 30$. For $n \geq 30$ and period p, the coarse-grain parallel algorithms using n_p processors, $n_p = p$, obtained a perfect speed-up. As expected, for $p > n_p$, the speed-up of the algorithm in practice is $\frac{p}{\lceil p/n_p \rceil}$. The scalability of the algorithm as p is increased with the number of processors (p/n_p and n are fixed) is perfect. However, the algorithm is not scalable as n is increased with the number of processors (n^2/n_p and p are

[1] see `http://www.myri.com` for a detailed description.

fixed). The matrix becomes too large to be stored in the memory of a single processor. For more performance results, see [7].

5.2. Performance of the medium-grain parallel reordering

We now evaluate the parallelism and scalability of our medium-grain parallel reordering algorithms.

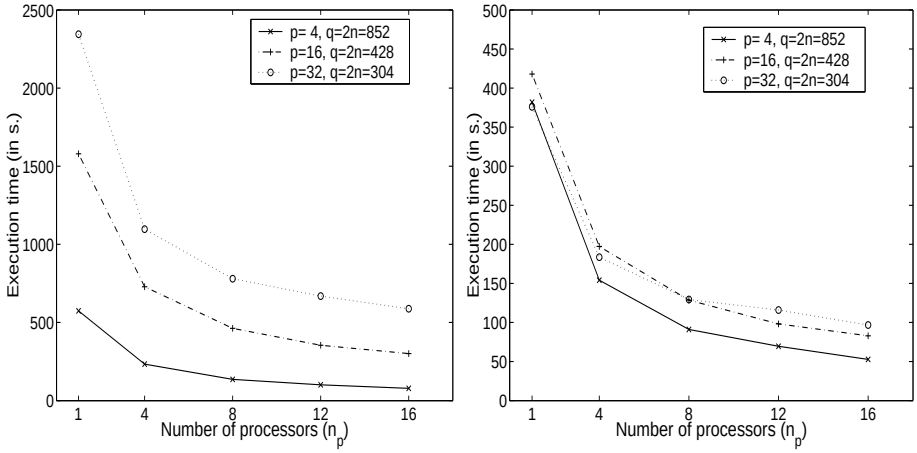

Figure 1. Execution time of the medium-grain parallelization of Algorithm 2.1 (left) and Algorithm 2.2 (right).

Figure 1 reports the execution time for the reordering procedures of the DPRE solvers (Algorithms 2.1 and 2.2), for $p = 4$, 16, and 32. The problem size $q = 2n$ was set to the size of the largest problem sizes that could be solved in 1 processor. The results in the figures show an important reduction in the execution time achieved by the medium-grain parallel reordering algorithms.

Figure 2 reports the scalability of the medium grain algorithms. To analyze the scalability vs. the problem size, $(2n)^2 p/n_p$ is fixed at 460, and we report the Mflop ratio per node, with $p = 4$, on n_p = 4, 8, 16 and 30 processors. There is only a minor decrease in performance and the parallel algorithms can be considered scalable in n. The algorithm however is not scalable with p: As p is increased while $(2n)^2 p/n_p$ is kept fixed, the matrix size per processor is reduced and the performance of the computational matrix kernels will become lower.

Table 3 reports the speed-up of the medium grain reordering algorithms for $p = 4$ and $n = 852$.

5.3. Performance of the DPRE solver

Once the reordering procedure provides the corresponding matrix pair, we only need to use our subspace extraction method, based on matrix disk function, to obtain the solution of the DPRE.

We have already noted that the coarse-grain parallel DPRE solvers only requires a serial implementation of the matrix disk function. In case $p \geq n_p$, the algorithm will achieve a theoretical and experimental speed-up of $\frac{p}{\lceil p/n_p \rceil}$. (There is no overhead due to syncronization or communication as the X_k's can be computed independently). Otherwise, the attained speed-up will be p.

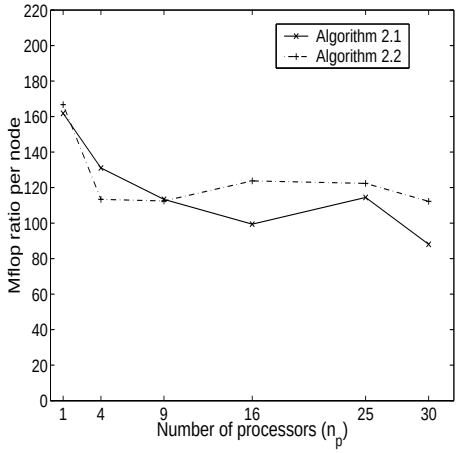

Figure 2. Scalability of the medium grain algorithms vs. problem size. $(2n)^2 p/n_p$=460.

In the medium-grain parallel solvers all the processors in the system participate in the computation of each X_k. Figure 3 reports the execution time of the DPRE solver for a matrix pair of size $q = 2n = 700$, using $n_p = 4$, 8, 12 and 16 processors. The execution time on 1 processor is that of a serial implementation of the algorithm. Ten inverse-free iterations were required in all cases for convergence. The figure also reports the scalability of the solver. In this case we set $n/\sqrt{n_p}$=1000 and evaluate the Mflop ratio (millions of flops per second) per node achieved by the algorithm using a square-like mesh of processors ($n_p = 2 \times 2$, 3×3, 4×4, 5×5, and 5×6).

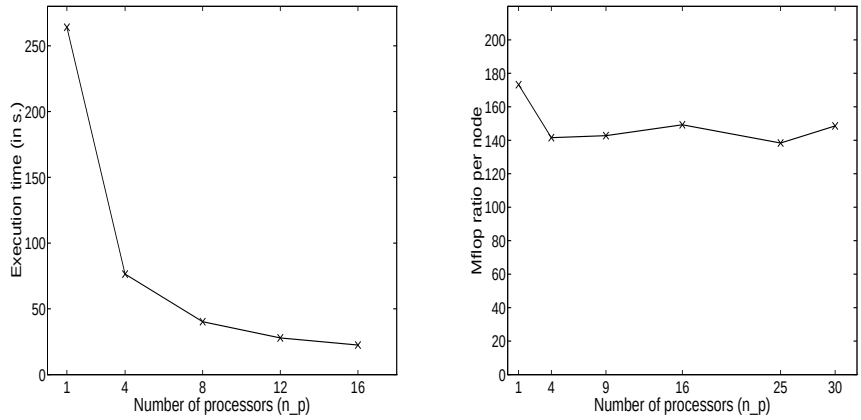

Figure 3. Execution time (left) and scalability (right) of the DPRE solver.

Table 3 reports the speed-up of the DPRE solver for a problem of size $q = 2n = 700$ (see column labeled as Algorithm 3.1).

A comparison between the DPRE solver employed by the coarse-grain and the medium-grain solvers is straight-forward. The coarse-grain solver (serial) will achieve a better performance as long as $p \geq n_p$, as overhead due to communications does not increase. In case $p < n_p$ we would just need to compare the experimental execution time of the medium-grain DPRE solver with that of the serial solver for $p = 1$.

Table 3
Speed-ups of the medium-grain algorithms.

n_p	n=852, p=4		n=350
	Alg. 4.1	Alg. 4.2	Alg. 3.1
4	2.58	2.47	3.45
8	4.71	4.19	6.56
12	5.98	5.49	9.47
16	8.47	7.25	11.76

6. CONCLUDING REMARKS

We have investigated the performance of two parallel algorithms for linear-quadratic optimal control of discrete-time periodic systems. Two different approaches are used to parallelize these solvers. A coarse-grain parallel approach is better suited for problems of period larger than the number of available processors, and moderate dimension of the matrices. The medium-grain parallel solver obtains better results for problems with large-scale matrices. The period does not play an essential role in this case.

The experimental results report the high performance and scalability of the parallel algorithms on a Beowulf cluster.

REFERENCES

1. A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model for parallel computation. *J. Parallel and Distributed Computing*, 44(1):71–79, 1997.
2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, second edition, 1995.
3. Z. Bai, J. Demmel, J. Dongarra, A. Petitet, H. Robinson, and K. Stanley. The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers. *SIAM J. Sci. Comput.*, 18:1446–1461, 1997.
4. Z. Bai, J. Demmel, and M. Gu. An inverse free parallel spectral divide and conquer algorithm for nonsymmetric eigenproblems. *Numer. Math.*, 76(3):279–308, 1997.
5. P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems*. Logos–Verlag, Berlin, Germany, 1997. *Also:* Dissertation, Fakultät für Mathematik, TU Chemnitz–Zwickau, 1997.
6. P. Benner and R. Byers. Evaluating products of matrix pencils and collapsing matrix products. Technical report, Mathematics Research Reports 99-12-01, Department of Mathematics, University of Kansas, 1999. To appear in *Numerical Linear Algebra with Appl.*
7. P. Benner, R. Mayo, E.S. Quintana-Ortí, and V. Hernández. A coarse grain parallel solver for periodic riccati equations. Technical Report 2000–01, Depto. de Informática, 12080-Castellón, Spain, 2000. In preparation.
8. P. Benner, R. Mayo, E.S. Quintana-Ortí, and V. Hernández. Solving discrete-time periodic Riccati equations on a cluster. In A. Bode, T. Ludwig, and R. Wismüller, editors, *Euro-Par 2000 – Parallel Processing*, number 1900 in Lecture Notes in Computer Science, pages 824–828. Springer-Verlag, Berlin, Heidelberg, New York, 2000.
9. P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT - a subroutine library in systems and control theory. In B.N. Datta, editor, *Applied and Computational Control, Signals, and Circuits*, volume 1, chapter 10, pages 499–539. Birkhäuser, Boston, MA, 1999.
10. S. Bittanti and P. Colaneri. Periodic control. In J.G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronic Engineering*, volume 16, pages 59–74. John Wiley & Sons, Inc., New York, Chichester, 1999.
11. 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*, pages 127–162. Springer-Verlag, Berlin, Heidelberg, Germany, 1991.

12. L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

13. A. Bojanczyk, G.H. Golub, and P. Van Dooren. The periodic Schur decomposition. algorithms and applications. In F.T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations III*, volume 1770 of *Proc. SPIE*, pages 31–42, 1992.

14. K. Braman, R. Byers, and R. Mathias. The multi-shift QR-algorithm: Aggressive deflation, maintaining well focused shifts, and level 3 performance. Preprint 99-05-01, Department of Mathematics, University of Kansas, Lawrence, KS 66045-2142, May 1999. Available from `http://www.math.ukans.edu/~reports/1999.html`.

15. R. Buyya. *High Performance Cluster Computing: Architectures & Systems*. Prentice-Hall, 1999.

16. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16(1):1–17, 1990.

17. J. Dongarra, R. van de Geijn, and D. Walker. Scalability issues affecting the design of a dense linear algebra library. *J. Parallel and Distrib. Comput.*, 22(3):523–537, 1994.

18. B.A. Francis and T.T. Georgiou. Stability theory of linear time-invariant plants with periodic digital controllers. *IEEE Trans. Automat. Control*, 33:820–832, 1988.

19. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine – A Users Guide and Tutorial for Network Parallel Computing*. MIT Press, Cambridge, MA, 1994.

20. G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, second edition, 1989.

21. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 1994.

22. M.T. Heath, A.J. Laub, C.C. Paige, and R.C. Ward. Computing the SVD of a product of two matrices. *SIAM J. Sci. Statist. Comput.*, 7:1147–1159, 1987.

23. J.J. Hench and A.J. Laub. Numerical solution of the discrete-time periodic Riccati equation. *IEEE Trans. Automat. Control*, 39:1197–1210, 1994.

24. G. Henry and R. van de Geijn. Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality. *SIAM J. Sci. Comput.*, 17:870–883, 1997.

25. G. Henry, D.S. Watkins, and J.J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. LAPACK Working Note 121, University of Tennessee at Knoxville, 1997.

26. W. Johnson, editor. *Helicopter Theory*. Princeton University Press, Princton, NJ, 1981.

27. A.N. Malyshev. Parallel algorithm for solving some spectral problems of linear algebra. *Linear Algebra Appl.*, 188/189:489–520, 1993.

28. 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.

29. T. Pappas, A.J. Laub, and N.R. Sandell. On the numerical solution of the discrete-time algebraic Riccati equation. *IEEE Trans. Automat. Control*, AC-25:631–641, 1980.

30. P.H. Petkov, N.D. Christov, and M.M. Konstantinov. *Computational Methods for Linear Control Systems*. Prentice-Hall, Hertfordshire, UK, 1991.

31. V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, NY, 1996.

32. J. Sreedhar and P. Van Dooren. Periodic descriptor systems: Solvability and conditionability. *IEEE Trans. Automat. Control*, 44(2):310–313, 1999.

33. J. Sreedhar and P. Van Dooren. Periodic Schur form and some matrix equations. In U. Helmke, R. Mennicken, and J. Saurer, editors, *Systems and Networks: Mathematical Theory and Applications, Proc. Intl. Symposium MTNS '93 held in Regensburg, Germany, August 2–6, 1993*, pages 339–362. Akademie Verlag, Berlin, FRG, 1994.

34. R.A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA, 1997.

35. P. Van Dooren. Two point boundary value and periodic eigenvalue problems. In O. Gonzalez, editor, *Proc. 1999 IEEE Intl. Symp. CACSD, Kohala Coast-Island of Hawai'i, Hawai'i, USA, August 22–27, 1999* (CD-Rom), pages 58–63, 1999.

36. A. Varga. Periodic Lyapunov equations: some applications and new algorithms. *Internat. J. Control*, 67(1):69–87, 1997.

37. A. Varga and S. Pieters. Gradient-based approach to solve optimal periodic output feedback control problems. *Automatica*, 34(4):477–481, 1998.

Berichte aus der Technomathematik ISSN 1435-7968

http://www.math.uni-bremen.de/zetem/berichte.html

— Vertrieb durch den Autor —

Reports **Stand: 9. Februar 2001**

98–01. Peter Benner, Heike Faßbender:
An Implicitly Restarted Symplectic Lanczos Method for the Symplectic Eigenvalue Problem, Juli 1998.

98–02. Heike Faßbender:
Sliding Window Schemes for Discrete Least-Squares Approximation by Trigonometric Polynomials, Juli 1998.

98–03. Peter Benner, Maribel Castillo, Enrique S. Quintana-Ortí:
Parallel Partial Stabilizing Algorithms for Large Linear Control Systems, Juli 1998.

98–04. Peter Benner:
Computational Methods for Linear–Quadratic Optimization, August 1998.

98–05. Peter Benner, Ralph Byers, Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí:
Solving Algebraic Riccati Equations on Parallel Computers Using Newton's Method with Exact Line Search, August 1998.

98–06. Lars Grüne, Fabian Wirth:
On the rate of convergence of infinite horizon discounted optimal value functions, November 1998.

98–07. Peter Benner, Volker Mehrmann, Hongguo Xu:
A Note on the Numerical Solution of Complex Hamiltonian and Skew-Hamiltonian Eigenvalue Problems, November 1998.

98–08. Eberhard Bänsch, Burkhard Höhn:
Numerical simulation of a silicon floating zone with a free capillary surface, Dezember 1998.

99–01. Heike Faßbender:
The Parameterized SR Algorithm for Symplectic (Butterfly) Matrices, Februar 1999.

99–02. Heike Faßbender:
Error Analysis of the symplectic Lanczos Method for the symplectic Eigenvalue Problem, März 1999.

99–03. Eberhard Bänsch, Alfred Schmidt:
Simulation of dendritic crystal growth with thermal convection, März 1999.

99–04. Eberhard Bänsch:
Finite element discretization of the Navier-Stokes equations with a free capillary surface, März 1999.

99–05. Peter Benner:
Mathematik in der Berufspraxis, Juli 1999.

99–06. Andrew D.B. Paice, Fabian R. Wirth:
Robustness of nonlinear systems and their domains of attraction, August 1999.

99–07. Peter Benner, Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí:
Balanced Truncation Model Reduction of Large-Scale Dense Systems on Parallel Computers, September 1999.

99–08. Ronald Stöver:
Collocation methods for solving linear differential-algebraic boundary value problems, September 1999.

99–09. Huseyin Akcay:
Modelling with Orthonormal Basis Functions, September 1999.

99–10. Heike Faßbender, D. Steven Mackey, Niloufer Mackey:
Hamilton and Jacobi come full circle: Jacobi algorithms for structured Hamiltonian eigenproblems, Oktober 1999.

99–11. Peter Benner, Vincente Hernández, Antonio Pastor:
On the Kleinman Iteration for Nonstabilizable System, Oktober 1999.

99–12. Peter Benner, Heike Faßbender:
A Hybrid Method for the Numerical Solution of Discrete-Time Algebraic Riccati Equations, November 1999.

99–13. Peter Benner, Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí:
Numerical Solution of Schur Stable Linear Matrix Equations on Multicomputers, November 1999.

99–14. Eberhard Bänsch, Karol Mikula:
Adaptivity in 3D Image Processing, Dezember 1999.

00–01. Peter Benner, Volker Mehrmann, Hongguo Xu:
Perturbation Analysis for the Eigenvalue Problem of a Formal Product of Matrices, Januar 2000.

00–02. Ziping Huang:
Finite Element Method for Mixed Problems with Penalty, Januar 2000.

00–03. Gianfrancesco Martinico:
Recursive mesh refinement in 3D, Februar 2000.

00–04. Eberhard Bänsch, Christoph Egbers, Oliver Meincke, Nicoleta Scurtu:
Taylor-Couette System with Asymmetric Boundary Conditions, Februar 2000.

00–05. Peter Benner:
Symplectic Balancing of Hamiltonian Matrices, Februar 2000.

00–06. Fabio Camilli, Lars Grüne, Fabian Wirth:
A regularization of Zubov's equation for robust domains of attraction, März 2000.

00–07. Michael Wolff, Eberhard Bänsch, Michael Böhm, Dominic Davis:
Modellierung der Abkühlung von Stahlbrammen, März 2000.

00–08. Stephan Dahlke, Peter Maaß, Gerd Teschke:
Interpolating Scaling Functions with Duals, April 2000.

00–09. Jochen Behrens, Fabian Wirth:
A globalization procedure for locally stabilizing controllers, Mai 2000.

00–10. Peter Maaß, Gerd Teschke, Werner Willmann, Günter Wollmann:
Detection and Classification of Material Attributes – A Practical Application of Wavelet Analysis, Mai 2000.

00–11. Stefan Boschert, Alfred Schmidt, Kunibert G. Siebert, Eberhard Bänsch, Klaus-Werner Benz, Gerhard Dziuk, Thomas Kaiser:
Simulation of Industrial Crystal Growth by the Vertical Bridgman Method, Mai 2000.

00–12. Volker Lehmann, Gerd Teschke:
Wavelet Based Methods for Improved Wind Profiler Signal Processing, Mai 2000.

00–13. Stephan Dahlke, Peter Maass:
A Note on Interpolating Scaling Functions, August 2000.

00–14. Ronny Ramlau, Rolf Clackdoyle, Frédéric Noo, Girish Bal:
Accurate Attenuation Correction in SPECT Imaging using Optimization of Bilinear Functions and Assuming an Unknown Spatially-Varying Attenuation Distribution, September 2000.

00–15. Peter Kunkel, Ronald Stöver:
Symmetric collocation methods for linear differential-algebraic boundary value problems, September 2000.

00–16. Fabian Wirth:
The generalized spectral radius and extremal norms, Oktober 2000.

00–17. Frank Stenger, Ahmad Reza Naghsh-Nilchi, Jenny Niebsch, Ronny Ramlau:
A unified approach to the approximate solution of PDE, November 2000.

00–18. Peter Benner, Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí:
Parallel algorithms for model reduction of discrete–time systems, Dezember 2000.

00–19. Ronny Ramlau:
A steepest descent algorithm for the global minimization of Tikhonov–Phillips functional, Dezember 2000.

01–01. Efficient methods in hyperthermia treatment planning:
Torsten Köhler, Peter Maass, Peter Wust, Martin Seebass, Januar 2001.

01–02. Parallel Algorithms for LQ Optimal Control of Discrete-Time Periodic Linear Systems:
Peter Benner, Ralph Byers, Rafael Mayo, Enrique S. Quintana-Ortí, Vicente Hernández, Februar 2001.