A parallel Schur method for solving continuous-time algebraic Riccati equations*

Robert Granat**, Bo Kågström**, and Daniel Kressner***

***Abstract*— Numerical algorithms for solving the continuous-time algebraic Riccati matrix equation on a distributed-memory parallel computer are considered. In particular, it is shown that the Schur method, based on computing the stable invariant subspace of a Hamiltonian matrix, can be parallelized in an efficient and scalable way. Our implementation employs the state-of-the-art library ScaLAPACK as well as recently developed parallel methods for reordering the eigenvalues in a real Schur form. Some experimental results are presented, confirming the scalability of our implementation and comparing it with an existing implementation of the matrix sign iteration from the PLiCOC library.**

I. INTRODUCTION

The continuous-time algebraic Riccati equation (CARE) is defined as

$$0 = Q + A^T X + XA - XGX, \tag{1}$$

with $A \in \mathbb{R}^{n\times n}$, $Q, G \in \mathbb{R}^{n\times n}$ symmetric positive semi-definite, and the solution matrix $X \in \mathbb{R}^{n\times n}$. In many applications, such as quadratic optimal control of parabolic partial differential equations, the order n of the coefficients may become quite large. If n significantly exceeds $\mathcal{O}(10^4)$ then traditional approaches for solving the CARE, such as Laub's Schur method [33] implemented in the MATLAB control toolbox, will fail on a serial computer due to an excessive demand for memory and computing time. There are two approaches to avoid this problem:

(1) algorithms that exploit sparsity and/or low-rank structure of the coefficients (e.g., [4], [5], [26], [29]);
(2) parallel variants of existing serial algorithms (e.g., [6], [10], [15]).

While approach (2) still limits the value of n to, say, $\mathcal{O}(10^6)$, it has the advantage of not requiring any restrictive assumption on the coefficients of (1). Typically, approach (1) involves inexact iterative methods and consequently imposes limits on the accuracy and reliability of the solution. In this paper, we therefore focus on approach (2) and show how Laub's Schur method can be parallelized.

The rest of this paper is organized as follows. In Section II, we briefly review applications leading to CAREs. While Section II represents a summary of existing parallelization strategies for the CARE, our newly proposed parallel variant of Laub's Schur method is presented in Section IV. Finally, the main contribution is in Section V, showing the performance and scalability properties of the parallel Schur method and comparing it with the only (so far) publicly available parallel solver for the CARE, the PLiCOC implementation [10] based on the matrix sign function.

*This research was conducted using the resources of the High Performance Computing Center North (HPC2N). Financial support has been provided by the *Swedish Research Council* under grant VR 621-2001-3284 and by the *Swedish Foundation for Strategic Research* under grant A3 02:128.

**Department of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden. ***Seminar für angewandte Mathematik, ETH Zürich, Switzerland. {granat,bokg}@cs.umu.se, kressner@sam.math.ethz.ch

II. APPLICATIONS

All applications presented below are related to a continuous-time linear time-invariant system

$$\begin{array}{rcl} \dot{x}(t) & = & Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x_0, \\ y(t) & = & Cx(t), \end{array} \tag{2}$$

with states $x(t) \in \mathbb{R}^n$, inputs $u(t) \in \mathbb{R}^m$, and outputs $y(t) \in \mathbb{R}^p$.

A. Linear-quadratic optimal control

Consider the minimization of

$$J(u) = \int_0^\infty \left(y(t)^T \tilde{Q} y(t) + u(t)^T \tilde{R} u(t) \right) \, \mathrm{d}t, \tag{3}$$

subject to the dynamical constraints (2) for piecewise continuous controls u. It is required that $\tilde{Q} = \tilde{Q}^T$, $\tilde{R} = \tilde{R}^T$, and $\tilde{R}$ is positive definite. Under additional mild assumptions [36], the minimum of (3) is attained by the linear feedback law

$$u_\star = -\tilde{R}^{-1} B^T X x(t),$$

where X is the *stabilizing solution* of

$$0 = C^T \tilde{Q} C + A^T X + XA - XB\tilde{R}^{-1}B^T X, \tag{4}$$

i.e., the eigenvalues of the closed loop matrix $A - B\tilde{R}^{-1}B^T X$ are in the open left half plane.

Note that (4) is a CARE with $Q = C^T\tilde{Q}C$ and $G = B\tilde{R}^{-1}B^T$. In the frequently encountered case that there are much less inputs/outputs than states both coefficient matrices are of low rank.

B. Stochastic balancing

Again we consider (2), but with the output equation appended by a feedthrough term:

$$y(t) = Cx + Du(t). \tag{5}$$

We assume that A is stable, $p \leq m$, and $D \in \mathbb{R}^{p\times m}$ has full row rank. Sometimes unrealistic, the last requirement can always be achieved after a suitable regularization.

Under these conditions, the left spectral factor $W(s)$ of the transfer function $G(s) = C(sI - A)^{-1}B + D$ can be determined, such that $W^T(-s)W(s) = G(s)G^T(-s)$. A state

space realization (A_W, B_W, C_W, D_W) of W is obtained as $A_W = A$, $B_W = PC^T + BD^T$, $C_W = E^{-1/2}(C - B_W^T X)$ and $D_W = E^{1/2}$, with $E = DD^T$. Here, P is the solution of the Lyapunov equation

$$0 = AP + PA^T + BB^T,$$

while X is the stabilizing solution of the CARE

$$\begin{aligned} 0 &= (A - B_W E^{-1} C)^T X + X(A - B_W E^{-1} C) \\ &\quad + X B_W E^{-1} B_W^T X + C^T E^{-1} C. \end{aligned} \tag{6}$$

Stochastic balancing determines a state-space transformation T such that the correspondingly transformed solutions $T^{-1} P T^{-T}$, $T^T X T$ are equal and diagonal. Once P and X are computed, such a matrix T can be computed from the singular value decomposition of the matrix product PX, very much like in ordinary balancing, see also [42]. A main motivation of this procedure, *balanced stochastic truncation* then consists of truncating this balanced system. Note that the major computational task consists of solving the CARE (6).

C. Other applications

The need for solving CAREs arises in various other areas of systems and control theory, such as robust and H_∞ control [27], [43].

III. SURVEY OF EXISTING PARALLEL METHODS

The ubiquity of algebraic Riccati equations in systems and control has led to intensive research in designing parallel methods for solving large scale problems. Most of the existing approaches are based on building blocks for which a parallel implementation is either rather straightforward or already publicly available [12].

Some approaches use the intimate relation of CARE to the *Hamiltonian matrix*

$$H = \begin{bmatrix} A & G \\ Q & -A^T \end{bmatrix}. \tag{7}$$

Under mild conditions, the $2n \times 2n$ matrix H has n stable eigenvalues. If $\left[\begin{smallmatrix} X_1 \\ X_2 \end{smallmatrix}\right]$ with $X_1, X_2 \in \mathbb{C}^{n\times n}$ denotes a basis for the invariant subspace belonging to these eigenvalues then X_1 is invertible and $X = -X_2 X_1^{-1} \in \mathbb{R}^{n\times n}$ is the stabilizing solution of CARE.

A. Jacobi methods

The Hamiltonian-Jacobi method [15], [17], [35] aims at computing a unitary matrix U such that

$$U^H H U = \begin{bmatrix} T & N \\ 0 & -T^H \end{bmatrix}, \tag{8}$$

where $N = N^H \in \mathbb{C}^{n\times n}$ and $T \in \mathbb{C}^{n\times n}$ is upper triangular with all stable eigenvalues of H on the diagonal of T. Then the first n columns of U form a basis for the desired invariant subspace of H.

In the variants described in [15], [35], the factorization (8) is achieved by a sequence of unitary similarity transformations each of which transforms a 4×4 Hamiltonian submatrix of H to the form (8). One complete sweep of transformations processes the entire matrix and requires $\mathcal{O}(n^3)$ operations. For the examples reported in [15], between 10–20 sweeps were necessary to converge sufficiently close to the form (8). In general – extrapolating from well-known results for symmetric matrices – one would expect that the number of sweeps grows at least proportionally to $\log_2 n$, resulting in a significant increase of operations compared to standard methods for solving CARE. Another drawback is that the realness of H is not preserved. On the other hand, as each transformation requires only local information, Jacobi methods can be parallelized with very little communication overhead.

B. Sign function iteration

Given a Jordan canonical form of H,

$$P^{-1} H P = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix}, \qquad J_1, J_2 \in \mathbb{C}^{n\times n},$$

such that J_1 contains all stable eigenvalues, the *matrix sign function* of H is defined as

$$\operatorname{sign}(H) = P \begin{bmatrix} -I & 0 \\ 0 & I \end{bmatrix} P^{-1}.$$

In particular, the first n columns of $\operatorname{sign}(H) - I_{2n}$ form a basis for the desired invariant subspace of H.

The matrix sign function can be computed by applying the Newton iteration method to the equation $Z^2 = I$:

$$Z_0 \leftarrow H, \quad Z_{j+1} \leftarrow \frac{1}{2}\left(Z_j + Z_j^{-1}\right), \quad j = 0, 1, \ldots.$$

It can be shown [39] that the iterates Z_j converge globally and locally quadratic to $\operatorname{sign}(H)$. Initial convergence, however, might be slow, especially when H has eigenvalues close to the imaginary axis. In this case, also the norms of the initial iterates may significantly grow, adversely affecting the numerical stability of the method. Byers' determinantal scaling strategy [16], implemented in the PLiCOC library [10], aims at avoiding this effect:

$$Z_0 \leftarrow H, \quad Z_{j+1} \leftarrow \frac{1}{2}\left(\beta_j Z_j + \frac{1}{\beta_j} Z_j^{-1}\right), \quad j = 0, 1, \ldots,$$

with $\beta_j = (\det Z_j)^{-1/n}$. Note that β_j can be computed at virtually no extra cost if Z_j^{-1} is computed via the LU factorization of Z_j. For the examples reported in [10], convergence was usually attained after at most 10–25 iterations. However, even with the scaling strategy, maximum accuracy can only be obtained when combined with iterative refinement, see Section III-C below.

A major advantage of the sign function iteration is that its computational core consists of little more than explicit matrix inversion, which has a well-tested highly scalable parallel implementation in ScaLAPACK [12].

C. Newton method

The Newton method can also be directly applied to CARE. For some symmetric initial starting guess X_0, this leads to the iteration $X_{j+1} \leftarrow X_j + N_j$, where the correction N_j solves the Lyapunov equation

$$0 = (A - GX_j)^T N_j + N_j (A - GX_j) + \operatorname{Res}(X_j). \tag{9}$$

Here, $\mathrm{Res}(X_j) = Q + A^T X_j + X_j A - X_j G X_j$ denotes the residual of CARE for X_j.

Provided $A - GX_0$ is stable, the Newton iteration converges globally and locally quadratic to the stabilizing solution of CARE. Line search strategies [6] should be used to avoid undesirable initial behavior. To solve (9) in parallel, the PLiCOC library uses a specialized matrix sign function iteration. Alternatives could be based on the Bartels-Stewart method, combining the parallel QR algorithm in ScaLAPACK with the SCASY library [23], [24], [40]. The Newton method for CARE enjoys the self-correcting properties of general Newton methods and is thus numerically stable. However, since solving (9) is significantly more expensive than one step of the sign function iteration, it is advisable to use the Newton method only for refining a computed solution from a less accurate method, such as the sign function iteration, or the Schur method.

D. Other methods

By a Cayley transform, the Hamiltonian matrix (7) can be turned into a symplectic pencil. This simple trick often allows to extend methods for solving discrete-time Riccati equations to CARE. However, it is dubious whether such an approach offers any advantage over methods that tackle CARE directly. For example, numerical experiments in [10] reveal that Malyshev's inverse free iteration [34] applied to the symplectic pencil requires significantly more execution time than the sign function iteration.

IV. PARALLELIZATION OF THE SCHUR METHOD

The Schur method [33] applies the QR algorithm [21] to the Hamiltonian matrix H in (7) in order to compute an orthogonal matrix U_1 such that $S = U_1^T H U_1$ is in real Schur form, i.e., block upper triangular with 1×1 and 2×2 blocks on the diagonal each corresponding to a real eigenvalue or a pair of complex conjugate eigenvalues. Assuming that H has no eigenvalues on the imaginary axis, we can reorder the eigenvalues of S and compute an orthogonal matrix U_2 such that

$$U_2^T S U_2 = \bar{S} \equiv \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix}, \quad S_{11}, S_{22} \in \mathbb{R}^{n \times n},$$

where S_{11} is stable. As explained in Section III, the solution of CARE can then be easily obtained from the first n columns of $U = Q_1 Q_2$.

Our implementation builds on the existing functionality of ScaLAPACK (see, e.g., [12], [41]) and is based on its conventions of the parallel distributed memory (DM) environment, as follows:

- The parallel processers are organized into a rectangular $P_r \times P_c$ mesh labelled from $(0, 0)$ to $(P_r - 1, P_c - 1)$ according to their specific position indices in the mesh.
- The matrices are distributed over the mesh using 2-dimensional (2D) block cyclic mapping with the block sizes m_b and n_b in the row and column dimensions, respectively.

In this contribution we consider only square blocks in the data distribution ($n_b = m_b$). The implementation also follows the outline of the state-of-the-art routine `SB02MD` from SLICOT [8]:

- Construction of the Hamiltonian matrix H (7).
- Computation of the real Schur form S of$H$.
- Computation of the corresponding ordered real Schur form $\bar{S}$ by separating the stable and the unstable eigenvalues.
- Computation of the solution X from the resulting stable invariant subspace.

Each of these steps is a challenge in a distributed memory environment. First of all, the construction of the Hamiltonian matrix requires some care to avoid excessive data copying over the network; a specialized routine was developed for this purpose. Second, the efficient initial parallel reduction of the Hamiltonian matrix to Hessenberg form needed by the QR algorithm is a nontrivial task [11], [18] (see ScaLAPACK's `PDGEHRD`). This is even more the case for the QR algorithm itself, for which the only publicly available scalable implementation [28] (see ScaLAPACK's `PDLAHQR`) suffers from low node performance. Finally, reliable and efficient software for parallel eigenvalue reordering was missing in the context of ScaLAPACK until recently [25].

V. EXPERIMENTS

In this section, we present some experimental results that demonstrate the parallel performance of our implemented Schur-based CARE solver `PSB02MD`.

Our target machine is the 64-bit Opteron Linux Cluster *sarek* with 192 dual AMD Opteron nodes (2.2 GHz), 8Gb RAM per node and a Myrinet-2000 high-performance interconnect with 250 MB/sec bandwidth. All experiments were conducted using the Portland Group's `pgf77 1.2.5 64-bit` compiler, the compiler flag `-fast` and the following software: MPICH-GM 1.5.2 [37], LAPACK 3.0 [32], GOTO-BLAS r0.94 [22], ScaLAPACK 1.7.0 [41] and BLACS 1.1patch3 [13]. All experiments were performed in double precision arithmetic ($\epsilon_{\mathrm{mach}} \approx 2.2 \times 10^{-16}$).

We consider different collections of random problems and several scalable benchmark problems. Furthermore, we compare our implementation with existing software from PLiCOC [9], [10]: From PLiCOC we utilize the following two driver routines:

`PDGECOCA` contains an implementation of the sign function iteration with the scaling strategy described in Section III-B;

`PDGECRNX` contains an implementation of the Newton method described in Section III-C, with the Lyapunov equation (9) solved by a specialized sign function iteration.

Notice that `PDGECOCA` solves the LQ-optimal problem with the matrices G and Q factorized as in (4).

For fair comparisons, exactly the same problem is generated for both solvers. Moreover, our parallel Schur-based solver is sometimes combined with iterative refinement using

TABLE I

PARAMETERS AND OUTPUT VARIABLES FOR THE EXPERIMENTS.

$P_r \times P_c$	Parallel processer mesh configuration.
T_p	Overall parallel execution time in seconds.
S_p	Parallel speedup $T_{p_{\min}(n)}/T_p$, where $p = P_r \cdot P_c$ and $p_{\min}(n)$ is the min. number of processors used for n.
R_r	$\frac{\|Q + A^T X + XA - XGX\|_F}{(\|Q\|_F + 2\|A\|_F\|X\|_F + \|X\|_F\|G\|_F\|X\|_F)}$ (relative norm of residual).
R_e	Relative norm of forward error, $\|\tilde{X} - X\|_F / \|X\|_F$, where X and $\tilde{X}$ are the computed and the exact solutions, respectively.
I_s	Number of iterations needed by sign function iteration.
I_o	Number of outer Newton iterations needed by iterative refinement.
I_i	Overall number of inner sign function iterations needed by iterative refinement.

TABLE II

EXPERIMENTAL RESULTS FOR PSBO2MD WITHOUT ITERATIVE REFINEMENT SOLVING RANDOM PROBLEMS ON *sarek*.

		A diag. dom.			A not diag. dom.		
n	$P_r \times P_c$	T_p	S_p	R_r	T_p	S_p	R_r
1000	1×1	188	1.00	0.21E-15	145	1.00	0.76E-15
1000	2×2	97.4	1.93	0.77E-16	90.9	1.60	0.66E-15
1000	4×4	35.1	5.35	0.36E-15	38.6	3.77	0.72E-15
1000	8×8	14.5	13.0	0.10E-15	14.6	9.94	0.65E-15
2000	1×1	3934	1.00	0.16E-15	2829	1.00	0.96E-15
2000	2×2	665	5.92	0.15E-16	644	4.40	0.93E-15
2000	4×4	290	13.6	0.17E-15	235	12.0	0.99E-15
2000	8×8	96.1	40.9	0.12E-15	96.8	29.3	0.23E-14
3000	1×1	10820	1.00	0.28E-15	11260	1.00	0.11E-14
3000	2×2	3053	3.54	0.21E-16	2901	4.37	0.14E-14
3000	4×4	1015	10.7	0.16E-15	845	15.0	0.36E-14
3000	8×8	327	33.1	0.11E-15	279	45.5	0.17E-13

PDGECRNX. A general observation of the benefits of the refinement procedure is that it often improves the accuracy of the residual norm (see below) but not necessarily the forward error of the solution when compared with an exact solution (which is sometimes known for the considered benchmark examples).

A. Random problems

We generate a collection of uniformly distributed random problems using ScaLAPACK's matrix generator PDMATGEN, as follows: The matrix A is generated as a diagonal dominant matrix with random off-diagonal elements, B as a random matrix, X as a symmetric random matrix, G is computed as BB^T and Q is computed to satisfy the corresponding CARE. For the sign function iteration, G is kept in factorized form.

We also consider random problems with A as a completely random matrix (i.e., not diagonal dominant), which are expected to be more ill-conditioned.

Results are displayed in Table II.

B. Benchmarks

We consider some of the examples from the benchmark collection CAREX [7]. The matrices are generated using the associated Fortran 77 routines on one of the nodes and distributed to the whole process mesh prior to the invocation of the solvers. Unfortunately, this means that the available memory on one node dictates the limit of the problem size generated by the CAREX routines. A parallel implementation of this benchmark collection would avoid this effect, the problem size would then only be limited by the memory on the whole target machine.

TABLE III

EXPERIMENTAL RESULTS FOR PSBO2MD AND PDGECOCA SOLVING CAREX BENCHMARK PROBLEM 15 ON *sarek*.

		PSB02MD					
n	$P_r \times P_c$	T_p	S_p	R_r	I_s	I_o	I_i
1999	1×1	3427	1.00	0.32E-20	0	3	36
1999	2×2	1000	3.47	0.32E-20	0	3	36
1999	4×4	374	9.16	0.32E-20	0	3	36
1999	8×8	163	21.0	0.32E-20	0	3	36
3999	2×2	6553	1.00	0.11E-20	0	3	36
3999	4×4	1381	4.75	0.11E-20	0	3	36
3999	8×8	647	10.1	0.11E-20	0	3	39
		PDGECOCA					
n	$P_r \times P_c$	T_p	S_p	R_r	I_s	I_o	I_i
1999	1×1	1188	1.00	0.61E-20	12	3	36
1999	2×2	381	3.12	0.53E-20	12	3	36
1999	4×4	153	7.76	0.53E-20	12	3	36
1999	8×8	74.9	15.9	0.53E-20	12	3	36
3999	2×2	2818	1.00	0.19E-20	13	3	39
3999	4×4	947	2.98	0.19E-20	13	3	39
3999	8×8	382	7.38	0.19E-20	13	3	39

Example 15 in [7]. The system matrices describe a mathematical model of position and velocity for a string of high-speed vehicles; a problem also known as *smart* or *intelligent highway*. The system matrices are of order $n = 2N - 1$, where N denotes the number of vehicles. The closed loop eigenvalues are all of magnitude $O(1)$ and have a distance of at least 0.66 from the imaginary axis. Results are displayed in Table III.

Example 16 in [7]. All system matrices and the solution are circulant matrices. Most eigenvalues are of the corresponding Hamiltonian matrix have algebraic multiplicity 2 which may slow down the convergence of the multishift QR algorithm in the Schur method. All closed loop eigenvalues are real, of magnitude $O(1)$, and have at least a distance of 1 to the imaginary axis. Results are displayed in Table IV.

Optimal Cooling of Steel Profiles. Part of the Oberwolfach model reduction benchmark collection [31], the system matrices arise from the spatial discretization of a controlled $2D$ heat transfer process for optimal cooling of steel profiles. A similar example was previously used in LYAPACK [38]. The order n depends on the grid size of the FEM discretization; we have used the two coarsest discretizations available, leading to $n = 1357$ and $n = 5177$. Note that the original problem is a descriptor system, which has been turned into a standard system by the Cholesky factorization of the symmetric positive definite descriptor matrix E. Results are displayed in Table V, where we also include R_c, which is an estimate of the reciprocal of the condition number (in the 1-norm) of the Nth order system of equations from which the solution matrix X is obtained; this quantity is computed by PSB02MD only.

TABLE IV

EXPERIMENTAL RESULTS FOR PSBO2MD WITHOUT ITERATIVE REFINEMENT AND PDGECOCA WITH ITERATIVE REFINEMENTS SOLVING CAREX BENCHMARK PROBLEM 16 ON *sarek*.

		PSB02MD						
n	$P_r \times P_c$	T_p	S_p	R_r	R_e	I_s	I_o	I_i
1000	1×1	175	1.00	0.78E-16	0.11E-12	0	0	0
1000	2×2	94.0	3.46	0.79E-16	0.11E-12	0	0	0
1000	4×4	38.5	4.56	0.79E-16	0.11E-12	0	0	0
1000	8×8	14.5	12.1	0.77E-16	0.11E-12	0	0	0
2000	1×1	3806	1.00	0.60E-16	0.22E-12	0	0	0
2000	2×2	659	5.77	0.59E-16	0.22E-12	0	0	0
2000	4×4	244	15.6	0.60E-16	0.22E-12	0	0	0
2000	8×8	94.5	40.2	0.60E-16	0.22E-12	0	0	0
3000	2×2	2390	1.00	0.14E-18	0.72E-12	0	0	0
3000	4×4	768	3.11	0.33E-18	0.72E-12	0	0	0
3000	8×8	232	10.3	0.21E-18	0.72E-12	0	0	0
		PDGECOCA						
n	$P_r \times P_c$	T_p	S_p	R_r	R_e	I_s	I_o	I_i
1000	1×1	51.0	1.00	0.21E-15	0.11E-12	8	3	15
1000	2×2	17.0	2.99	0.21E-15	0.11E-12	8	3	15
1000	4×4	7.93	6.42	0.21E-15	0.11E-12	8	3	15
1000	8×8	4.65	11.0	0.21E-15	0.11E-12	8	3	15
2000	1×1	381	1.00	0.63E-15	0.27E-12	7	3	15
2000	2×2	114	3.34	0.63E-15	0.27E-12	7	3	15
2000	4×4	42.2	9.04	0.63E-15	0.27E-12	7	3	15
2000	8×8	18.5	20.6	0.63E-15	0.27E-12	7	3	15
3000	2×2	735	1.00	0.29E-18	0.72E-12	7	3	15
3000	4×4	245	3.00	0.27E-18	0.72E-12	7	3	15
3000	8×8	98.6	7.45	0.32E-18	0.72E-12	7	3	15

TABLE V

EXPERIMENTAL RESULTS FOR PSBO2MD AND PDGECOCA SOLVING THE OPTIMAL COOLING OF STEEL PROFILES PROBLEM FOR $n = 1357, 5177$ ON *sarek*.

		PSB02MD						
n	$P_r \times P_c$	T_p	S_p	R_r	R_c	I_s	I_o	I_i
1357	1×1	654	1.00	0.11E-17	0.80E-8	0	3	48
1357	2×2	303	2.16	0.82E-18	0.79E-8	0	4	64
1357	4×4	106	6.16	0.78E-18	0.79E-8	0	3	48
1357	8×8	71.6	9.13	0.86E-18	0.80E-8	0	4	64
5177	2×2		1.00			0	0	0
5177	4×4	5996		0.43E-18	0.22E-8	0	4	68
5177	8×8	2137		0.45E-18	0.22E-8	0	4	68
		PDGECOCA						
n	$P_r \times P_c$	T_p	S_p	R_r	R_c	I_s	I_o	I_i
1357	1×1	558	1.00	0.23E-17	0.80E-8	21	3	48
1357	2×2	178	3.13	0.18E-17	0.79E-8	21	3	48
1357	4×4	81.4	6.85	0.18E-17	0.79E-8	21	3	48
1357	8×8	47.4	11.8	0.19E-17	0.80E-8	21	3	48
5177	2×2		1.00			0	0	0
5177	4×4	2630		0.91E-18	0.22E-8	20	3	51
5177	8×8	933		0.88E-18	0.22E-8	20	3	51

C. Summary and profiling

In all presented examples, the sign function iteration clearly outperforms our preliminary parallel implementation of the Schur method. The difference is between a factor of 3–10 on a single node and a factor of 2–5 on 64 nodes.

These figures tell little about the potential performance of the Schur method itself, they merely illustrate how difficult it is – in comparison to the sign function iteration – to develop an efficient parallel implementation. It turns out that the parallel performance of the Schur method is highly dominated by the poor performance of the QR algorithm in ScaLAPACK, i.e., reduction of the Hamiltonian matrix from Hessenberg form to real Schur form. For example, for the considered CAREX benchmarks as much as 90–95% of the uniprocessor execution time can be due to the parallel QR algorithm alone; for multiple processors this ratio is typically less but still at least 75% of the total parallel execution time. A modern re-implementation of the parallel QR algorithm, almost entirely based on level 3 BLAS and equipped with aggressive early deflation [14], is expected to cut down this figure drastically, say, by a factor of 10 for sufficiently large matrices. This would make the Schur-based algorithm more competitive in comparison with the matrix sign function iteration regarding parallel performance.

VI. CONCLUSIONS AND FUTURE WORK

The results of our preliminary parallel implementation clearly demonstrate that there is a need for a thorough revision of the parallel QR algorithm implemented in ScaLAPACK.

The developed parallel Schur-based method can be generalized to cover the discrete-time variant of the Riccati equation (DARE) of the form

$$X = A^T X A - A^T X B (R + B^T X B)^{-1} B^T X A + Q, \quad (10)$$

as well as generalized *descriptor* variants of the CARE and DARE equations (see, e.g., [10], [20]). However, the latter problems require a reliable and efficient implementation of the parallel QZ algorithm and related eigenvalue reordering algorithms; recent progress in this direction was presented in [19], [1], [2], [3], [25] and in [30] multishift variants of the QZ algorithm with aggressive early deflation [14] was presented.

REFERENCES

[1] B. Adlerborn, K. Dackland, and B. Kågström. Parallel Two-Stage Reduction of a Regular Matrix Pair to Hessenberg-Triangular Form. In T. Sørvik and et al, editors, *Applied Parallel Computing: New Paradigms for HPC Industry and Academia*, volume 1947, pages 92–102. Springer-Verlag, Lecture Notes in Computer Science, 2001.

[2] B. Adlerborn, K. Dackland, and B. Kågström. Parallel and blocked algorithms for reduction of a regular matrix pair to Hessenberg-triangular and generalized Schur forms. In J. Fagerholm et al., editor, *Applied Parallel Computing PARA 2002*, volume 2367 of *Lecture Notes in Computer Science*, pages 319–328. Springer-Verlag, 2002.

[3] B. Adlerborn, D. Kressner, and B. Kågström. Parallel Variants of the Multishift QZ Algorithm with Advanced Deflation Techniques . In B. Kågström et al., editor, *Applied Parallel Computing - State of the Art in Scientific Computing (PARA'06)*, volume 4699, pages 117–126. Lecture Notes in Computer Science, Springer, 2007.

[4] J. M. Bada, P. Benner, R. Mayo, and E. S. Quintana-Ortí. Parallel solution of large-scale and sparse generalized algebraic Riccati equations. In W. E. Nagel, W. V. Walter, and W. Lehner, editors, *Euro-Par 2006 Parallel Processing: 12th International Euro-Par Conference*, volume 4128, pages 710–719. Springer-Verlag, 2006.

[5] P. Benner. Solving large-scale control problems. *IEEE Control Systems Magazine*, 24(1):44–59, 2004.

[6] P. Benner, R. Byers, E. S. Quintana-Ortí, and G. Quintana-Ortí. Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search. *Parallel Comput.*, 26(10):1345–1368, 2000.

[7] P. Benner, A. J. Laub, and V. Mehrmann. Benchmarks for the numerical solution of algebraic Riccati equations. *IEEE Control Systems Magazine*, 7(5):18–28, 1997.

[8] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT—a subroutine library in systems and control theory. In *Applied and computational control, signals, and circuits, Vol. 1*, pages 499–539. Birkhäuser Boston, Boston, MA, 1999.

[9] P. Benner, E. Quintana-Ort'i, and G. QuintanaOrt. A portable subroutine library for solving linear control problems on distributed memory computers, 1999.

[10] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving linear-quadratic optimal control problems on parallel computers. Preprint sfb393/05-19, TU Chemnitz, Fakultät für Mathematik, D-09107 Chemnitz (Germany), 2005. Submitted to Optimization: Methods & Software.

[11] M. W. Berry, J. J. Dongarra, and Y. Kim. A parallel algorithm for the reduction of a nonsymmetric matrix to block upper-Hessenberg form. *Parallel Comput.*, 21(8):1189–1211, 1995.

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

[13] BLACS - Basic Linear Algebra Communication Subprograms. See `http://www.netlib.org/blacs/index.html`.

[14] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm, II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2002.

[15] A. Bunse-Gerstner and H. Faßbender. A Jacobi-like method for solving algebraic Riccati equations on parallel computers. *IEEE Trans. Automat. Control*, 42(8):1071–1084, 1997.

[16] R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Linear Algebra Appl.*, 85:267–279, 1987.

[17] R. Byers. A Hamiltonian-Jacobi algorithm. *IEEE Trans. Automat. Control*, 35:566–570, 1990.

[18] J. Choi, J. J. Dongarra, and D. W. Walker. The design of a parallel dense linear algebra software library: reduction to Hessenberg, tridiagonal, and bidiagonal form. *Numer. Algorithms*, 10(3-4):379–399, 1995.

[19] K. Dackland and B. Kågström. Blocked Algorithms and Software for Reduction of a Regular Matrix Pair to Generalized Schur Form. *ACM Trans. Math. Software*, 25(4):425–454, 1999.

[20] B. N. Datta. *Numerical Methods for Linear Control Systems*. Elsevier Academic Press, San Diego, CA, 2004.

[21] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[22] GOTO-BLAS - High-Performance BLAS by Kazushige Goto. See `http://www.cs.utexas.edu/users/flame/goto/`.

[23] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part I: Theory and Algorithms. *ACM TOMS*, 2007. *submitted*.

[24] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part II: the SCASY Software. *ACM TOMS*, 2007. *submitted*.

[25] R. Granat, B. Kågström, and D. Kressner. Parallel eigenvalue reordering in real Schur forms. *Concurrency and Computation: Practice and Experience*, 2007. submitted.

[26] L. Grasedyck, W. Hackbusch, and B. N. Khoromskij. Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices. *Computing*, 70(2):121–165, 2003.

[27] M. Green and D. J. N. Limebeer. *Linear Robust Control*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[28] G. Henry, D. S. Watkins, and J. J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. *SIAM J. Sci. Comput.*, 24(1):284–311, 2002.

[29] K. Jbilou. Block Krylov subspace methods for large algebraic Riccati equations. *Numer. Algorithms*, 34(2-4):339–353, 2003. International Conference on Numerical Algorithms, Vol. II (Marrakesh, 2001).

[30] B. Kågström and D. Kressner. Multishift Variants of the QZ Algorithm with Aggressive Early Deflation. *SIAM J. Matrix Anal. Appl.*, 29(1):199–227, 2006.

[31] J.G. Korvink and E.B. Rudnyi. Oberwolfach benchmark collection. In P. Benner, V. Mehrmann, and D.C. Sorensen, editors, *Dimension Reduction of Large-Scale Systems*, volume 45 of *Lecture Notes in Computational Science and Engineering*, pages 311–315. Springer-Verlag, Berlin/Heidelberg, Germany, 2005. See .

[32] LAPACK - Linear Algebra Package. See `http://www.netlib.org/lapack/`.

[33] A. J. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Trans. Automat. Control*, AC-24:913–921, 1979.

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

[35] C. Mehl. On asymptotic convergence of nonsymmetric Jacobi algorithms. Technical report no. 393, DFG Research Center Matheon, Berlin, 2007.

[36] 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, 1991.

[37] MPI - Message Passing Interface. See `http://www-unix.mcs.anl.gov/mpi/`.

[38] T. Penzl. Lyapack users guide. Technical report SFB393/00-33, Sonderforschungsbereich 393 Numerische Simulation auf massiv parallelen Rechnern, TU Chemnitz, 09107 Chemnitz, FRG, 2000.

[39] J. D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32(4):677–687, 1980.

[40] SCASY - ScaLAPACK-style solvers for Sylvester-type matrix equations. See `http://www.cs.umu.se/~granat/scasy.html`.

[41] ScaLAPACK Users' Guide. See `http://www.netlib.org/scalapack/slug/`.

[42] A. Varga. On stochastic balancing related model reduction. 2000.

[43] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice-Hall, Upper Saddle River, NJ, 1996.