Parallel Solution of Selected Problems in Control Theory

Enrique S. Quintana-Ortí* Robert van de Geijn†

Abstract

Lyapunov and Stein matrix equations arise in many important analysis and synthesis applications in control theory. The traditional approach to solving these equations relies on the QR algorithm which is notoriously difficult to parallelize. We investigate iterative solvers based on the matrix sign function and the Smith iteration which are highly efficient on parallel distributed computers. We also show that by coding using the Parallel Linear Algebra Package (PLAPACK) it is possible to exploit structure in the matrices and reduce the cost of these solvers.

The experimental results on a Cray T3E report the high efficiency of these parallel solvers.

1 Introduction

Consider the Lyapunov matrix equation,

$$A^T X_L + X_L A + Q = 0, \tag{1}$$

and the Stein matrix equation,

$$A^T X_S A - X_S + Q = 0, \tag{2}$$

where A, Q, X_L, $X_S \in \mathbb{R}^{n \times n}$, $Q = Q^T$, and $X_L = X_L^T$, $X_S = X_S^T$ are the corresponding sought-after solutions. These matrix equations arise in many applications of control theory and stability analysis of linear control systems governed by first-order ordinary differential equations [1, 19, 23, 25, 27]. In particular, large scale equations of these types arise in stabilization of linear control systems, J-inner-outer factorizations of rational matrices, optimal linear-quadratic control problems, etc. [4, 5, 32, 33].

In order to guarantee that (1) has a unique solution, we assume that $\lambda_i + \lambda_j \neq 0$, for all λ_i, $\lambda_j \in \Lambda(A)$ (here, $\Lambda(A)$ denotes the eigenspectrum of A). The existence and uniqueness of the solution of (2) is guaranteed by assuming that $\lambda_i \cdot \lambda_j \neq 1$, for all λ_i, $\lambda_j \in \Lambda(A)$.

In particular, in control applications we are interested in the semidefinite ($Q \geq 0$) c-stable case of the Lyapunov equation ($\Lambda(A) \subset \mathbb{C}^-$, the open left half complex plane), and the semidefinite ($Q \geq 0$) d-stable case of the Stein equation ($|\lambda_i| \neq 1$, for all $\lambda_i \in \Lambda(A)$). Note that the stability conditions guarantee the existence and uniqueness of the solution for both types of equations.

*Departamento de Informática, Universidad Jaime I, 12.080-Castellón (Spain), quintana@inf.uji.es.

†Department of Computer Sciences, The University of Texas at Austin, Taylor Hall 2.124, Austin, TX 78712, rvdg@cs.utexas.edu.

Numerical QR-type solvers for Lyapunov and Stein equations include the Bartels-Stewart method, the Hessenberg-Schur method, and Hammarling's algorithm [3, 14, 17, 18]. The common initial step in all these methods is the reduction of the coefficient matrix A to real Schur form by means of the QR algorithm [15]. This is followed by a less-expensive, specialized back substitution procedure. The overall cost of these algorithms is about $31n^3$ flops (floating-point arithmetic operations).

All attempts to parallelize these algorithms for distributed-memory architectures suffer thus from the same problem: the difficulties in parallelizing the implicit double-shift QR algorithm [20]. A new multishift technique, which increases the granularity, has been recently analyzed in [21]. A different technique relies on a block Hankel distribution (see references in [20]), which improves the balancing of the computational load. Nevertheless, the parallelism and scalability of these parallel QR algorithms are still far from those of matrix factorizations, triangular linear systems solvers, etc. (see, e.g., [9, 13]).

Explicit iterative methods for solving Lyapunov and Stein matrix equations have been known for some years now [26, 28]. These algorithms have received renewed interest [6] as they rely on so-called Level-3 BLAS computations such as matrix inversions, linear system solvers, and matrix products, which are highly efficient on current high performance parallel distributed architectures. Furthermore, although these methods can not be considered as numerically stable, the numerical results obtained in practice are close to those obtained by means of QR-type solvers [7].

In this paper we show that by coding in PLAPACK [31], and using the given off-the-shelf kernels in this library, it is easy to obtain parallel iterative distributed solvers for these matrix equations. Furthermore, we also show that PLAPACK allows us to specialize our parallel kernels to exploit the symmetry in the equations, thus reducing the cost of the iterative methods.

The paper is structured as follows. In Sections 2 and 3 we briefly review the theory behind the iterative solvers for the Lyapunov and Stein equations, respectively. In Section 4, we discuss the parallel implementation of a specialized dense matrix kernel, which allows one to exploit symmetry in the problem. Performance results obtained on a Cray T3E-600 are given in Section 5, and concluding remarks follow in the final section.

2 Solving Lyapunov matrix equations with the matrix sign function

The Newton iteration for the matrix sign function

$$\begin{array}{rclrcl} A_{k+1} & = & \frac{1}{2}\left(A_k + A_k^{-1}\right), & A_0 & = & A, \\ Q_{k+1} & = & \frac{1}{2}\left(Q_k + A_k^{-T} Q_k A_k^{-1}\right), & Q_0 & = & Q, \end{array} \tag{3}$$

was proposed by Roberts as a Lyapunov solver in [26]. In case matrix A is c-stable, the iteration can be shown to converge to $A_\infty = -I_n$ (the identity matrix of order n), and $Q_\infty = 2X_L$.

Although the convergence of the Newton iteration is globally quadratic, the initial convergence may be slow. Several quasi-optimal acceleration schemes have been proposed to speed up the initial convergence (optimal schemes require complete knowledge of $\Lambda(A)$). Among these, the

determinantal scaling [10] is usually preferred because of its efficiency and simplicity. Specifically, when using the determinantal scaling, iteration (3) becomes

$$
\begin{array}{rclcrcl}
A_{k+1} & = & \frac{1}{2}\left(A_k/\gamma_k + \gamma_k A_k^{-1}\right), & & A_0 & = & A, \\
Q_{k+1} & = & \frac{1}{2}\left(Q_k/\gamma_k + \gamma_k A_k^{-T} Q_k A_k^{-1}\right), & & Q_0 & = & Q,
\end{array}
\tag{4}
$$

with $\gamma_k = |\det(A_k)|^{1/n}$. For an excellent survey on the matrix sign function and scaling schemes, see [22].

The convergence of A_k to $-I_n$ in the Newton iteration suggests the following convergence criterion

$$\|A_k + I_n\|_1 / \|A_k\|_1 < c\sqrt{\epsilon},$$

where ϵ is the machine precision and c is a small-order constant. When this criterion is satisfied, two more iterations are performed as the ultimate quadratic convergence of the Newton iteration will then ensure the maximum attainable accuracy.

The numerical solution of Lyapunov equations by means of the matrix sign function can not be considered as a numerically stable procedure; in fact, the numerical stability depends on the distance of the eigenspectrum of A to the imaginary axis. However, recent studies [7, 11] show that the matrix sign function approach, with careful shifts and scaling, can obtain numerical results which are close to those obtained by means of the numerically stable QR-type algorithms. A more detailed study of the numerical stability of the iterative solvers based on the matrix sign function and the Smith iteration is beyond the scope of this work.

The Newton iterative scheme has a cost of $6n^3$ flops per iteration. In practice, 7–10 iterations are usually enough to achieve convergence. Thus, the overall cost is around $42n^3$–$60n^3$ flops compared to $31n^3$ flops for the QR-type algorithms. Nevertheless, the higher cost of the Newton iteration is balanced by its higher efficiency on current high performance computers, and its higher degree of parallelism on parallel distributed computers.

A naive implementation of the Lyapunov equation solver requires only efficient parallel kernels for matrix inversion and matrix-matrix multiplication. Efficient parallel inversion is discussed in [29] and efficient parallel matrix-matrix multiplication in [12, 16, 30].

The cost per iteration can be reduced by exploiting the symmetry in the sequence for Q_k. As Q_0 is symmetric, all Q_k's are also symmetric and we only need to compute the upper (lower) half part of these matrices. We therefore reduce the cost of the Newton iterative scheme to $5n^3$ per iteration.

Exploiting the symmetry in matrix computations is difficult (to code and to optimize) on parallel distributed computers. In section 4 we show that by coding in PLAPACK we easily obtain parallel algorithms which exploit this characteristic and obtain maximum performance.

3 Solving Stein matrix equations via the Smith iteration

The Smith iteration for the Stein equation

$$
\begin{array}{rclcrcl}
A_{k+1} & = & A_k^2, & & A_0 & = & A, \\
Q_{k+1} & = & Q_k + A_k^T Q_k A_k, & & Q_0 & = & Q,
\end{array}
\tag{5}
$$

is described in [28]. This iteration converges, provided A is d-stable, to $A_\infty = 0$, and $Q_\infty = X_S$.

The Smith iteration has received considerable less attention than (the Newton iteration for) the matrix sign function. Only very recently this iteration has been used for solving large sparse Lyapunov equations in [24]. A first study of the parallelization of this iteration is reported in [8].

As A_k converges to 0_n, an appropriate stopping criterion is based on

$$\|A_k\|_1 < c\sqrt{\epsilon}.$$

As in the previous case, when this criterion is satisfied, we perform two more iterations to maximize the attainable accuracy.

The cost of the Smith iteration is $6n^3$ per iteration. As for the Newton iteration, this can be reduced to $5n^3$ flops by exploiting the symmetry in the Q_k's.

A naive implementation of the Stein equation solver is given in Fig. 1. This implementation does not take advantage of symmetry in Q. Taking advantage of symmetry requires a special kernel for computing $A_k^T Q_k A_k$ that only updates the lower (or upper) triangular portion of Q_k. Parallel implementation of this kernel is discussed in Section 4.

4 Parallel Implementation of $Q \leftarrow Q + A^T Q A$

In this section, we show how by exploiting symmetry in Q one can reduce the cost of this operation from $4n^3$ operations to $3n^3$ operations, thereby reducing the cost of the Newton or the Smith iterations from $6n^3$ to $5n^3$ per iteration.

Notice that $Q \leftarrow Q + A^T Q A$ can be implemented by the following steps:

1. $B \leftarrow QA$ (symmetric matrix-matrix multiplication)
2. $Q \leftarrow Q + A^T B$ which requires a matrix-matrix multiplication that only updates the lower-triangular portion of Q.

While the first operation is a standard matrix-matrix operation provided by the level-3 BLAS, and parallelized as part of parallel libraries like PLAPACK, the second is not part of those widely used kernels. Thus we must describe how to parallelize it explicitly.

In [16, 31, 30] we discuss parallel implementation of matrix-matrix multiplication as a sequence of rank-k updates or matrix-panel multiplies. The only difference for the required operation is that now a sequence of rank-k updates that affect only the lower triangular part of A is substituted: Partition

$$A = \left(\frac{A_1}{A_2}\right) \text{ and } B = \left(\frac{B_1}{B_2}\right)$$

where A_1 and B_1 are $b \times n$ submatrices. Then

$$Q = Q + A^T B = Q + \left(\begin{array}{c|c} A_1^T & A_2^T \end{array}\right)\left(\frac{B_1}{B_2}\right) = Q + A_1^T B_1 + A_2^T B_2$$

so that the operation can be performed by the steps

```
int PLA_Stein( PLA_Obj A, PLA_Obj Q )
/*
   Solve the stable Stein equation A' X A - X + Q = 0 by the Smith iteration.
   On exit, A is overwritten by intermediate results and Q is overwritten by X.
*/
{
  /* Declarations */
  PLA_Obj B = NULL,
          one  = NULL,     zero     = NULL;
  int     dummy, convergence, size;
  double  res_norm, tolerance, eps = 2.2e-16;

  /* Extract template from any of the matrices*/
  PLA_Obj_global_length( Q, &size );

  /* Create local objects */
  PLA_Matrix_create_conf_to( Q, &B );
  PLA_Create_constants_conf_to( A, NULL, &zero, &one );
  PLA_Mscalar_create_conf_to( zero, PLA_ALL_ROWS, PLA_ALL_COLS, &res_norm );

  /* constants */
  tolerance = size*sqrt(eps);

  convergence = 0;
  while ( TRUE ){
    PLA_Gemm( PLA_TRANS,    PLA_NO_TRANS, one, A, Q, zero, B );   /* B = A' Q  */
    PLA_Gemm( PLA_NO_TRANS, PLA_NO_TRANS, one, B, A, one,  Q );   /* Q = B A   */
    PLA_Gemm( PLA_NO_TRANS, PLA_NO_TRANS, one, A, A, zero, B );   /* B = A A   */
    PLA_Copy( B, A );                                             /* A = B     */

    /* Check end of iteration */
    PLA_Matrix_one_norm( A, res_norm );

    if( convergence == 2 ) break;
    else if( res_norm < tolerance ) convergence++;
  }

  PLA_Obj_free( &B );
  PLA_Obj_free( &one );              PLA_Obj_free( &zero );

  return PLA_SUCCESS;
}
```

Figure 1: Naive PLAPACK implementation of Stein equation solver using the Smith iteration.

- Partition $A = \left(\frac{A_1}{A_2}\right)$ and $B = \left(\frac{B_1}{B_2}\right)$.
- Update lower triangular part of $Q = Q + A_1^T B_1$.
- Continue recursively with updated Q, $B = B_2$ and $A = A_2$.

Parallelization of the rank-k update $Q \leftarrow Q + A_1^T B_1$ is very similar in nature to the symmetric rank-k update $C \leftarrow C - A_1 A_1^T$ used for example in a right-looking Cholesky factorization and is well-understood [12, 31]. A parallel implementation using the PLAPACK infrastructure of the presented algorithm is given in Fig. 2.

5 Experimental Results

In this section, we report the performance attained by the special kernel for computing $Q = Q + A^T B$ as well as the overall performance of the Lyapunov and Stein equation solvers.

Performance results are given for the Cray T3E-600 (300 MHz), with all computations performed in 64-bit arithmetic. The algorithms were implemented using PLAPACK Version R1.2, which performs all communication by means of MPI. We report performance measuring MFLOPS/sec./node (millions of floating point operations per second per processing node) and total MFLOPS/sec. on 16 and 32 processor configurations. For reference, the following table shows performance of matrix-matrix multiplication on a single node of the T3E-600 in MFLOPS/sec:

	`gemm`		`PLA_Gemm`	
n	$C = AB$	$C = A^T B$	$C = AB$	$C = A^T B$
500	418	418	370	338
1000	443	444	404	375
1500	425	424	418	364

The columns marked `gemm` indicate performance of a call to the 64 bit BLAS matrix-matrix multiplication kernel. The columns marked `PLA_Gemm` show performance of the parallel matrix-matrix multiplication kernels provided by PLAPACK, when executed on one node.

Since the Newton and the Smith iterations are composed of calls to multiplication and inversion of matrices, Fig. 3(a) reports performance of parallel implementations of those kernels. Notice that communication and load-balance is much more complex in the parallel matrix inversion routine and thus performance of that kernel is substantially less than that of matrix-matrix multiplication, in particular for small problems. In Fig. 3(b) we show the performance of various approaches for computing $Q = Q + A^T B$. In this figure, we report performance for a version that does not take advantage of symmetry and the discussed algorithm that does take advantage of symmetry. For both implementations, we use an operation count of n^3, which represents the *useful* operations performed by the algorithms. Thus, the performance of the version that does not take advantage of symmetry is half of what was reported for $C = A^T B$ in Fig. 3(a). Notice that there is a clear benefit to exploiting symmetry when the matrices are large.

```
int Sym_Q_plus_AtB( PLA_Obj A, PLA_Obj B, PLA_Obj Q, int nb_alg )
{
  PLA_Obj  Acur = NULL, A_1 = NULL, A_1_dup = NULL, A_1_dup_temp = NULL,
           Bcur = NULL, B_1 = NULL, B_1_dup = NULL, B_1_dup_temp = NULL,
           one = NULL;
  int size;

  PLA_Create_constants_conf_to( A, NULL, NULL, &one );

  PLA_Obj_view_all( A, &Acur );
  PLA_Obj_view_all( B, &Bcur );

  PLA_Pmvector_create_conf_to( Q, PLA_PROJ_ONTO_ROW, PLA_ALL_ROWS,
                               nb_alg, &B_1_dup );
  PLA_Pmvector_create_conf_to( Q, PLA_PROJ_ONTO_COL, PLA_ALL_COLS,
                               nb_alg, &A_1_dup );

  while( TRUE ){
    PLA_Obj_global_length( Acur, &size );
    if ( 0 == ( size = min( size, nb_alg ) ) ) break;

    PLA_Obj_horz_split_2( Acur, size,   &A_1,
                                        &Acur );
    PLA_Obj_horz_split_2( Bcur, size,   &B_1,
                                        &Bcur );

    PLA_Obj_set_orientation( A_1, PLA_PROJ_ONTO_ROW );
    PLA_Obj_set_orientation( B_1, PLA_PROJ_ONTO_ROW );

    PLA_Obj_vert_split_2( A_1_dup, size, &A_1_dup_temp, PLA_DUMMY );
    PLA_Obj_horz_split_2( B_1_dup, size, &B_1_dup_temp,
                                         PLA_DUMMY );

    PLA_Copy( A_1, A_1_dup_temp );
    PLA_Copy( B_1, B_1_dup_temp );
    Sym_AB_perform_local_part( PLA_LOWER_TRIANGULAR,
                one, A_1_dup_temp, B_1_dup_temp, one, Q );
  }

  PLA_Obj_free( &Acur );          PLA_Obj_free( &A_1 );
  PLA_Obj_free( &A_1_dup );       PLA_Obj_free( &A_1_dup_temp );
  PLA_Obj_free( &Bcur );          PLA_Obj_free( &B_1 );
  PLA_Obj_free( &B_1_dup );       PLA_Obj_free( &B_1_dup_temp );
  PLA_Obj_free( &one );
}
```

Figure 2: PLAPACK implementation of $Q \leftarrow Q + A^T B$.

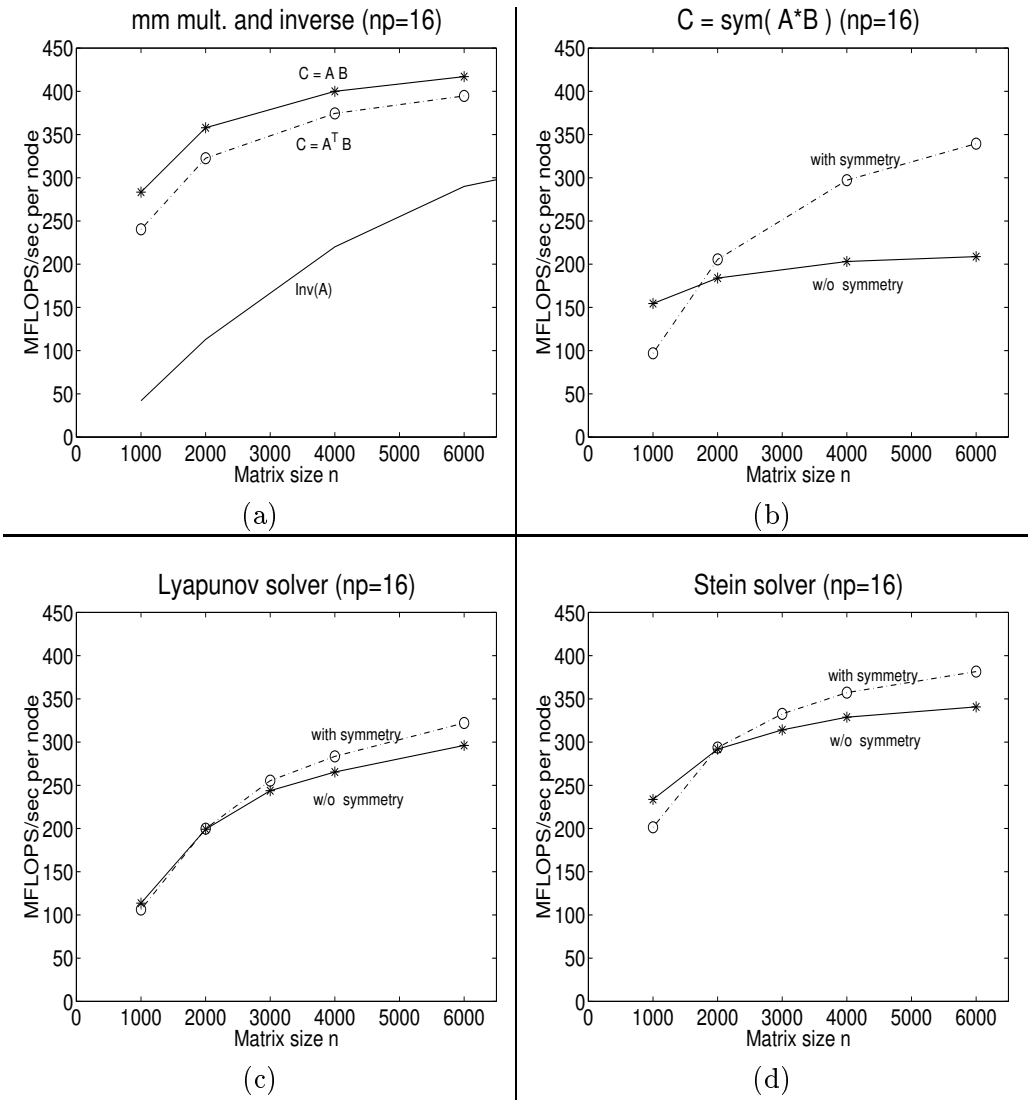

Figure 3: MFLOPS/sec. per node on the CRAY T3E-600. The curves labeled `with symmetry` and `w/o symmetry` report performance when symmetry is and is not exploited, respectively. The operation count for the Stein and Lyapunov equation solvers was taken to equal $5n^3$, the number of *useful* operations.

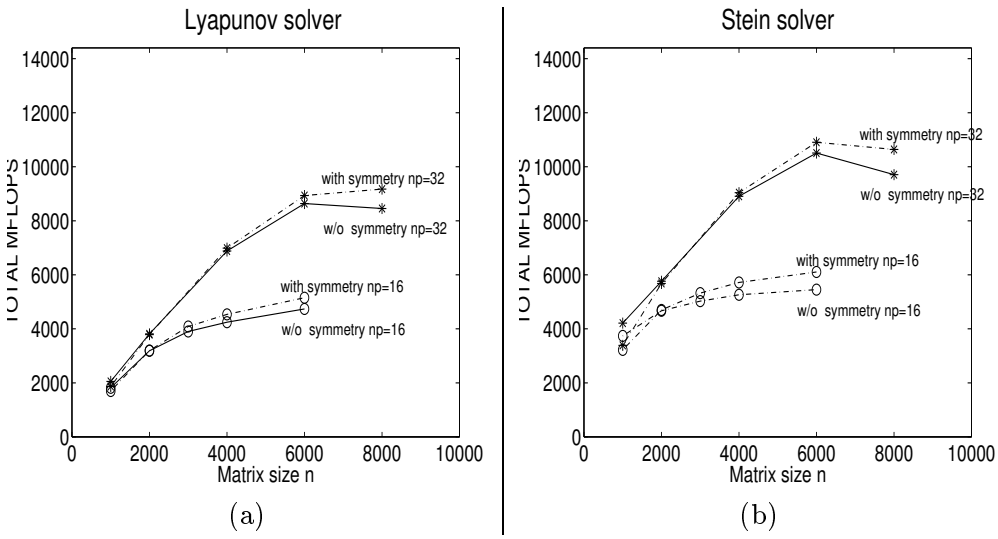

Figure 4: TOTAL MFLOPS/sec. on the CRAY T3E-600.

In Fig. 3(c) we show performance of approaches for implementing the Newton iteration for solving the Lyapunov equation. While the version that does exploit symmetry performs fewer operations, notice that the operation count is only reduced from $6n^3$ to $5n^3$ per iteration, which limits the overall benefit of exploiting symmetry.

In Fig. 3(d) we show performance of approaches for implementing the Smith iteration for solving the Stein equation. Again, the version that does exploit symmetry performs fewer operations and thus attains better performance. Overall performance is better than that attained by the Newton iteration since for the Newton iteration 40% of the computation is in the matrix inversion which does not attain the same level of performance as matrix-matrix multiplication, and is more difficult to parallelize.

In Fig. 4 we report the total performance attained by the solvers, for 16 and 32 processor configurations of the Cray T3E.

6 Concluding Remarks

The results in this paper clearly demonstrate the viability of the Newton and the Smith iterations for the solution of Lyapunov and Stein equations, respectively. In particular, we show impressive performance can be attained by parallel implementations of these algorithms.

The implementations of the solvers that do take advantage of symmetry do not perform dramatically better than those that do not take advantage of symmetry. This is in part due to the fact that only 1/3 of the total time is spent in the part of the code that benefits from symmetry. Since the kernel that takes advantage of symmetry is extremely simple to implement using PLAPACK, one might as well exploit symmetry, even though the benefits are marginal.

As mentioned, these approaches have a higher operation count than the more traditional approaches based on the QR algorithm. We reiterate that sequential implementation of the QR algorithm executes at much lower rates than the kernels used by the Newton and the Smith iterations. This can be attributed to a less favorable ratio between computational and memory operations. In addition, the QR algorithm does not parallelize well.

We plan to study a number of potential additional improvements to the algorithms. In particular, by implementing the operation $Q \leftarrow Q + A^T QA$ as a single operation there exists a potential for consolidating communication that is now performed separately within each operation, thus reducing overall overhead. Also, we have developed an algorithm for computing $Q + L^{-T}(U^{-T}QU^{-1})L^{-1}$ in parallel, updating only the lower triangular portion of Q_k in-place. This could provide an alternative approach to computing $Q + A^{-T}QA^{-1}$ in the Newton iteration. While this will clearly be less efficient than the current implementation, it may result in a slight improvement of stability of the algorithm.

Availability

The discussed implementations are available by accessing

`http://www.cs.utexas.edu/users/plapack`

Acknowledgments

Primary support for this project came from the Parallel Research on Invariant Subspace Methods (PRISM) project (ARPA grant P-95006), a grant from the Intel Research Council, the CICYT project TIC 96-1062-C03-03, and the *Consellería de Cultura Educación y Ciencia* of the *Generalitat Valenciana*.

Access to equipment was provided by the National Partnership for Advanced Computational Infrastructure (NPACI) and The University of Texas Advanced Computing Center (TACC).

References

[1] B. D. O. Anderson and J. B. Moore. *Optimal Control – Linear Quadratic Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[2] Z. Bai and J. Demmel. Using the matrix sign function to compute invariant subspaces. *SIAMMatrix*, 19(1):205–225, 1998.

[3] R.H. Bartels and G.W. Stewart. Solution of the matrix equation $AX + XB = C$: Algorithm 432. *Comm. ACM*, 15:820–826, 1972.

[4] P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems.* Dissertation, Fakultät für Mathematik, TU Chemnitz–Zwickau, 09107 Chemnitz, Germany, February 1997.

[5] P. Benner and R. Byers. An exact line search method for solving generalized continuous-time algebraic Riccati equations. *IEEE Trans. Automat. Control*, 43(1):101–107, 1998.

[6] P. Benner, J. M. Claver, and E.S. Quintana-Ortí. Parallel distributed solvers for large stable generalized Lyapunov equations. *Parallel Processing Letters*, 1998. To appear.

[7] P. Benner and E.S. Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. Technical Report SPC 97_23, Fak. f. Mathematik, TU Chemnitz, 09107 Chemnitz, FRG, 1997.

[8] P. Benner and E.S. Quintana-Ortí. Solving stable stein equations on parallel computers via smith iteration. Technical report, Fak. f. Mathematik, TU Chemnitz, 1998. in preparation.

[9] L.S. Blackford *et al. ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

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

[11] R. Byers, C. He, and V. Mehrmann. The matrix sign function method and the computation of invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 18(3):615–632, 1997.

[12] Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, and Robert A. van de Geijn. Parallel implementation of BLAS: General techniques for level 3 BLAS. *Concurrency: Practice and Experience*, to appear.

[13] J.J. Dongarra, A. Sameh, and D. Sorensen. Implementation of some concurrent algorithms for matrix factorization. *Parallel Comput.*, 3:25–34, 1986.

[14] G. H. Golub, S. Nash, and C. F. Van Loan. A Hessenberg–Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control*, AC-24:909–913, 1979.

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

[16] John Gunnels, Calvin Lin, Greg Morrow, and Robert van de Geijn. A flexible class of parallel matrix multiplication algorithms. In *Proceedings of First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (1998 IPPS/SPDP '98)*, pages 110–116, 1998.

[17] S.J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2:303–323, 1982.

[18] S.J. Hammarling. Numerical solution of the discrete-time, convergent, non-negative definite Lyapunov equation. *Sys. Control Lett.*, 17:137–139, 1991.

[19] U. Helmke and J. B. Moore. *Optimization and Dynamical Systems*. Springer-Verlag, London, GB, 1994.

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

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

[22] C. Kenney and A.J. Laub. The matrix sign function. *IEEE Trans. Automat. Control*, 40(8):1330–1348, 1995.

[23] 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.

[24] T. Penzl. A cyclic low rank Smith method for large sparse Lyapunov equations with applications in model reduction and optimal control. Technical Report SFB393/98-06, Fak. f. Mathematik, TU Chemnitz-Zwickau, 09107 Chemnitz, FRG, 1998.

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

[26] J.D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32:677–687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).

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

[28] R. A. Smith. Matrix equation $xA + Bx = C^*$. *SIAM J. Appl. Math.*, 16(1):198–201, 1968.

[29] Xiaobai Sun, Enrique S. Quintana, Gregorio Quintana, and Robert van de Geijn. Efficient matrix inversion via gauss-jordan elimination and its parallelization. Technical Report CS TR 98-19, Department of Computer Sciences, The University of Texas at Austin, 1998.

[30] Robert van de Geijn and Jerrell Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, to appear.

[31] Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package.* The MIT Press, 1997.

[32] A. Varga. Computation of Kronecker-like forms of a system pencil: Applications, algorithms and software. In *Proc. CACSD'96 Symposium, Dearborn, MI*, pages 77–82, 1996.

[33] A. Varga and T. Katayama. Computation of j–inner–outer factorizations of rational matrices. *Internat. J. Robust and Nonlinear Cont.*, 8:245–263, 1998.