Parallel ScaLAPACK-Style Algorithms for Solving Continuous-Time Sylvester Matrix Equations

Robert Granat, Bo Kågström, and Peter Poromaa

Department of Computing Science and HPC2N, Umeå University,
SE-901 87 Umeå, Sweden.
{granat,bokg,peterp}@cs.umu.se

Abstract. An implementation of a parallel ScaLAPACK-style solver for the general Sylvester equation, $\mathrm{op}(A)X - X\mathrm{op}(B) = C$, where $op(A)$ denotes A or its transpose A^T, is presented. The parallel algorithm is based on explicit blocking of the Bartels-Stewart method. An initial transformation of the coefficient matrices A and B to Schur form leads to a reduced triangular matrix equation. We use different matrix traversing strategies to handle the transposes in the problem to solve, leading to different new parallel wave-front algorithms. We also present a strategy to handle the problem when 2 x 2 diagonal blocks of the matrices in Schur form, corresponding to complex conjugate pairs of eigenvalues, are split between several blocks in the block partitioned matrices. Finally, the solution of the reduced matrix equation is transformed back to the originally coordinate system. The implementation acts in a ScaLAPACK environment using 2-dimensional block cyclic mapping of the matrices onto a rectangular grid of processes. Real performance results are presented which verify that our parallel algorithms are reliable and scalable.

Keywords: Sylvester matrix equation, continuous-time, Bartels–Stewart method, blocking, GEMM-based, level 3 BLAS, SLICOT, ScaLAPACK-style algorithms.

1 Introduction

We present a parallel ScaLAPACK-style solver for the Sylvester equation (SYCT)

$$\mathrm{op}(A)X - X\mathrm{op}(B) = C, \tag{1}$$

where $\mathrm{op}(A)$ denotes A or its transpose A^T. Here A of size $M \times M$, B of size $N \times N$ and C of size $M \times N$ are arbitrary matrices with real entries. Equation (1) has a unique solution X of size $M \times N$ if and only if $\mathrm{op}(A)$ and $\mathrm{op}(B)$ have disjoint spectra. The Sylvester equation appears naturally in several applications. Examples include block-diagonalizing of a matrix in Schur form and condition estimation of eigenvalue problems (e.g., see [15, 10, 16]).

Our method for solving SYCT (1) is based on the Bartels–Stewart method [1]:

1. Transform A and B to upper (quasi)triangular form T_A and T_B, respectively, using orthogonal similarity transformations:
$$Q^T AQ = T_A, \qquad P^T BP = T_B.$$
2. Update the matrix C with respect to the transformations done on A and B:
$$\widetilde{C} = Q^T CP.$$
3. Solve the reduced (quasi)triangular matrix equation:
$$\mathrm{op}(T_A)\widetilde{X} - \widetilde{X}\mathrm{op}(T_B) = \widetilde{C}.$$
4. Transform the solution $\widetilde{X}$ back to the original coordinate system:
$$X = Q\widetilde{X}P^T.$$

The quasitriangular form mentioned in Step 1 is also called the real Schur form, which means that the matrix is upper block triangular with 1×1 and 2×2 diagonal blocks, corresponding to real and complex conjugate pairs of eigenvalues, respectively. To carry out Step 1 we use the QR-algorithm [2]. The updates in Step 2 and the back-transformation in Step 4 are carried out using ordinary GEMM-operations $C \leftarrow \beta C + \alpha \mathrm{op}(A)\mathrm{op}(B)$, where α and β are scalars [5, 13, 14].

Our focus is on Step 3. Using the Kronecker product notation, $\otimes$, we can rewrite the triangular Sylvester equation as a linear system of equations
$$Zx = y, \tag{2}$$
where $Z = I_N \otimes \mathrm{op}(A) - \mathrm{op}(B)^T \otimes I_M$ is a matrix of size $MN \times MN$, $x = \mathrm{vec}(X)$ and $y = \mathrm{vec}(C)$. As usual, $\mathrm{vec}(X)$ denotes an ordered stack of the columns of the matrix X from left to right starting with the first column. The linear system (2) can be solved to the cost of $O(M^3N^3)$ using ordinary LU factorization with pivoting. This is a very expensive operation, even for moderate-sized problems. Since A and B are (quasi)triangular, the triangular Sylvester equation can indeed be solved to the cost $O(M^2N + MN^2)$ using a combined backward/forward substitution process [1]. In blocked algorithms, the explicit Kronecker matrix representation $Zx = y$ is used in kernels for solving small-sized matrix equations (e.g., see [11, 12, 15]).

The rest of the paper is organized as follows: In Section 2, we give a brief overview of blocked algorithms for solving the triangular SYCT equation. Section 3 is devoted to parallel algorithms focusing on the solution of the reduced triangular matrix equations. Finally, in Section 4, we present experimental results and discuss the performance of our general ScaLAPACK-style solver.

Our parallel implementations mainly adopt to the ScaLAPACK software conventions [3]. The P processors (or virtual processes) are viewed as a rectangular processor grid $P_r \times P_c$, with $P_r \geq 1$ processor rows and $P_c \geq 1$ processor columns such that $P = P_r \cdot P_c$. The data layout of dense matrices on a rectangular grid is assumed to be done by the two-dimensional (2D) block-cyclic distribution scheme.

2 Blocked Algorithms

Blocking is a powerful tool in Numerical Linear Algebra to restructure well-known standard algorithms in level 3 operations with the potential to reuse data already stored in cache or registers. This will make things faster and more efficient on one processor or in a shared memory environment. Blocking is also useful for parallelizing tasks in distributed memory environments.

2.1 The Non-transposed Case

We start by reviewing the serial block algorithm proposed in [15] for the non-transposed triangular Sylvester equation

$$AX - XB = C. \tag{3}$$

Here A and B have already been transformed to real Schur form. Let MB and NB be the block sizes used in the partitioning of A and B, respectively. Then MB is the row-block size and NB is the column-block size of C and X (which overwrites C). Now, the number of diagonal blocks of A and B can be expressed as $D_a = \lceil M/MB \rceil$ and $D_b = \lceil N/NB \rceil$, respectively. Then Equation (3) can be rewritten in block-partitioned form:

$$A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij} - (\sum_{k=i+1}^{D_a} A_{ik}X_{kj} - \sum_{k=1}^{j-1} X_{ik}B_{kj}), \tag{4}$$

where $i = 1, 2, \ldots, D_a$ and $j = 1, 2, \ldots, D_b$. Based on this summation formula, a serial blocked algorithm can be formulated, see Figure 1.

---

for j=1, D_b
    for i=D_a, 1, -1
        {Solve the (i, j)th subsystem}
        $A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij}$
        for k=1, $i - 1$
            {Update block column j of C}
            $C_{kj} = C_{kj} - A_{ki}X_{ij}$
        end
        for k=$j + 1$, D_b
            {Update block row i of C}
            $C_{ik} = C_{ik} + X_{ij}B_{jk}$
        end
    end
end

Fig. 1. Block algorithm for solving $AX - XB = C$, A and B in upper Schur form.

2.2 The Transposed Cases

The three other cases, namely,

$$AX - XB^T = C, \qquad A^T X - XB = C, \qquad \text{and} \;\; A^T X - XB^T = C,$$

can be treated in the same way. Each of these matrix equations correspond to a summation formula based on the same block partitioning of the matrices:

$$A_{ii}X_{ij} - X_{ij}B_{jj}^T = C_{ij} - (\sum_{k=i+1}^{D_a} A_{ik}X_{kj} - \sum_{k=j+1}^{D_b} X_{ik}B_{jk}^T),$$

$$A_{ii}^T X_{ij} - X_{ij}B_{jj} = C_{ij} - (\sum_{k=1}^{i-1} A_{ki}^T X_{kj} - \sum_{k=1}^{j-1} X_{ik}B_{kj}),$$

$$A_{ii}^T X_{ij} - X_{ij}B_{jj}^T = C_{ij} - (\sum_{k=1}^{i-1} A_{ki}^T X_{kj} - \sum_{k=j+1}^{D_b} X_{ik}B_{jk}^T).$$

For each of these summation formulas a serial block algorithm is formulated. In Figure 2, we present the one corresponding to the reduced triangular equation $A^T X - XB^T = C$.

---

for j= D_b, 1, -1
    for i=1, D_a
        {Solve the $(i, j)th$ subsystem}
        $A_{ii}^T X_{ij} - X_{ij}B_{jj}^T = C_{ij}$
        for k=$i+1$, D_a
            {Update block column j of C}
            $C_{kj} = C_{kj} - A_{ik}^T X_{ij}$
        end
        for k=1, $j-1$
            {Update block row i of C}
            $C_{ik} = C_{ik} + X_{ij}B_{kj}^T$
        end
    end
end

Fig. 2. Block algorithm for solving $A^T X - XB^T = C$, A and B in upper Schur form.

Notice that each summation formula sets a starting point in the matrix C/X where we start to compute the solution. For example, while solving subsystems and updating C/X with respect to these subsolutions in Figure 1, we traverse the matrix C/X along its block diagonals from South-East to North-West (or vice versa). This "wavefront" starts in the South-West corner of C/X, as depicted in Figure 3, and moves in the North-Eastern direction. Along the way, each computed X_{ij} will be used to update block-row i and block-column j of C.

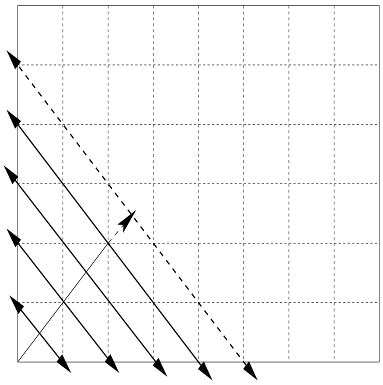

Fig. 3. Traversing the matrix C/X when solving $AX - XB = C$.

3 Parallel Block Algorithms

We assume that the matrices A, B and C are distributed using 2D block-cyclic mapping across a $P_r \times P_c$ processor grid. To carry out Steps 1, 2 and 4 of the Bartels–Stewart method in parallel we use the ScaLAPACK library-routines `PDGEHRD`, `PDLAHQR` and `PDGEMM` [3]. The first two routines are used in Step 1 to compute the Schur decompositions of A and B (reduction to upper Hessenberg form followed by the parallel QR algorithm [9, 8]). `PDGEMM` is the parallel implementation of the level 3 BLAS DGEMM operation and is used in Steps 2 and 4 for doing the two-sided matrix multiply updates.

To carry out Step 3 in parallel, we traverse the matrix C/X along its block diagonals, starting in the corner that is decided by the data dependencies. To be able to compute X_{ij} for certain values of i and j, we need A_{ii} and B_{jj} to be owned by the same process that owns C_{ij}. We also need to have the blocks used in the updates in the right place at the right time. The situation is illustrated in Figure 4, where all the data lie on the right processors. This will however not be the general case.

In general, we have to communicate for some blocks during the solves and updates. For example, while traversing C/X for the triangular $AX - XB^T = C$, we solve the small subsystems

$$A_{ii}X_{ij} - X_{ij}B_{jj}^T = C_{ij}, \tag{5}$$

associated with the current block diagonal in parallel. Then we do the GEMM-updates,

$$\begin{cases} C_{kj} = C_{kj} - A_{ki}X_{ij}, & k = 1, \ldots, j-1 \\ C_{ik} = C_{ik} + X_{ij}B_{kj}^T, & k = 1, \ldots, i-1, \end{cases} \tag{6}$$

of block-row i and block-column j, which can also be done in parallel. In equation (6), the submatrix X_{ij} has been computed in the preceeding step and broadcasted (see Figure 4) to the processors involved in the GEMM-updates. It can be

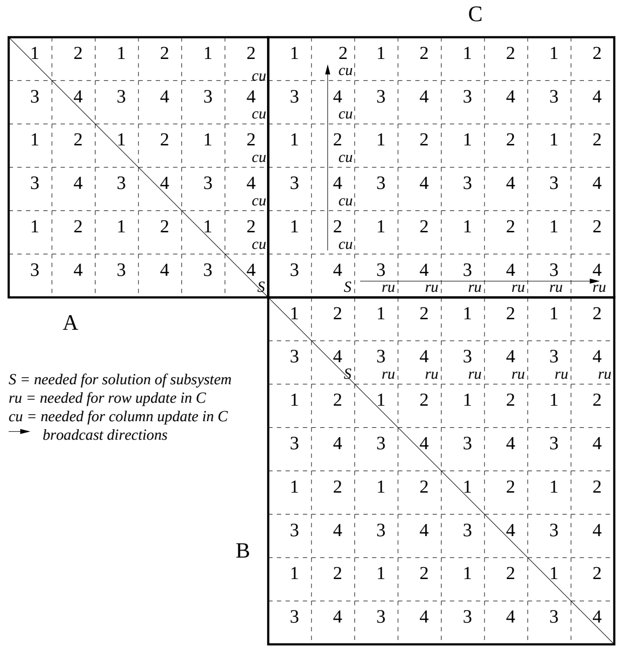

Fig. 4. Data dependencies and mapping for $AX - XB = C$ on a 2×2 processor grid.

shown that this gives a theoretical limit for the speedup of the triangular solver as $\max(P_r, P_c)$ for solving the subsystems, and as $P_r \cdot P_c$ for the GEMM-updates [7, 16]. A high-level parallel block algorithm for the solving the general triangular SYCT equation (1) is presented in Figure 5.

3.1 The 2×2 diagonal block split problem

When entering the triangular solver we already have transformed A, B and C distributed across the processor grid (2D block cyclic mapping). Now, we have to assure that no 2×2 diagonal block of A or B in Schur form, corresponding to conjugate pairs of complex eigenvalues is being split between two blocks (processors). This happens when any element on the first subdiagonal of the real Schur form is not equal to zero and does not belong to the same block (submatrix) as the closest elements to the North, East and North-East. We solve this problem by extending one block in the real Schur form of the matrix such that the lost element is included in that block. At the same time we have to diminish some other neighboring blocks. An explicit redistribution of the matrices would cause to much overhead. Instead, we do the redistribution *implicitly*, that is, we only exchange elements in one row and one column, which are stored in local workarrays. Somehow we must keep track of the extensions/reductions done. As we can see, they are completely determined by the looks of A_{ii} and B_{jj}. Therefore, we can use two 1D-arrays, call them INFO_ARRAY_A and INFO_ARRAY_B

for k=1, # block diagonals in C
  {Solve subsystems on current block diagonal in parallel}
  if($mynode$ holds C_{ij})
    if($mynode$ does not hold A_{ii} and/or B_{jj})
      Communicate for A_{ii} and/or B_{jj}
    Solve for X_{ij} in $op(A_{ii})X_{ij} - X_{ij}op(B_{ij}) = C_{ij}$
    Broadcast X_{ij} to processors that need X_{ij} for updates
  elseif($mynode$ needs X_{ij})
    Receive X_{ij}
    if($mynode$ does not hold needed block in A for updating block column j)
      Communicate for requested block in A
    Update block column j of C in parallel
    if($mynode$ does not hold needed block in B for updating block row i)
      Communicate for requested block in B
    Update block row i of C in parallel
  endif
end

Fig. 5. Parallel block algorithm for $\mathrm{op}(A)X - X\mathrm{op}(B) = C$, A and B in Schur form.

of length $\lceil M/MB \rceil$ and $\lceil N/NB \rceil$, respectively, which store information of the extensions as integer values as follows:

$$INFO\_ARRAY\_A(i) = \begin{cases} 0 \text{ if } A_{ii} \text{ is unchanged} \\ 1 \text{ if } A_{ii} \text{ is extended} \\ 2 \text{ if } A_{ii} \text{ is diminished} \\ 3 \text{ if } A_{ii} \text{ is extended } \textit{and} \text{ diminished} \end{cases}$$

The first thing to do in our triangular solver is traversing the first subdiagonal of A and B and assigning values to their INFO_ARRAY:s. Here, we are forced to do some broadcasts since the information must be global, but since this is an $O(\lceil M/MB \rceil)$ or $O(\lceil N/NB \rceil)$ operation they only effect the overall performance marginally. Then, using the data in the global arrays, we can carry out an implicit redistribution by exchanging data between the processors and build up local arrays of double precision numbers holding the extra rows and/or columns for each block of A, B and C. These local arrays can then be used to form the "correct" submatrices for our solves and updates in the parallel triangular solver.

4 Performance Results and Analysis

We present measured performance results of our ScaLAPACK-style algorithms using up to 64 processors on the IBM Scalable POWERparallel (SP) system at High Performance Computing Center North (HPC2N). A theoretical scalability analysis ongoing work and is not included in this paper.

We vary $P = P_r \cdot P_c$ between 1 and 64 in multiples of 2. Speedup S_p and efficiency E_p are computed with respect to the run for the current problem size that we were able to solve with as few processors as possible. Therefore, the results for the speedup and efficiency must be understood from the context. All timings are performed on random generated problems which typically are pretty ill-conditioned and have large-normed solutions X.

In Table 1, we present performance results for the triangular solver `PDTRSY` when solving $AX - XB = C$ and $AX - XB^T = C$, and A and B are in upper real Schur form.

In Table 2, we present performance results for the general solver `PDGESY` when solving $AX - XB^T = C$. Here, the timings include all four steps. Moreover, we display the number of 2×2 diagonal split problems that were involved and the absolute residual of the solutions. The sizes of the residuals are due to the fact that the random problems are rather ill-conditioned, i.e., the separation between A and B are quite small resulting in near to singular systems to solve.

Finally, in Table 3, we present the execution profile of the results of Table 2. As expected, it is the transformations to Schur form in Step 1 that dominate the execution time. However, it is still important to have a scalable and efficient solver for the triangular SYCT equations, since in condition estimation we typically have to call `PDTRSY` several (about five) times [15, 11, 12].

$M = N$	MB	P_r	P_l	Time (sec.)		S_p		E_p	
				A, B	A, B^T	A, B	A, B^T	A, B	A, B^T
1024	64	1	1	58	46	1.0	1.0	1.00	1.00
1024	64	2	1	14	13	4.1	3.4	2.06	1.68
1024	64	2	2	7.8	9.4	7.5	4.9	1.87	1.22
1024	64	2	4	7.2	8.2	8.1	5.6	1.01	0.70
1024	64	4	4	7.3	6.0	8.0	7.7	0.50	0.48
2048	64	2	2	133	143	1.0	1.0	1.00	1.00
2048	64	4	2	39	59	3.4	2.4	1.69	1.21
2048	64	4	4	30	32	4.4	4.5	1.11	1.12
2048	64	8	4	25	28	5.3	5.1	0.67	0.64
2048	64	8	8	22	20	6.0	7.2	0.38	0.45
4096	64	4	4	281	301	1.0	1.0	1.00	1.00
4096	64	8	4	168	188	1.4	1.6	0.84	0.80
4096	64	8	8	117	97	2.4	3.1	0.60	0.78

Table 1. Performance of `PDTRSY` solving $AX - XB = C$ and $AX - XB^T = C$.

Our software is designed for integration in state-of-the-art software libraries such as ScaLAPACK [3] and SLICOT [17, 6].

Acknowledgements

This research was conducted using the resources of the High Performance Computing Center North (HPC2N).

$M=N$	MB	P_r	P_l	Time (sec.)	S_p	E_p	#Ext	Abs. residual
1024	64	1	1	696	1.0	1.00	179	$0.10E-10$
1024	64	2	1	397	1.7	0.85	160	$0.98E-11$
1024	64	2	2	260	2.7	0.68	160	$0.96E-11$
1024	64	4	2	183	3.8	0.48	148	$0.89E-11$
1024	64	4	4	140	5.0	0.31	146	$0.89E-11$
2048	64	2	2	2057	1.0	1.00	663	$0.27E-10$
2048	64	4	2	1061	1.9	0.97	664	$0.26E-10$
2048	64	4	4	553	3.7	0.93	704	$0.26E-10$
2048	64	4	8	384	5.4	0.67	604	$0.24E-10$
2048	64	8	8	364	5.7	0.35	663	$0.24E-10$
4096	64	4	4	5158	1.0	1.00	3400	$0.72E-10$
4096	64	8	4	2407	2.1	1.10	3376	$0.67E-10$
4096	64	8	8	1478	3.5	0.87	3360	$0.68E-10$

Table 2. Performance results of `PDGESY` solving $AX - XB^T = C$.

$M=N$	MB	P_r	P_c	Step 1 (%)	Steps 2+4 (%)	Step 3 (%)	Total time (sec.)
1024	64	1	1	83	12	4	696
1024	64	2	1	90	6	3	397
1024	64	2	2	92	4	4	260
1024	64	4	2	89	5	4	183
1024	64	4	4	89	2	4	140
2048	64	2	2	81	12	7	2057
2048	64	4	2	85	6	9	1061
2048	64	4	4	90	3	6	553
2048	64	4	8	87	4	7	384
2048	64	8	8	86	3	5	364
4096	64	4	4	75	17	8	5158
4096	64	8	4	79	12	8	2407
4096	64	8	8	89	4	7	1478

Table 3. Execution time profile of `PDGESY` solving $AX - XB^T = C$.

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.

References

1. R.H. Bartels and G.W. Stewart Algorithm 432: Solution of the Equation $AX + XB = C$, *Comm. ACM*, 15(9):820–826.
2. E. Anderson, Z. Bai, C. Bischof. J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenny, S. Ostrouchov and D. Sorensen. *LAPACK User's Guide*. Third Edition. SIAM Publications, 1999.

3. S. Blackford, J. Choi, A. Clearly, 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 Publications, Philadelphia, 1997.
4. K. Dackland and B. Kågström. An Hierarchical Approach for Performance Analysis of ScaLAPACK-based Routines Using the Distributed Linear Algebra Machine. In Wasniewski et.al., editors, *Applied Parallel Computing in Industrial Computation and Optimization, PARA96*, Lecture Notes in Computer Science, Springer, Vol. 1184, pages 187–195, 1996.
5. J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Soft.*, 16(1):1–17, 1990.
6. E. Elmroth, P. Johansson, B. Kågström, and D. Kreissner, A Web Computing Environment for the SLICOT Library, In P. Van Dooren and S. Van Huffel, *The Third NICONET Workshop on Numerical Control Software*, pp 53–61, 2001.
7. R. Granat, A Parallel ScaLAPACK-style Sylvester Solver, *Master Thesis*, UMNAD 435/03, Dept. Computing Science, Umeå University, Sweden, January, 2003.
8. 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.
9. G. Henry, D. Watkins, and J. Dongarra, J. A Parallel Implementation of the Nonsymmetric QR Algorithm for Distributed Memory Architectures. Technical Report CS-97-352 and Lapack Working Note 121, University of Tennessee, 1997.
10. N.J. Higham. Perturbation Theory and Backward Error for $AX - XB = C$, *BIT*, 33:124–136, 1993.
11. I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part I: One-Sided and Coupled Sylvester-Type Equations, *ACM Trans. Math. Software*, Vol. 28, No. 4, pp 393–415, 2002.
12. I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part II: Two-Sided and Generalized Sylvester and Lyapunov Equations, *ACM Trans. Math. Software*, Vol. 28, No. 4, pp 416–435, 2002.
13. B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Software*, 24(3):268–302, 1998.
14. B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: Portability and optimization issues. *ACM Trans. Math. Software*, 24(3):303–316, 1998.
15. B. Kågström and P. Poromaa. Distributed and shared memory block algorithms for the triangular Sylvester equation with Sep^{-1} estimators, *SIAM J. Matrix Anal. Appl.*, 13 (1992), pp. 99–101.
16. P. Poromaa. Parallel Algorithms for Triangular Sylvester Equations: Design, Scheduling and Scalability Issues. In Kågström et al. (eds), *Applied Parallel Computing. Large Scale Scientific and Industrial Problems*, Lecture Notes in Computer Science, Vol. 1541, pp 438–446, Springer-Verlag, 1998.
17. SLICOT library in the Numerics in Control Network (NICONET) website: www.win.tue.nl/niconet/index.html