Combining Explicit and Recursive Blocking for Solving Triangular Sylvester-Type Matrix Equations on Distributed Memory Platforms

Robert Granat, Isak Jonsson and Bo Kågström

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

Abstract. Parallel ScaLAPACK-style hybrid algorithms for solving the triangular continuous-time Sylvester (SYCT) equation $AX - XB = C$ using recursive blocked node solvers from the novel high-performance library RECSY are presented. We compare our new hybrid algorithms with parallel implementations based on the SYCT solver DTRSYL from LAPACK. Experiments show that the RECSY solvers can significantly improve on the serial as well as on the parallel performance if the problem data is partitioned and distributed in an appropriate way. Examples include cutting down the execution time by 47% and 34% when solving large-scale problems using two different communication schemes in the parallel algorithm and distributing the matrices with blocking factors four times larger than normally. The recursive blocking is automatic for solving subsystems of the global explicit blocked algorithm on the nodes.

Keywords: Sylvester matrix equation, continuous-time, Bartels–Stewart method, blocking, GEMM-based, level 3 BLAS, LAPACK, ScaLAPACK-style algorithms, RECSY, recursive algorithms, automatic blocking

1 Introduction

This contribution deals with parallel algorithms and software for the numerical solution of the triangular continuous-time Sylvester equation (SYCT)

$$AX - XB = C, \tag{1}$$

on distributed memory (DM) environments, where 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. The matrices A and B are in upper (quasi-)triangular Schur form. A quasi-triangular matrix is upper triangular with some 2×2 blocks on the diagonal that correspond to complex conjugate pairs of eigenvalues. SYCT has a unique solution X of size $m \times n$ if and only if A and B have disjoint spectra, or equivalently the separation $\text{sep}(A, B) \neq 0$. The Sylvester equation appears naturally in several applications. Examples include block-diagonalization of a matrix in Schur form and condition estimation of eigenvalue problems (e.g., see [17, 10, 19]).

Using the Kronecker product notation, $\otimes$, we can rewrite the Sylvester equation as a linear system of equations

$$Z_{\mathrm{SYCT}} x = c, \tag{2}$$

where $Z_{\mathrm{SYCT}} = I_n \otimes A - B^T \otimes I_m$ is a matrix of size $mn \times mn$, $x = \mathrm{vec}(X)$ and $c = \mathrm{vec}(C)$. 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. Since A and B are (quasi-)triangular, the triangular Sylvester equation can be solved to the cost $O(m^2 n + mn^2)$ using a combined backward/forward substitution process [1]. In blocked algorithms, the explicit Kronecker matrix representation $Zx = c$ is used in kernels for solving small-sized matrix equations (e.g., see [11, 12, 17]).

Our objective is to investigate the performance of our ScaLAPACK-style algorithms for solving SYCT [7, 6] when combined with recursive blocked matrix equation solvers from the recently developed high-performance library RECSY [11–13].

The rest of the paper is organized as follows; In Section 2, we review our ScaLAPACK-style algorithms for solving SYCT. Then the RECSY library, which is used for building the hybrid algorithms, is briefly presented in Section 3. In Section 4, we display and compare some experimental results of the standard and hybrid ScaLAPACK-style algorithms, respectively. Finally, in Section 5, we summarize our findings and outline ongoing and future work.

2 Parallel ScaLAPACK-style algorithms for solving SYCT using explicit blocking

To solve SYCT we transform it to triangular form, following the Bartels–Stewart method [1], before applying a direct solver. This is done by means of a Hessenberg reduction, followed by the QR-algorithm applied to both A and B. The right hand side C must also be transformed with respect to the Schur decomposition of A and B. Reliable and efficient algorithms for the reduction step can be found in LAPACK [2], for the serial case, and in ScaLAPACK [9, 8, 3] for distributed memory environments. Assuming that this reduction step has already been performed, we partition the matrices A and B in SYCT using the blocking factors mb and nb, respectively. This implies that mb is the row-block size and nb is the column-block size of the matrices C and X (which overwrites C). By defining $D_a = \lceil m/mb \rceil$ and $D_b = \lceil n/nb \rceil$, SYCT can be rewritten in blocked form as

$$A_{ii} X_{ij} - X_{ij} B_{jj} = C_{ij} - (\sum_{k=i+1}^{D_a} A_{ik} X_{kj} - \sum_{k=1}^{j-1} X_{ik} B_{kj}), \tag{3}$$

where $i = 1, 2, \ldots, D_a$ and $j = 1, 2, \ldots, D_b$. The resulting serial blocked algorithm is outlined in Figure 1 [17, 19].

We now assume that the matrices A, B and C are distributed using 2D block-cyclic mapping across a $P_r \times P_c$ processor grid. We then traverse the matrix C/X along its block diagonals from South-West to North-East, starting in the South-West corner. To be able to compute X_{ij} for different values of i and j, we need A_{ii} and B_{jj} to be held by the same process that holds C_{ij}. We also need to have the blocks used in the general matrix-multiply and add (GEMM) updates of C_{ij} in the right place at the right time. In general, this means we have to communicate for some blocks during the solves and updates. This can be done “on demand” [7]. A brief outline of

```
for j=1, D_b
    for i=D_a, 1, -1
        {Solve the (i, j)th subsystem using a kernel solver}
        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 real Schur form.

a parallel algorithm `PTRSYCTD` that uses this approach is presented in Figure 2. The matrices can also be shifted one step across the process mesh for every block diagonal that we solve for [19, 6]. This brings all the blocks needed for the solves and updates associated with the current block diagonal into the right place in one single global communication operation. A brief outline of such a parallel algorithm is presented in Figure 3. The "matrix-shifting" approach puts a restriction on the data distribution: the last rows/columns of A and B must be mapped onto the last process row/column [19]. Both communication schemes have been implemented in the same routine `PGESYCTD` [7, 6], which can solve four variants of SYCT with one or both of A and B replaced by their transposes.

The parallel algorithms presented in Figures 2 and 3 both tend to give speedup of $O(\sqrt{p})$, where p is the number of processors used in the parallel execution [19, 6, 7].

```
for k=1, the number of 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 A_ii X_ij − X_ij 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 block in A needed for updating block column j)
        Communicate for requested block in A
      Update block column j of C in parallel
      if(mynode does not hold block in B needed for updating block row i)
        Communicate for requested block in B
      Update block row i of C in parallel
    endif
end
```

Fig. 2. Parallel "communicate-on-demand" block algorithm for $AX - XB = C$, A and B in real Schur form.

Notice that we are free to choose any kernel solver for the subsystems $A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij}$ in the algorithms presented in Figures 1, 2 and 3. Here A_{ii} and B_{jj} are of size $mb \times mb$ and $nb \times nb$, respectively, and C/X is of size $mb \times nb$. The

```
for k=1, number of block diagonals in C
    if(m = n) then
        if(P_c ≠ 1) Shift A East
        if(P_r ≠ 1) Shift B North
    elseif(m < n) then
            Shift A South-East
            if(P_r ≠ 1) Shift C South
    else
            Shift B North-West
            if(P_c ≠ 1) Shift C West
    endif
    {Solve subsystems on current block diagonal in parallel}
    if(mynode holds C_ij)
            Solve for X_ij in A_ii X_ij − X_ij B_ij = C_ij
            Broadcast X_ij to processors that need X_ij for updates
    elseif(mynode needs X_ij)
            Receive X_ij
            Update block column j of C in parallel
            Update block row i of C in parallel
    endif
end
```

Fig. 3. Parallel "matrix-shifting" block algorithm for $AX - XB = C$, A and B in real Schur form.

original implementation of the parallel algorithms used LAPACK's DTRSYL as node solver, which is essentially a level-2 BLAS algorithm. For more information about the ScaLAPACK-style algorithms we refer to [19, 7, 6].

3 RECSY—using recursive blocked algorithms for solving Sylvester-type subsystems

RECSY [13] is a high-performance library for solving triangular Sylvester-type matrix equations, based on recursive blocked algorithms, which are rich in GEMM-operations [4, 15, 16]. The recursive blocking is automatic and has the potential of matching the memory hierarchies of today's high-perfromance computing systems. RECSY comprises a set of Fortran 90 routines, all equipped with Fortran 77 interfaces and LAPACK/SLICOT wrappers, which solve 42 transpose and sign variants of eight common Sylvester-type matrix equations. Table 1 lists the standard variants of these matrix equations.

Table 1. The Sylvester-type matrix equations considered in the RECSY library. CT and DT denote the continuous-time and discrete-time variants, respectively.

Name	Matrix Equation
Standard Sylvester (CT)	$AX - XB = C$
Standard Lyapunov (CT)	$AX + XA^T = C$
Standard Sylvester (DT)	$AXB^T - X = C$
Standard Lyapunov (DT)	$AXA^T - X = C$
Generalized Coupled Sylvester	$(AX - YB, DX - YE) = (C, F)$
Generalized Sylvester	$AXB^T - CXD^T = E$
Generalized Lyapunov (CT)	$AXE^T + EXA^T = C$
Generalized Lyapunov (DT)	$AXA^T - EXE^T = C$

Depending on the sizes of m and n, three alternatives for doing a *recursive splitting* are considered [11, 13]. In Case 1 ($1 \le n \le m/2$), A is split by rows and columns, and C by rows only. Similarly, in Case 2 ($1 \le m \le n/2$), B is split by rows and columns, and C by columns only. Finally, in Case 3 ($n/2 < m < 2n$) both rows and columns of the matrices A, B and C are split:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} - \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

This recursive splitting results in the following four triangular SYCT equations:

$$\begin{aligned} A_{11}X_{11} - X_{11}B_{11} &= C_{11} - A_{12}X_{21}, \\ A_{11}X_{12} - X_{12}B_{22} &= C_{12} - A_{12}X_{22} + X_{11}B_{12}, \\ A_{22}X_{21} - X_{21}B_{11} &= C_{21}, \\ A_{22}X_{22} - X_{22}B_{22} &= C_{22} + X_{21}B_{12}. \end{aligned}$$

First, X_{21} is solved for in the third equation. After updating C_{11} and C_{22} with respect to X_{21}, one can solve for X_{11} and X_{22}. Both updates and the triangular Sylvester solves are independent operations and can be executed concurrently. Finally, one updates C_{12} with respect to X_{11} and X_{22}, and solves for X_{12}. In practice, all four subsystems are solved using the recursive blocked algorithm. If a splitting point ($m/2$ or $n/2$) appears at a 2×2 diagonal block of A or B, the matrices are split just below this diagonal block.

The recursive approach is natural to SMP-parallelize, which is implemented in RECSY using OpenMP. The performance gain compared to standard algorithms is remarkable, including 10-fold speedups, partly due to new superscalar kernels. The software and documentation concerning RECSY is available for download [14]. For details we also refer to the papers by Jonsson and Kågström [11, 12].

4 Computational experiments

In this section, we compare measured performance results for the parallel algorithms in Figures 2 and 3 solving SYCT using two different node solvers DTRSYL (from LAPACK) and RECSYCT (from RECSY) in `PGESYCTD`. The test results are for different values of $m = n$ and different process configurations $P_r \times P_c$ on the HPC2N Super Linux Cluster **seth**. The cluster consists of 120 dual Athlon MP2000+ nodes (1.667 GHz), where each node has 1–4 GB memory. The cluster is connected through the Wolfkit3 SCI high-speed interconnect with a bandwidth of 667 Mbytes/second. The network connects the nodes in a 3-dimensional torus organized as a $4 \times 5 \times 6$ grid, where each link is "one-way" directed. The theoretical peak system performance of **seth** is 800 Gflops/sec. The fraction t_a/t_w, where t_a and t_w denote the time for one flop and the per-word transfer time, respectively, is approximately 0.025. Compared to other more well-balanced systems, e.g., the HPC2N IBM SP system which has $t_a/t_w = 0.11$, communication is almost a factor 10 more expensive on **seth**.

The results are displayed in Tables 2 and 3. The variables q_s and q_d are the ratios between the execution times of `PGESYCTD` using the two different communication schemes. These ratios are presented for both node solvers (LAPACK, RECSY). If a ratio is larger than 1.0, the RECSY variant is the fastest, and represents the speedup gain compared to the LAPACK variant.

Table 2. Timing results (in seconds) of `PGESYCTD` using different kernel solvers DTRSYL (LAPACK) and RECSYCT (RECSY) and different communication schemes "matrix-shifting" (S) and "on-demand"(D). Here we use moderate-sized blocking factors $mb = nb = 128$.

		LAPACK		RECSY		Ratios				LAPACK		RECSY		Ratios	
$m=n$	$P_r \times P_c$	S	D	S	D	q_s	q_d	$m=n$	$P_r \times P_c$	S	D	S	D	q_s	q_d
2048	1×1	18.0	18.1	16.0	15.9	1.12	1.12	6144	2×2	573	215	528	200	1.08	1.08
2048	2×1	25.3	15.1	26.5	13.6	0.95	1.11	6144	4×2	277	156	276	148	1.00	1.05
2048	2×2	20.9	9.8	20.2	8.4	1.04	1.16	6144	4×4	160	112	160	103	1.00	1.09
2048	4×2	11.8	8.2	11.5	6.8	1.03	1.21	6144	8×4	74.2	73.0	73.4	62.3	1.01	1.17
2048	4×4	7.6	6.8	7.5	5.5	1.01	1.24	6144	8×8	68.9	65.2	68.4	59.5	1.01	1.09
2048	8×4	4.6	5.4	5.0	4.1	0.91	1.32	8192	4×2	662	359	651	347	1.02	1.03
2048	8×8	4.4	4.6	4.0	3.8	1.10	1.21	8192	4×4	369	247	367	231	1.01	1.07
4096	1×1	134	134	125	126	1.07	1.07	8192	8×4	172	152	169	133	1.02	1.14
4096	2×1	198	111	196	106	1.01	1.05	8192	8×8	153	136	152	127	1.00	1.08
4096	2×2	159	66.1	156	62.7	1.02	1.05	10240	4×4	742	462	714	442	1.04	1.04
4096	4×2	84.9	50.1	84.8	45.9	1.00	1.09	10240	8×4	362	272	336	245	1.08	1.11
4096	4×4	50.1	38.1	49.1	33.8	1.02	1.13	10240	8×8	302	247	301	234	1.00	1.06
4096	8×4	23.8	26.7	22.3	21.8	1.07	1.23	12288	8×4	559	441	556	406	1.01	1.08
4096	8×8	23.3	23.6	22.8	20.8	1.02	1.14	12288	8×8	490	405	488	385	1.00	1.05

Table 3. Timing results (in seconds) of `PGESYCTD` using different kernel solvers DTRSYL (LAPACK) and RECSYCT (RECSY) and different communication schemes "matrix-shifting" (S) and "on-demand" (D). Here we use large blocking factors $mb = nb = 512$. The sign '–' means that the restriction on the data distribution imposed by the "matrix-shifting" scheme was not fulfilled (see Section 2).

		LAPACK		RECSY		Ratios				LAPACK		RECSY		Ratios	
$m=n$	$P_r \times P_c$	S	D	S	D	q_s	q_d	$m=n$	$P_r \times P_c$	S	D	S	D	q_s	q_d
2048	1×1	57.5	54.7	13.0	10.8	4.43	5.06	6144	2×2	425	410	187	123	2.28	3.34
2048	2×1	63.6	53.5	14.9	9.7	4.27	5.52	6144	4×2	329	381	109	93.7	3.02	4.07
2048	2×2	38.3	40.0	10.9	7.2	3.51	5.55	6144	4×4	198	335	75.3	72.5	2.63	4.63
2048	4×2	35.6	38.8	7.9	6.2	4.51	6.25	6144	8×4	–	297	–	59.2	–	5.02
2048	4×4	28.1	32.7	6.4	5.6	4.35	5.83	6144	8×8	–	245	–	50.9	–	4.81
2048	8×4	–	–	–	–	–	–	8192	4×2	580	707	247	202	2.35	3.51
2048	8×8	–	–	–	–	–	–	8192	4×4	350	614	158	152	2.21	4.04
4096	1×1	258	255	80.9	80.1	3.19	3.19	8192	8×4	288	521	107	113	2.68	4.60
4096	2×1	267	234	87.1	63.3	3.05	3.69	8192	8×8	183	413	90.7	91.7	2.02	4.50
4096	2×2	167	170	57.7	40.5	2.89	4.20	10240	4×4	542	989	296	275	1.83	3.60
4096	4×2	138	162	36.6	32.7	3.78	4.97	10240	8×4	–	848	–	200	–	4.24
4096	4×4	89.2	143	31.1	26.5	2.86	5.40	10240	8×8	–	688	–	170	–	4.06
4096	8×4	80.3	122	20.2	22.1	3.98	5.53	12288	8×4	657	1220	311	314	2.11	3.89
4096	8×8	55.7	95.4	17.1	17.1	3.26	5.57	12288	8×8	406	971	257	256	1.58	3.80

Table 4. Ratios q_{best} and gain $g = 1 - q_{\text{best}}^{-1}$ in percent between the best timing results from Tables 2 and 3 for `PGESYCTD` using different kernel solvers DTRSYL (LAPACK) and RECSYCT (RECSY) and different communication schemes "matrix-shifting" (S) and "on-demand" (D).

$P_r \times P_c$	$m=n$	q_{best}	$g(\%)$	$m=n$	q_{best}	$g(\%)$	$P_r \times P_c$	$m=n$	q_{best}	$g(\%)$	$P_r \times P_c$	$m=n$	q_{best}	$g(\%)$
1×1	2048	1.67	40	4096	1.67	40	2×2	6144	1.75	43	8×4	8192	1.42	30
2×1	2048	1.56	36	4096	1.75	43	4×2	6144	1.66	40	8×8	8192	1.50	33
2×2	2048	1.36	26	4096	1.63	39	4×4	6144	1.54	35	4×4	10240	1.68	40
4×2	2048	1.32	24	4096	1.53	35	8×4	6144	1.23	19	8×4	10240	1.36	26
4×4	2048	1.24	19	4096	1.44	31	8×8	6144	1.28	22	8×8	10240	1.45	31
8×4	2048	–	–	4096	1.18	15	4×2	8192	1.22	18	8×4	12288	1.42	30
8×8	2048	–	–	4096	1.35	26	4×4	8192	1.63	39	8×8	12288	1.58	37

5 Discussion and conclusions

The results in Table 2 show that the RECSYCT solver decreases the execution time up to 24% for moderate-sized block sizes $mb = nb = 128$ when "on-demand" communication is used, while the gain is only up to 9% for the "matrix-shifting" scheme. Note the exceptions $m = n = 2048$ using a 2×1 and an 8×4 processor grid when `PGESYCTD` with the LAPACK solver DTRSYL gives 5% and 9% shorter execution time, respectively.

From the results in Table 3, we conclude that the execution times for `PGESYCTD` using RECSYCT decrease for larger block sizes ($mb = nb = 512$), while the execution times for `PGESYCTD` using DTRSYL increase drastically compared to the results in Table 2.

In Table 4, we display the ratios of the shortest execution times of `PGESYCTD` using DTRSYL and RECSYCT, respectively, and one of the two communication schemes for a given processor grid and problem size. Overall the RECSYCT solver decreases the execution times between 15% and 43% compared to DTRSYL. The best results for RECSYCT are obtained when "on-demand" communication is used, while the best results for DTRSYL are obtained for the "matrix-shifting" scheme.

In conclusion, `PGESYCTD` with the RECSYCT solver has a large impact on the performance when mb and nb are several hundreds. Typically, `PGESYCTD` with the DTRSYL solver is optimal for smaller block sizes. We also expect `PGESYCTD` with RECSYCT to give less speedup compared to using DTRSYL, since a much faster node solver makes overlapping of communication and computation harder. On the other hand, by the use of larger block sizes, i.e., larger SYCT subsystems are solved on the nodes, we also get less but larger messages to communicate, which may well compensate for the worse communication-computation overlap.

Future work includes extending the comparisons to other parallel platforms, e.g., the HPC2N IBM SP system which has much less compute power but provides a better "compute/communicate ratio". Our aim is to construct a software package of ScaLAPACK-style algorithms for solving all matrix equations listed in Table 1. The implementations will build on standard node solvers from LAPACK and SLICOT [18, 20, 5], and recursive blocked solvers from RECSY. By using the LAPACK/SLICOT wrappers provided in the RECSY library, the ScaLAPACK-style hybrid algorithms come for free.

Acknowledgements

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.

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. 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.
5. E. Elmroth, P. Johansson, B. Kågström, and D. Kressner, 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.
6. R. Granat, A Parallel ScaLAPACK-style Sylvester Solver, *Master Thesis*, UMNAD 435/03, Dept. Computing Science, Umeå University, Sweden, January, 2003.
7. R. Granat, B. Kågström, P. Poromaa. Parallel ScaLAPACK-style Algorithms for Solving Continous-Time Sylvester Matrix Equations, In H. Kosch et.al. (editors), *Euro-Par 2003 Parallel Processing*, Lecture Notes in Computer Science, Springer-Verlag, Vol. 2790, pages 800–809, 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. 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. I. Jonsson and B. Kågström. RECSY - A High Performance Library for Solving Sylvester-Type Matrix Equations, In H. Kosch et.al. (editors), *Euro-Par 2003 Parallel Processing*, Lecture Notes in Computer Science, Springer-Verlag, Vol. 2790, pages 810–819, 2003.
14. I. Jonsson and B. Kågström. RECSY — A High Performance Library for Sylvester-Type Matrix Equations. `www.cs.umu.se/research/parallel/recsy`, 2003.
15. 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.
16. 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.
17. 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.
18. NICONET Task II: Model Reduction, website: www.win.tue.nl/niconet/NIC2/NICtask2.html
19. 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.
20. SLICOT library in the Numerics in Control Network (NICONET), website: www.win.tue.nl/niconet/index.html