Evaluating Parallel Algorithms for Solving Sylvester-Type Matrix Equations: Direct Transformation-Based versus Iterative Matrix-Sign-Function-Based Methods

Robert Granat and Bo Kågström

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

Abstract. Recent ScaLAPACK-style implementations of the Bartels-Stewart method and the iterative matrix-sign-function-based method for solving continuous-time Sylvester matrix equations are evaluated with respect to generality of use, execution time and accuracy of computed results. The test problems include well-conditioned as well as ill-conditioned Sylvester equations. A method is considered more general if it can effectively solve a larger set of problems. Ill-conditioning is measured with respect to the separation of the two matrices in the Sylvester operator. Experiments carried out on two different distributed memory machines show that the parallel explicitly blocked Bartels-Stewart algorithm can solve more general problems and delivers far more accuracy for ill-conditioned problems. It is also up to four times faster for large enough problems on the most balanced parallel platform (IBM SP), while the parallel iterative algorithm is almost always the fastest of the two on the less balanced platform (HPC2N Linux Super Cluster).

Keywords: Sylvester matrix equation, continuous-time, Bartels–Stewart method, explicit blocking, GEMM-based, level 3 BLAS, matrix sign function, c-stable matrices, Newton iteration, ScaLAPACK, PSLICOT.

1 Introduction

We consider two different methods for solving the continuous-time Sylvester (SYCT) equation

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

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. SYCT has a unique solution X of size $m \times n$ if and only if A and B have disjoint spectra, i.e., they have no eigenvalues in common, or equivalently the separation $\mathrm{sep}(A, B) \neq 0$, where $\mathrm{sep}(A, B) = \inf_{\|X\|_F=1} \|AX - XB\|_F$.

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 [13, 11, 15]).

In this contribution, we experimentally compare parallel implementations of two methods for solving SYCT on distributed memory systems. The first is based on Bartels-Stewart method [1, 15, 7, 8] and is reviewed in Section 2. Several papers, starting with [16], have considered iterative methods for solving SYCT. The second parallel implementation is an iterative method based on a Newton iteration of the matrix sign function [3, 4], and is reviewed in Section 3. In Section 4, we display and compare some computational results of implementations of the two parallel algorithms. A discussion of the measured execution times on two different parallel distributed memory platforms is presented in Section 5. Finally, in Section 6, we summarize our findings and outline ongoing and future work.

2 Explicitly blocked algorithms for solving SYCT

The explicitly blocked method, implemented as the routine `PGESYCTD`, is based on the Bartels-Stewart method [1]:

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

To carry out Step 1 we use a Hessenberg reduction directly followed by the QR-algorithm. The updates in Step 2 and the back-transformation in Step 4 are carried out using general matrix multiply and add (GEMM) operations $C \leftarrow \beta C + \alpha \mathrm{op}(A)\mathrm{op}(B)$, where α and β are scalars and $\mathrm{op}(A)$ denotes A or its transpose A^T [2, 12]. We now focus on Step 3. If the matrices A and B are in Schur form, we partition the matrices 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 $D_b = \lceil n/nb \rceil$, (1) can be rewritten as

$$A_{ii} X_{ij} - X_{ij} B_{jj} = C_{ij} - \Big(\sum_{k=i+1}^{D_a} A_{ik} X_{kj} - \sum_{k=1}^{j-1} X_{ik} B_{kj} \Big), \tag{2}$$

where $i = 1, 2, \ldots, D_a$ and $j = 1, 2, \ldots, D_b$. From (2), a serial blocked algorithm can be formulated, see Figure 1.

Assume that the matrices A, B and C are distributed using 2D block-cyclic mapping across a $P_r \times P_c$ processor grid. For Steps 1, 2 and 4, we use the ScaLAPACK library routines `PDGEHRD`, `PDLAHQR` and `PDGEMM` [6]. The first two routines are used in Step 1 to compute the Schur decompositions of A and B (reduction to upper Hessenberg form

```
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 upper real Schur form.

followed by the parallel QR algorithm [10, 9]). `PDGEMM` is the parallel implementation of the level 3 BLAS GEMM 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 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 owned by the same process that owns C_{ij}. We also need to have the blocks used in the updates of C_{ij} in the right place at the right time. This means that in general we have to communicate for some blocks during the solves and updates. This is done "on demand": whenever a processor misses any block that it needs for solving a node subsystem or doing a GEMM update, it is received from the owner in a single point-to-point communication [8]. A brief outline of a parallel algorithm `PTRSYCTD` is presented in Figure 2. The (quasi-)triangular subsystems $A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij}$ in Figure 2 are solved on the grid nodes using the LAPACK-solver `DTRSYL` [2].

```
for k=1, D_a + D_b − 1
    {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 block algorithm for $AX - XB = C$, A and B in upper real Schur form.

The explicitly blocked method is general since (in theory) it can be used to solve any instance of SYCT as long as the spectra of A and B are disjoint. Notice that the

algorithm contains an iterative part, the reduction of A and B in upper Hessenberg form to upper Schur form, which is the most expensive in terms of execution time [7, 8]. For further details we refer to [7, 8, 15].

3 Solving SYCT using Newton iteration for the matrix sign function

The sign function method can be used to solve SYCT if the spectra of A and $-B$ are contained in the open left half complex plane, i.e., A and $-B$ are so called *Hurwitz-* or c-stable [3].

Let Z be a real $p \times p$ matrix with real eigenvalues and let

$$Z = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1} \tag{3}$$

denote its Jordan decomposition with $J^- \in C^{k\times k}, J^+ \in C^{(p-k)\times(p-k)}$ containing the Jordan blocks corresponding to the eigenvalues in the open left and right half planes, respectively. Then the *matrix sign function* of Z is defined as

$$\operatorname{sign}(Z) = S \begin{bmatrix} -I_k & 0 \\ 0 & I_{p-k} \end{bmatrix} S^{-1}. \tag{4}$$

The sign function can be computed via the Newton iteration for the equation $Z^2 = I$ where the starting point is chosen as Z, i.e.,

$$\begin{aligned} &Z_0 = Z, \\ &Z_{k+1} = (Z_k + Z_k^{-1})/2, \qquad k = 0, 1, 2, \ldots \end{aligned} \tag{5}$$

It can be proved [16] that $\operatorname{sign}(Z) = \lim_{k\to\infty} Z_k$ and moreover that

$$\operatorname{sign}\left(\begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}\right) + I_{m+n} = 2 \begin{bmatrix} 0 & X \\ 0 & I \end{bmatrix}, \tag{6}$$

which means that under the given assumptions, SYCT can be solved by applying the Newton iteration (5) to

$$Z_0 = \begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}. \tag{7}$$

This iterative process only requires basic numerical linear algebra operations as matrix-matrix multiplication, inversion and/or solving linear systems of equations. The method has been parallelized and implemented as the PSLICOT [14, 17] routine `psb04md`. The parallel implementation uses the ScaLAPACK routines `PDGETRF` (LU factorization), `PDGETRS` (solving linear systems of equations), `PDGETRI` (inversion based on LU factorization), `PDTRSM` (solution of triangular systems with multiple right-hand sides) and `PDLAPIV` (pivoting of a distributed matrix). We expect this iterative method to be fast and scalable since it consists of computationally well-known and highly parallel operations. The obvious drawback of the method is the lower degree of generality, i.e., the fact that we cannot solve all instances of SYCT due to the restrictions on the spectra of A and B.

The matrix sign function method can also be applied to other related problems, e.g., the stable generalized Lyapunov equation $A^T XE + E^T XA = C$, where $A, E, X, C \in$

$R^{n\times n}$ and $C = C^T$ [3], and it can also be combined with iterative refinement for higher accuracy. However, this has not been incorporated in `psb04md` [4]. For further details we refer to [16, 4, 3, 5].

4 Computational results

In this section, we present and compare measured speed and accuracy results of `PGESYCTD` and `psb04md` using two different parallel platforms. We solve a number of differently conditioned (regarding the separation of A and B) problems of various sizes using different processor mesh configurations.

Our target machines are the IBM Scalable POWERparallel (SP) system and the Super Linux Cluster at High Performance Computing Center North (HPC2N), where we utilize up to 64 processors of each machine (see Table 4). The test problems are constructed as follows. Consider the matrix A in the form $A = Q(\alpha D_A + \beta M_A)Q^T$, where D_A is (quasi-)diagonal, M_A is strictly upper triangular, Q is orthogonal and α and β are real scalars. We choose M_A as a random matrix with uniformly distributed elements in [-1,1] and prescribe the eigenvalues of A by specifying the elements of D_A. If the matrix B is constructed similarly, we can construct differently conditioned problems by varying the eigenvalues in D_A and D_B and choosing appropriate values of the scaling factors. For example, the factor β is used to control the distance from normality, $\beta\|M_A\|$, of the matrix A.

A representative selection of our results for problems with $m = n$ are shown in Tables 1, 2, and 3. Results for problems with $m \neq n$ are not presented here but will not lead to any different conclusions. We have chosen close to optimal block sizes for the different parallel algorithms and parallel architectures. The upper parts of Tables 1 and 2 correspond to well-conditioned problems, and the lower parts represent moderately to very ill-conditioned problems. We display the performance ratios q_T, q_X and q_R, which correspond to the execution time ratio and two accuracy ratios, the Frobenius norms of the absolute (forward) error $\|X - \tilde{X}\|_F$ and the absolute residual $\|R\|_F = \|C - A\tilde{X} + \tilde{X}B\|_F$ of the two implementations, where X and $\tilde{X}$ denote the exact and computed solutions, respectively. If a ratio is larger than 1.0, `psb04md` shows better results, otherwise `PGESYCTD` is better.

When the algorithm in `psb04md` converged, it did so in less than 20 iterations. We used an upper threshold of 100 iterations. To signal that `psb04md` does not converge, we use the notation *dnc*.

Recall that SYCT has a unique solution if and only if the A and B have disjoint spectra, or equivalently $\mathrm{sep}(A, B) \neq 0$, where

$$\mathrm{sep}(A, B) = \inf_{\|X\|_F=1} \|AX - XB\|_F = \sigma_{\min}(Z_{\mathrm{SYCT}}) = \|Z_{\mathrm{SYCT}}^{-1}\|_2^{-1}, \qquad (8)$$

and Z_{SYCT} is the $mn \times mn$ matrix representation of the Sylvester operator defined by $Z_{\mathrm{SYCT}} = I_n \otimes A + B^T \otimes I_m$. Moreover, $\mathrm{sep}(A, B)$ is a condition number for the SYCT equation, but it is expensive to compute in practice ($O(m^3n^3)$ operations). However, we can compute a lower bound of $\mathrm{sep}(A, B)^{-1} = \|Z_{SYCT}^{-1}\|_2$ in parallel, which is based on the same technique as described in [13, 11]. Since a tiny value of $\mathrm{sep}(A, B)$ signals ill-conditioning for SYCT, the sep^{-1}-estimate signals ill-conditioning when it gets large. We expect the explicitly blocked method to handle ill-conditioned problems better than the fully iterative, since it relies on orthogonal transformations and it has a direct (backward stable) solution method for the reduced triangular problem. To signal

Table 1. Speed and accuracy results on IBM SP system for the routines PGESYCTD and psb04md solving the general equation $AX - XB = C$ for well- and ill-conditioned problems. All problems use the blocking factors $mb = nb = 64$. In the upper part of the table, A and $-B$ have the eigenvalues $\lambda_i = -i, i = 1, 2, \ldots, m = n$, and $\alpha = \beta = 1.0$. For the lower part A and $-B$ have the eigenvalues: a) $\lambda_{Ai} = -i, \lambda_{Bi} = -1000.0 \cdot i^{-1}$, $\alpha = \beta = 1.0$, b) $\lambda_{Ai} = -i, \lambda_{Bi} = -2000.0 \cdot i^{-1}, \alpha = \beta = 1.0$, c) A and $-B$ have the eigenvalues $\lambda_i = -i, \alpha_A = \alpha_B = \beta_A = 1.0, \beta_B = 50.0$.

			PGESYCTD			psb04md			Performance ratios		
$m = n$	sep^{-1}	$P_r \times P_c$	T_p	$\|X - \tilde{X}\|_F$	$\|R\|_F$	T_p	$\|X - \tilde{X}\|_F$	$\|R\|_F$	q_T	q_X	q_R
512	3.7E-3	1 × 1	60.0	4.1E-12	1.2E-9	43.8	1.1E-12	2.4E-10	1.37	3.73	0.50
512	3.8E-3	2 × 1	49.2	4.1E-12	1.2E-9	37.1	2.1E-12	9.1E-10	1.33	1.95	1.32
512	3.8E-3	2 × 2	40.4	4.3E-12	1.2E-9	33.0	1.8E-12	8.0E-10	1.22	2.39	1.50
512	3.8E-3	4 × 2	33.6	3.8E-12	1.1E-9	33.1	1.9E-12	8.0E-10	1.02	2.00	1.38
1024	2.1E-3	1 × 1	534	1.2E-11	6.6E-9	529	1.0E-10	3.3E-8	1.01	0.12	0.20
1024	2.1E-3	2 × 1	287	1.2E-11	6.5E-9	253	8.2E-12	6.8E-9	1.13	1.38	0.96
1024	2.1E-3	2 × 2	177	1.1E-11	5.4E-9	169	2.2E-11	8.9E-9	1.05	0.50	0.61
1024	2.1E-3	4 × 2	138	1.1E-11	6.6E-9	170	8.1E-12	6.8E-9	0.81	1.36	0.97
1024	2.1E-3	4 × 4	106	1.3E-11	6.7E-9	142	6.6E-12	5.5E-9	0.75	1.97	1.22
2048	1.1E-3	2 × 2	1625	3.8E-11	3.5E-8	6141	2.9E-11	4.9E-8	0.26	1.31	0.71
2048	1.1E-3	4 × 2	835	3.5E-11	3.7E-8	1031	3.5E-11	4.8E-8	0.81	1.00	0.08
2048	1.1E-3	4 × 4	446	3.3E-11	3.7E-8	658	2.1E-11	3.6E-8	0.68	1.57	1.03
2048	1.1E-3	8 × 4	359	4.0E-11	3.7E-8	670	2.5E-11	3.9E-8	0.54	1.60	0.95
2048	1.1E-3	8 × 8	302	3.5E-11	3.7E-8	588	2.5E-11	3.9E-8	0.51	1.40	0.95
3072	7.6E-4	4 × 2	3355	6.3E-11	9.7E-8	11504	5.7E-11	1.5E-7	0.29	1.11	0.65
3072	7.6E-4	4 × 4	1677	6.3E-11	1.0E-7	2473	4.9E-11	1.3E-7	0.68	1.29	0.77
3072	7.6E-4	8 × 4	1056	6.4E-11	1.0E-7	2078	1.3E-10	2.0E-7	0.51	0.49	0.50
3072	7.6E-4	8 × 8	705	6.6E-11	1.0E-7	1556	6.3E-10	6.8E-7	0.45	0.10	0.15
4096	5.9E-4	4 × 4	3788	9.5E-11	2.0E-7	11602	8.3E-11	2.7E-7	0.33	1.14	0.74
4096	5.9E-4	8 × 4	2330	9.6E-11	2.1E-7	5036	7.8E-11	2.5E-7	0.46	1.23	0.84
4096	5.9E-4	8 × 8	1365	1.0E-10	2.1E-7	3102	8.8E-11	2.7E-7	0.44	0.11	0.78
512^a	0.17	1 × 1	61.7	6.1E-11	7.2E-10	33.8	1.1E-9	2.6E-7	1.8	5.5E-2	2.8E-3
512^a	0.16	2 × 1	47.0	8.5E-11	6.9E-10	29.7	9.4E-10	2.2E-7	1.6	3.1E-3	3.1E-3
512^a	0.15	2 × 2	36.1	8.8E-11	6.9E-10	26.8	1.2E-9	2.8E-7	1.4	7.3E-2	2.5E-3
512^a	0.16	4 × 2	30.6	8.3E-11	6.7E-10	25.1	1.1E-9	2.8E-7	1.2	7.5E-2	2.4E-3
1024^b	2.8	1 × 1	635	7.6E-9	4.0E-9	575	4.9E-5	2.9E-2	1.1	1.6E-4	1.4E-7
1024^b	2.8	2 × 1	326	4.1E-9	3.9E-9	206	3.1E-5	1.8E-2	1.6	1.3E-4	2.2E-7
1024^b	2.7	2 × 2	188	8.8E-9	3.8E-9	156	6.0E-5	3.3E-2	1.2	1.5E-4	1.2E-7
1024^b	2.7	4 × 2	125	1.1E-8	3.8E-9	131	3.6E-5	2.1E-2	0.95	3.1E-4	1.8E-7
1024^b	3.4	4 × 4	91	9.3E-9	3.8E-9	118	4.4E-5	2.7E-2	0.77	2.1E-4	1.4E-7
2048^c	*mem*	2 × 2	1797	8.3E-5	4.1E-8	43708	*dnc*	*dnc*	0.0	0.0	0.0
2048^c	4.8E4	4 × 2	892	5.8E-5	4.1E-8	1158	1.2	1343	0.77	4.8E-5	3.1E-11
2048^c	5.2E4	4 × 4	446	7.2E-5	4.1E-8	708	0.86	982	0.63	8.4E-5	4.2E-11
2048^c	5.4E4	8 × 4	348	5.1E-5	4.1E-8	738	0.92	1034	0.47	5.5E-5	4.0E-11
2048^c	4.8E4	8 × 8	281	4.2E-5	4.1E-8	733	0.88	1003	0.38	4.8E-5	4.1E-11
3072^c	*mem*	4 × 2	3148	1.4E-5	1.1E-7	11358	2.9	5054	0.28	4.8E-6	2.2E-11
3072^c	3.9E3	4 × 4	1635	1.1E-5	1.1E-7	2936	1.8	3198	0.51	6.1E-6	3.4E-11
3072^c	4.7E3	8 × 4	878	7.3E-6	1.1E-7	2051	1.9	3305	0.43	3.8E-6	3.3E-11
3072^c	4.8E3	8 × 8	769	8.6E-6	1.1E-7	1785	1.7	2926	0.43	5.1E-6	3.8E-11
4096^c	*mem*	4 × 4	3805	1.9E-4	2.1E-7	15435	85.9	180879	0.25	2.2E-6	1.2E-12
4096^c	5.5E4	8 × 4	2066	3.3E-4	2.1E-7	41439	*dnc*	*dnc*	0.0	0.0	0.0
4096^c	6.5E4	8 × 8	1327	1.9E-4	2.1E-7	22487	*dnc*	*dnc*	0.0	0.0	0.0

when there is not enough memory for both solving the problem and doing condition estimation, we use the notation *mem*.

5 Discussion of computational results

The experimental results from the last section reveal the following: For triangular problems ($Q_A = Q_B = I$), see Table 3 which displays results from the Linux Cluster, the parallel Bartels-Stewart based method is very much faster than the fully iterative since it always computes the solution directly using a fixed number of arithmetic operations.

Table 2. Speed and accuracy results on Super Linux Cluster `seth` for the routines `PGESYCTD` and `psb04md` solving the general equation $AX - XB = C$ for well- and ill-conditioned problems. All problems use the blocking factors $mb = nb = 32$. In the upper part of the table, A and $-B$ have the eigenvalues $\lambda_i = -i, i = 1, 2, \ldots, m = n$, and $\alpha = \beta = 1.0$. For the lower part A and $-B$ have the eigenvalues: a) $\lambda_{Ai} = -i, \lambda_{Bi} = -1000.0 \cdot i^{-1}$, $\alpha = \beta = 1.0$, b) $\lambda_{Ai} = -i, \lambda_{Bi} = -2000.0 \cdot i^{-1}, \alpha = \beta = 1.0$, c) A and $-B$ have the eigenvalues $\lambda_i = -i, \alpha_A = \alpha_B = \beta_A = 1.0, \beta_B = 50.0$.

			PGESYCTD			psb04md			Performance ratios		
$m = n$	sep^{-1}	$P_r \times P_c$	T_p	$\|X - \tilde{X}\|_F$	$\|R\|_F$	T_p	$\|X - \tilde{X}\|_F$	$\|R\|_F$	q_T	q_X	q_R
1024	2.5E-3	1 × 1	277	6.4E-12	3.5E-9	101	2.9E-12	2.4E-9	2.74	2.21	1.46
1024	2.4E-3	2 × 2	136	6.6E-12	3.5E-9	44	2.8E-12	2.3E-9	3.09	2.36	1.52
1024	2.4E-3	3 × 3	54	6.2E-12	3.5E-9	27	2.8E-12	2.3E-9	2.00	2.21	1.52
1024	2.3E-3	4 × 4	50	6.4E-12	3.6E-9	21	2.8E-12	2.3E-9	2.38	2.29	1.57
1024	2.3E-3	5 × 5	41	7.1E-12	3.6E-9	18	2.8E-12	2.3E-9	2.28	2.54	1.57
1024	2.3E-3	6 × 6	30	6.2E-12	3.6E-9	16	2.8E-12	2.3E-9	1.88	2.21	1.57
1024	2.3E-3	7 × 7	29	7.0E-12	3.6E-9	14	2.8E-9	2.3E-9	2.07	2.50	1.57
1024	2.5E-3	8 × 8	27	5.9E-12	3.4E-9	18	2.9E-12	2.4E-9	1.50	2.03	1.17
2048	1.3E-3	1 × 1	2053	2.1E-11	1.9E-8	1166	1.0E-11	1.8E-8	1.76	2.10	1.06
2048	1.2E-3	2 × 2	763	1.8E-11	1.9E-8	269	1.0E-11	1.7E-8	2.84	1.80	1.12
2048	1.2E-3	3 × 3	505	1.9E-11	2.0E-8	211	1.0E-11	1.7E-8	2.50	1.90	1.18
2048	1.2E-3	4 × 4	331	1.9E-11	1.9E-8	120	1.0E-11	1.7E-8	2.76	1.90	1.12
2048	1.2E-3	5 × 5	181	1.9E-11	2.0E-8	87	1.0E-11	1.7E-8	2.08	1.90	1.18
2048	1.2E-3	6 × 6	143	1.9E-11	2.0E-8	71	1.0E-11	1.7E-8	2.01	1.90	1.18
2048	1.2E-3	7 × 7	128	1.9E-11	2.0E-8	63	1.0E-11	1.7E-8	2.03	1.90	1.18
2048	1.3E-3	8 × 8	140	2.0E-11	2.0E-8	54	1.0E-11	1.7E-8	2.59	2.00	1.18
4096	7.0E-4	2 × 2	6514	5.2E-11	1.1E-7	3174	3.8E-11	1.3E-7	2.05	1.37	0.85
4096	7.0E-4	3 × 3	3338	5.0E-11	1.1E-7	1131	3.8E-11	1.3E-7	2.95	1.32	0.85
4096	7.0E-4	4 × 4	2912	5.0E-11	1.1E-7	909	3.8E-11	1.3E-7	3.20	1.32	0.85
4096	7.0E-4	5 × 5	1466	5.1E-11	1.1E-7	526	3.8E-11	1.3E-7	2.79	1.34	0.85
4096	7.0E-4	6 × 6	1027	5.1E-11	1.1E-7	398	3.8E-11	1.3E-7	2.58	1.34	0.85
4096	7.0E-4	7 × 7	804	5.8E-11	1.1E-7	342	3.8E-11	1.3E-7	2.35	1.53	0.85
4096	6.6E-4	8 × 8	890	5.4E-11	1.1E-7	295	3.8E-11	1.3E-7	3.02	1.42	0.85
8192	*mem*	4 × 4	15543	1.6E-10	7.5E-7	7425	1.5E-10	9.8E-7	2.09	1.07	0.77
8192	3.5E-4	5 × 5	10435	1.5E-10	7.2E-7	3817	1.5E-10	9.9E-7	2.73	1.00	0.73
8192	3.5E-4	6 × 6	7987	1.5E-10	6.4E-7	2740	1.5E-10	9.9E-7	2.91	1.00	0.65
8192	3.5E-4	7 × 7	6224	1.7E-10	6.5E-7	2197	1.5E-10	9.8E-7	2.83	1.13	0.66
8192	3.6E-4	8 × 8	5313	1.7E-10	6.8E-7	2247	1.5E-10	9.9E-7	2.36	1.13	0.69
1024[a]	9.2	1 × 1	274	4.3E-9	2.2E-9	73	2.1E-5	1.2E-2	3.75	2.0E-4	1.8E-7
1024[a]	8.2	2 × 2	121	6.3E-9	2.1E-9	36	1.2E-5	7.2E-3	3.36	5.3E-4	2.9E-7
1024[a]	5.9	3 × 3	52	4.7E-9	2.1E-9	22	2.1E-5	1.2E-2	2.36	2.2E-4	1.8E-7
1024[a]	7.2	4 × 4	51	3.3E-9	2.1E-9	18	3.1E-5	1.8E-2	2.83	1.1E-4	1.2E-7
1024[a]	4.9	5 × 5	35	5.2E-9	2.1E-9	15	3.5E-5	2.1E-2	2.33	1.5E-4	1.0E-7
1024[a]	6.4	6 × 6	28	5.5E-9	2.1E-9	12	1.9E-5	1.1E-2	2.33	2.9E-4	1.2E-7
1024[a]	9.0	7 × 7	28	6.3E-9	2.1E-9	12	1.9E-5	1.2E-2	2.33	3.3E-4	1.8E-7
1024[a]	4.3	8 × 8	26	4.2E-9	2.1E-9	10	1.6E-5	9.6E-3	2.60	2.6E-4	2.2E-7
2048[b]	5.1E4	1 × 1	2223	2.2E-5	2.3E-8	1211	0.34	389	1.84	6.5E-4	5.9E-11
2048[b]	4.9E4	2 × 2	859	1.7E-5	2.3E-8	284	0.36	407	3.02	4.7E-4	5.7E-11
2048[b]	6.4E4	3 × 3	496	3.4E-5	2.3E-8	178	0.41	474	2.79	8.3E-5	4.9E-11
2048[b]	5.1E4	4 × 4	356	6.3E-5	3.1E-8	132	0.36	421	2.70	1.8E-4	7.4E-11
2048[b]	6.6E4	5 × 5	189	4.4E-5	2.3E-8	95	0.36	428	1.99	1.2E-4	5.4E-11
2048[b]	6.7E4	6 × 6	152	8.5E-5	2.3E-8	77	0.34	422	1.97	2.5E-4	5.5E-11
2048[b]	6.7E4	7 × 7	127	3.5E-5	2.3E-8	68	0.36	412	1.87	9.7E-5	5.6E-11
2048[b]	6.6E4	8 × 8	145	3.1E-5	2.3E-8	58	0.35	394	2.50	8.9E-5	5.8E-11
4096[b]	1.3E6	2 × 2	6470	8.1E-3	1.2E-7	24008	*dnc*	*dnc*	0.27	0.0	0.0
4096[b]	6.6E4	3 × 3	3611	2.2E-4	1.2E-7	1470	63.3	1.5E5	2.46	3.5E-6	8.0E-13
4096[b]	5.6E4	4 × 4	2271	1.9E-4	5.1E-6	1202	57.9	1.4E5	1.89	3.3E-6	3.6E-11
4096[b]	5.8E4	5 × 5	1628	2.8E-4	1.2E-7	4201	*dnc*	*dnc*	0.39	0.0	0.0
4096[b]	6.4E4	6 × 6	1134	2.3E-4	1.2E-7	476	62.4	1.5E5	2.38	3.7E-6	8.0E-13
4096[b]	8.4E4	7 × 7	839	2.2E-4	1.2E-5	409	44.9	1.0E5	2.05	4.9E-6	1.2E-10
4096[b]	6.7E4	8 × 8	916	1.5E-4	1.2E-7	337	51.5	1.2E5	2.72	2.9E-6	1.0E-12
8192[b]	mem	4 × 4	16538	5.6E-4	6.4E-7	7652	1.22	5.7E3	2.16	4.6E-4	1.1E-10
8192[b]	2.5E4	5 × 5	10532	7.5E-4	6.6E-7	4037	1.22	5.7E3	2.61	6.1E-4	1.2E-10
8192[b]	2.5E4	6 × 6	8388	2.7E-4	6.3E-7	3146	0.87	4.1E3	2.67	3.1E-4	1.5E-10
8192[b]	4.8E4	7 × 7	5623	3.5E-4	6.3E-7	2387	0.37	1.8E3	2.36	9.5E-4	3.5E-10
8192[b]	3.4E4	8 × 8	5365	8.8E-4	6.0E-7	2240	0.67	2.4E3	2.40	1.3E-4	2.5E-10

Table 3. Speed and accuracy results on Super Linux Cluster `seth` for the routines `PGESYCTD` and `psb04md` solving the triangular ($Q_A = Q_B = I$) equation $AX - XB = C$, A and B in real Schur form for well-conditioned problems. All problems use the blocking factors $mb = nb = 128$. A and $-B$ have the eigenvalues $\lambda_i = -i, i = 1, 2, \ldots, m = n$, and $\alpha = \beta = 1.0$.

			PGESYCTD			psb04md			Performance ratios		
$m = n$	sep^{-1}	$P_r \times P_c$	T_p	$\|X - \tilde{X}\|_F$	$\|R\|_F$	T_p	$\|X - \tilde{X}\|_F$	$\|R\|_F$	q_T	q_X	q_R
1024	2.5E-3	1 × 1	2.6	3.2E-13	3.8E-10	94.3	3.8E-13	4.1E-10	2.7E-2	0.84	0.93
1024	2.5E-3	2 × 2	1.7	3.9E-13	5.3E-10	45.7	4.7E-13	5.5E-10	3.7E-2	0.83	0.96
1024	2.5E-3	3 × 3	1.6	3.9E-13	4.9E-10	30.3	4.7E-13	5.1E-10	5.2E-2	0.83	0.96
1024	2.5E-3	4 × 4	1.5	3.8E-13	5.2E-10	23.5	4.7E-13	5.5E-10	6.4E-2	0.81	0.95
1024	2.5E-3	5 × 5	1.4	4.0E-13	4.7E-10	22.9	4.8E-13	5.0E-10	6.1E-2	0.83	0.94
1024	2.5E-3	6 × 6	1.4	4.1E-13	4.7E-10	22.5	4.9E-13	5.0E-10	6.2E-2	0.84	0.94
1024	2.5E-3	7 × 7	1.4	4.4E-13	5.1E-10	21.1	5.0E-13	5.3E-10	6.6E-2	0.88	0.96
1024	$1.1E-3$	8 × 8	1.0	3.9E-13	5.5E-10	17.0	4.5E-13	5.8E-10	5.9E-2	0.87	0.95
2048	1.3E-3	1 × 1	17.8	1.2E-12	3.1E-9	1116	1.1E-12	2.3E-9	1.6E-2	1.01	1.35
2048	1.3E-3	2 × 2	9.5	1.6E-12	4.2E-9	287	1.5E-12	3.7E-9	3.3E-2	1.07	1.14
2048	1.3E-3	3 × 3	7.9	1.6E-12	3.9E-9	166	1.5E-12	3.4E-9	4.8E-2	1.07	1.15
2048	1.3E-3	4 × 4	6.7	1.5E-12	4.0E-9	125	1.5E-12	3.5E-9	5.4E-2	1.00	1.14
2048	1.3E-3	5 × 5	5.9	1.6E-12	3.9E-9	114	1.6E-12	3.4E-9	5.2E-2	1.00	1.15
2048	1.3E-3	6 × 6	5.2	1.5E-12	3.8E-9	87.4	1.5E-12	3.2E-9	5.9E-2	1.00	1.19
2048	1.3E-3	7 × 7	5.0	1.6E-12	3.8E-9	83.8	1.6E-12	3.2E-9	6.0E-2	1.00	1.19
2048	1.3E-3	8 × 8	4.4	1.5E-12	4.1E-9	64.8	1.5E-12	3.6E-9	6.7E-2	1.00	1.14
4096	7.0E-4	1 × 1	129	4.9E-12	2.5E-8	8812	3.0E-12	1.3E-8	1.5E-2	1.63	1.92
4096	6.9E-4	2 × 2	63.4	6.1E-12	3.3E-8	2687	5.1E-12	2.6E-8	2.3E-2	1.20	1.27
4096	6.9E-4	3 × 3	45.7	6.1E-12	3.2E-8	1018	5.4E-12	2.4E-8	4.5E-2	1.13	1.33
4096	6.9E-4	4 × 4	37.1	6.2E-12	3.2E-8	701	5.3E-12	2.4E-8	5.3E-2	1.17	1.33
4096	6.9E-4	5 × 5	29.3	6.3E-12	3.1E-8	554	5.5E-12	2.3E-8	5.3E-2	1.15	1.35
4096	6.9E-4	6 × 6	25.3	6.3E-12	3.0E-8	439	5.6E-12	2.3E-8	5.8E-2	1.13	1.30
4096	6.9E-4	7 × 7	25.6	6.2E-12	3.0E-8	385	5.6E-12	2.2E-8	6.6E-2	1.10	1.36
4096	6.9E-4	8 × 8	22.3	6.2E-12	3.0E-8	326	5.5E-12	2.4E-8	6.8E-2	1.13	1.25
8192	3.5E-4	4 × 4	235	2.5E-11	2.5E-7	6003	2.0E-11	1.8E-7	3.9E-2	1.25	1.39
8192	3.5E-4	5 × 5	167	2.5E-11	2.5E-7	*dnc*	*dnc*	*dnc*	0.0	0.0	0.0
8192	3.5E-4	6 × 6	139	2.6E-11	3.5E-7	2202	2.1E-11	1.6E-7	6.3E-2	1.23	2.19
8192	3.5E-4	7 × 7	146	2.5E-11	2.5E-7	2045	2.1E-11	1.7E-7	7.1E-2	1.19	1.47
8192	3.5E-4	8 × 8	128	2.6E-11	2.5E-7	*dnc*	*dnc*	*dnc*	0.0	0.0	0.0

For general problems ($Q_A \neq I, Q_B \neq I$) the results differ depending on the target machine. For the IBM SP system (see Table 1), `PGESYCTD` is able to compete with the fully iterative method regarding both speed, accuracy and residual errors, for both well- and ill-conditioned problems. For example, `PGESYCTD` uses only 33% (well-conditioned case) and 25% (ill-conditioned case) of the execution time of `psb04md` for $m = n = 4096, P_r = 4, P_c = 4$. For the ill-conditioned problems, the difference in accuracy on the IBM SP system is remarkable: as expected, the explicitly blocked method gives far more accuracy than the fully iterative, which sometimes did not even converge. For the Linux-Cluster (see Table 2), `psb04md` is about two or three times faster than `PGESYCTD` for both types of problems. This difference in speed can be explained by the different characteristics of the machines, see Table 4. For uniprocessor runs `psb04md` is faster on both machines, but when we go in parallel the superior memory and network bandwidth of the SP system makes it possible for `PGESYCTD` to scale much better than `psb04md`. On the less balanced Super Linux Cluster the heavy communication in `PGESYCTD` becomes a bottleneck.

For the ill-conditioned problems `PGESYCTD` gives the best accuracy, both regarding the forward error $\|X - \tilde{X}\|_F$ and the residual error $\|R\|_F$ up to magnitudes 6 and 12, respectively. For the well-conditioned problems, the routines in general have forward and residual errors of the same order, even if `psb04md` shows slightly better forward error (up to 63% for the largest problems on the Linux Cluster ($m = n = 4096, 8192$)), and `PGESYCTD` shows slightly better residual error (up to 35% for the largest problems

Table 4. Hardware characteristics for Super Linux Cluster and IBM SP System. The parameter t_a denotes the time for performing an arithmetic operation, t_s denotes the experimentally measured startup time for message passing, t_n denotes the time to transfer one byte over a single link in the network and t_m denotes the peak time to transfer one byte through the memory of a node. The SP system has 3 times better flop/network bandwidth ratio and over 12 times better flop/memory bandwidth ratio than the Super Linux Cluster.

Hardware	Super Linux Cluster	IBM SP System	Parameter	Super Linux Cluster	IBM SP System
CPU	120 × 2 Athlon MP2k+	64 thin P2SC	t_a	3.0×10^{-10} sec.	2.1×10^{-9} sec.
&	1.667Ghz nodes,	120 MHz nodes,	t_s	3.7×10^{-6} sec.	4.0×10^{-5} sec.
Memory	1-4 Gb/node,	128 Mb/node,	t_n	3.0×10^{-9} sec.	6.7×10^{-9} sec.
	peak 800 Gflops/sec.	peak 33.6 Gflops/sec.	t_m	9.6×10^{-10} sec.	5.6×10^{-10} sec.
Network	Wolfkit3 SCI h s i,	Multistage network,	t_a/t_s	8.1×10^{-5}	5.3×10^{-5}
	3-dim. torus,	peak 150 Mb/sec.	t_a/t_n	0.10	0.31
	peak 667 Mb/sec.		t_a/t_m	0.31	3.8

on the Linux Cluster). We remark that the column sep^{-1} in Tables 1–3 are lower bounds on the exact values.

6 Summary and future work

We have presented a comparison of parallel ScaLAPACK-style implementations of two different methods for solving the continuous-time Sylvester equation (1), the Bartels-Stewart method and the iterative matrix-sign-function-based method. The comparison has dealt with generality, speed and accuracy.

Experiments carried out on two different distributed memory machines show that the parallel explicitly blocked Bartels-Stewart algorithm can solve more general problems and delivers far more accuracy for ill-conditioned problems. A method that imposes more restrictions on the spectra on A and B in $AX - XB = C$ is considered less general. Ill-conditioning is measured with respect to the separation of A and B, $\text{sep}(A, B)$, as defined in equation (8). We remark that $\text{sep}(A, B)$ can be much smaller than the minimum distance between the eigenvalues of A and B. This means that we can have an ill-conditioned problem even if the spectra of A and B are well-separated, which for some examples could favour the iterative matrix-sign-function-based method. The Bartels-Stewart method is also up to four times faster for large enough problems on the most balanced parallel platform (IBM SP), while the parallel iterative algorithm is almost always the fastest of the two on the less balanced platform (HPC2N Linux Super Cluster).

Ongoing work includes implementing general Bartels–Stewart solvers for related matrix equations, e.g., the continuous-time Lyapunov equation $AX + XA^T = C$, $C = C^T$ and the discrete-time Sylvester equation $AXB^T - X = C$. Our objective is to construct a software package $SCASY$ of ScaLAPACK-style algorithms for the most common matrix equations, including generalized forms of the Sylvester/Lyapunov equations.

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, 1972.
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. P. Benner, E.S. Quintana-Orti. Solving Stable Generalized Lyapunov Equations with the matrix sign functions, *Numerical Algorithms*, 20 (1), pp. 75-100, 1999.
4. P. Benner, E.S. Quitana-Orti, G. Quintana-Orti. Numerical Solution of Discrete Schur Stable Linear Matrix Equations on Multicomputers, *Parallel Alg. Appl.*, Vol. 17, No. 1, pp. 127-146, 2002.
5. P. Benner, E.S. Quitana-Orti, G. Quintana-Orti. Solving Stable Sylvester Equations via Rational Iterative Schemes, Preprint SFB393/04-08, TU Chemnitz, 2004.
6. 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.
7. R. Granat, A Parallel ScaLAPACK-style Sylvester Solver, *Master Thesis*, UMNAD 435/03, Dept. Computing Science, Umeå University, Sweden, January, 2003.
8. R. Granat, B. Kågström, P. Poromaa. Parallel ScaLAPACK-style Algorithms for Solving Continous-Time Sylvester Equations. In H. Kosch et al., Euro-Par 2003 Parallel Processing. *Lecture Notes in Computer Science*, Vol. 2790, pp. 800-809, 2003.
9. 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.
10. 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.
11. N.J. Higham. Perturbation Theory and Backward Error for $AX - XB = C$, *BIT*, 33:124–136, 1993.
12. 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.
13. 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.
14. Niconet Task II: Model Reduction, website: www.win.tue.nl/niconet/NIC2/NICtask2.html
15. 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.
16. J.D. Roberts. Linear model reduction and solution of the algebraic Ricatti equation by use of the sign function, *Intern. J. Control*, 32:677-687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).
17. SLICOT library in the Numerics in Control Network (NICONET), website: www.win.tue.nl/niconet/index.html