Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part II: Two-Sided and Generalized Sylvester and Lyapunov Equations [1]

Isak Jonsson and Bo Kågström [2]

September 2001 [3]

[1] This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001-Leuven-Heverlee, BELGIUM. This report is also available by anonymous ftp from `wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2001-5.ps.Z`
[2] Department of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden, {isak,bokg}@cs.umu.se
[3] Also published as Report UMINF-01.06, 2001, ISSN-0348-0542, September 2001

Abstract

We continue our study on high-performance algorithms for solving triangular matrix equations. They appear naturally in different condition estimation problems for matrix equations and various eigenspace computations, and as reduced systems in standard algorithms. Building on our successful recursive approach applied to one-sided matrix equations (Part I), we now present recursive blocked algorithms for two-sided matrix equations, which include matrix product terms such as AXB^T. Examples are the discrete-time standard and generalized Sylvester and Lyapunov equations. The means for high-performance are the recursive variable blocking, which has the potential of matching the memory hierarchies of today's high-performance computing systems, and level 3 computations which mainly are performed as GEMM operations. Different implementation issues are discussed, focusing on similarities and differences between one-sided and two-sided matrix equations. We present uniprocessor and SMP parallel performance results of recursive blocked algorithms and routines in the state-of-the-art SLICOT library. The performance improvements of our recursive algorithms are remarkable, including 10-folded speedups or more, compared to standard algorithms.

Keywords: Matrix equations, standard discrete-time Sylvester and Lyapunov, generalized Sylvester and Lyapunov, recursion, automatic blocking, superscalar, GEMM-based, level 3 BLAS, SMP parallelization

Contents

1 Introduction

We continue the presentation of recursive blocked algorithms for solving various types of triangular matrix equations. Our goal is to design efficient algorithms for today's HPC systems with multi-level memory hierarchies. The hierarchical recursive blocking allows for good data locality and combined with our highly optimized superscalar kernels, we obtain a notable boost in performance compared to existing algorithms as implemented in state-of-the-art libraries [1, 39]. In Part I [22], we introduced and discussed new recursive blocked algorithms for *one-sided* Sylvester-type equations, including the continuous-time standard Sylvester and Lyapunov equations, and a generalized coupled Sylvester equation.

In this contribution (Part II), we introduce and discuss new recursive blocked algorithms for *two-sided* matrix equations, which include matrix product terms of type $\mathrm{op}(A)X\mathrm{op}(B)$, where $\mathrm{op}(Y)$ can be Y or its transpose, Y^T. Examples include discrete-time standard and generalized Sylvester and Lyapunov equations. The standard methods for solving two-sided matrix equations are also based on the Bartels-Stewart method [3]. As in Part I, we focus on the solution of the two-sided triangular counterparts, which typically are obtained after an initial transformation of matrices (or regular matrix pairs) to Schur (or generalized Schur) form [1, 5].

Before we go into any further details, we outline the contents of the rest of the paper. In Section 2, we introduce Sep-functions associated with the two-sided matrix equations and re-establish the relationship between the solution of triangular matrix equations and condition estimation. Section 3 introduces our recursive blocked algorithms for two-sided matrix equations, including the discrete-time Sylvester (Section 3.1) and Lyapunov (Section 3.2) equations, and a generalized Sylvester equation (Section 3.3). We end this section by introducing recursive blocked algorithms for generalized Lyapunov equations, both the continuous-time case (Section 3.4) and the discrete-time case (Section 3.5). In Section 4, we revisit our discussion about different implementation issues, including the design of optimized two-sided matrix product kernels, which are critical for obtaining high performance. Sample performance results of our recursive blocked algorithms are presented and discussed in Section 5. Finally, we give some concluding remarks in Section 6.

2 Condition Estimation of Matrix Equations Revisited

The two-sided matrix equations can also be written as a linear system of equations $Zx = c$, where Z is a *Kronecker product matrix representation* of the associated matrix equation operator. The solution x and the right hand side c are represented in $\mathrm{vec}(\cdot)$ notation, where $\mathrm{vec}(X)$ denotes a column vector with the columns of X stacked on top of each other.

We introduce the following Z-matrices:

$$
\begin{aligned}
Z_{\mathrm{SYDT}} &= Z_{AXB^T-X} = B \otimes A - I_n \otimes I_m, \\
Z_{\mathrm{LYDT}} &= Z_{AXA^T-X} = A \otimes A - I_n \otimes I_n, \\
Z_{\mathrm{GSYL}} &= Z_{AXB^T-CXD^T} = B \otimes A - D \otimes C, \\
Z_{\mathrm{GLYCT}} &= Z_{AXE^T+EXA^T} = E \otimes A + A \otimes E, \\
Z_{\mathrm{GLYDT}} &= Z_{AXA^T-EXE^T} = A \otimes A - E \otimes E,
\end{aligned}
$$

where from top to bottom they represent the matrix operators of the discrete-time Sylvester and Lyapunov equations, the generalized (discrete/continuous-time) Sylvester equation, and the generalized continuous-time and discrete-time Lyapunov equations.

Similar techniques, as described in the Part I paper [22], can be used for estimating the conditioning of two-sided matrix equations. This leads to solving triangular two-sided matrix equations for estimating the Sep-function:

$$\mathrm{Sep}[\cdot] = \|Z^{-1}\|_2^{-1} = \sigma_{\min}(Z),$$

where Z is any of the Kronecker product matrices above. The Sep-functions associated with the two-sided matrix equations considered are:

$$\begin{array}{rcl}
\mathrm{Sep}[\mathrm{SYDT}] = & \inf_{\|X\|_F=1} \|AXB^T - X\|_F & = \sigma_{\min}(Z_{\mathrm{SYDT}}), \\
\mathrm{Sep}[\mathrm{LYDT}] = & \inf_{\|X\|_F=1} \|AXA^T - X\|_F & = \sigma_{\min}(Z_{\mathrm{LYDT}}), \\
\mathrm{Sep}[\mathrm{GSYL}] = & \inf_{\|X\|_F=1} \|AXB^T - CXD^T\|_F & = \sigma_{\min}(Z_{\mathrm{GSYL}}), \\
\mathrm{Sep}[\mathrm{GLYCT}] = & \inf_{\|X\|_F=1} \|AXE^T - EX(-A^T)\|_F & = \sigma_{\min}(Z_{\mathrm{GLYCT}}), \\
\mathrm{Sep}[\mathrm{GLYDT}] = & \inf_{\|X\|_F=1} \|AXA^T - EXE^T\|_F & = \sigma_{\min}(Z_{\mathrm{GLYDT}}).
\end{array}$$

One-norm Sep^{-1}-estimators based on the techniques of LAPACK [16, 18, 1] are implemented in the SLICOT library [39]. Each estimator involves solving several (around five) triangular matrix equations. Therefore, it is important that these triangular matrix equation problems can be solved efficiently on today's memory tiered systems.

The underlying perturbation theory for standard and generalized matrix equations is presented in [19] and [23], respectively.

3 Recursive Blocked Algorithms for Two-Sided Matrix Equations

As mentioned in the introduction, the standard methods for solving two-sided matrix equations are also based on the Bartels-Stewart method [3]. There is a quite extensive literature on the solution of matrix equations, and we refer to the following selection of fundamental papers [3, 10, 17, 4, 29, 8, 28, 36] that all present reliable algorithms. Our recursive approach also applies to triangular two-sided matrix equations, which is the topic of this section.

Some of the matrix equations considered can be seen as special cases of other formulations. However, this is mainly of theoretical interest, since either these equivalences include matrix inversion (when transforming a generalized matrix equation to a standard counterpart), or the matrix equations have symmetry structure that we want to utilize in the algorithms.

As in Part I, we define recursive splittings for each matrix equation which in turn lead to a few similar subproblems to be solved. These splittings are recursively applied to all "half-sized" triangular matrix equations. We end the recursion when the new problem sizes (M and/or N) are smaller than a certain block size, $blks$, which is chosen such that at least a few submatrices involved in the current matrix equation fit in the first level cache memory. For the solution of the small-sized problems, we apply our new high-performance kernels based on standard algorithms (see Section 4.2).

We also present Matlab-style functions for our recursive blocked solvers. We remark that all updates with respect to the solution of subproblems in the recursion are GEMM-rich operations of the type

$$C \leftarrow \beta C + \alpha \mathrm{op}(A) X \mathrm{op}(B)^T,$$

where α and β are real scalars, and op(Y) is Y or Y^T. A and/or B can be dense or triangular. This is due to the "two-sidedness" of the matrix equations. We refer to Section 4.1 for a discussion of the design of a high-performance implementation of this operation. In the algorithm descriptions, we use the function $[C] = \mathbf{axb}(A, X, B)$, which implements the matrix-product operation $C = C + AXB$. Other functions, including Level 3 BLAS [6, 7, 25, 26] operations are introduced the first time they are used in the algorithm descriptions.

3.1 Recursive triangular discrete-time Sylvester solvers

Consider the real *discrete-time Sylvester* (SYDT) matrix equation

$$AXB^T - X = C, \tag{1}$$

where A $(M \times M)$and B $(N \times N)$ are upper triangular or quasi-upper triangular, i.e., in real Schur form. The right hand side C and the solution X are of size $M \times N$, and typically, the solution overwrites the right hand side ($C \leftarrow X$). The SYDT equation (1) has a *unique solution* if and only if A and B^{-1} (or A^{-1} and B) have disjoint spectra, i.e., $\lambda_i(A)\lambda_j(B) \neq 1$ for all i and j. This follows from the definition of Sep[SYDT] in Section 2.

Depending on the values of M and N, we consider three alternatives for doing a *recursive splitting.*

Case 1 ($1 \leq N \leq M/2$). We split A by rows and columns, and C by rows only:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} B^T - \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix},$$

or equivalently

$$\begin{aligned} A_{11}X_1B^T - X_1 &= C_1 - A_{12}X_2B^T, \\ A_{22}X_2B^T - X_2 &= C_2. \end{aligned}$$

The original problem is split in two triangular discrete-time Sylvester equations. First, we solve for X_2 and after updating C_1 with respect to X_2, we can solve for X_1.

Case 2 ($1 \leq M \leq N/2$). We split B by rows and columns, and C by columns only:

$$A \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} B_{11}^T & \\ B_{12}^T & B_{22}^T \end{bmatrix} - \begin{bmatrix} X_1 & X_2 \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \end{bmatrix},$$

or equivalently

$$\begin{aligned} AX_1B_{11}^T - X_1 &= C_1 - AX_2B_{12}^T, \\ AX_2B_{22}^T - X_2 &= C_2. \end{aligned}$$

As in Case 1, we first solve for X_2 and then after updating the right hand side of the first equation, X_1 can be solved for.

Case 3 ($N/2 \leq M \leq 2N$). We split A, B and C by rows and columns:

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

This recursive splitting results in the following four triangular discrete-time Sylvester equations:

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

We start by solving for X_{22} in the fourth equation. After updating C_{12} and C_{21} with respect to X_{22}, we can solve for X_{12} and X_{21}. Both updates and the triangular Sylvester solves are independent operations and can be executed concurrently. Finally, after updating C_{11} with respect to X_{12}, X_{21} and X_{22}, we solve for X_{11}. We remark that two of the matrix multiply operations can be combined in one large GEMM operation:

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

Alternatively, the update of C_{11} can be performed as

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

Algorithm 1: rtrsydt

Input: A $(M \times M)$ and B $(N \times N)$ in quasi-triangular (Schur) form. C $(M \times N)$ dense matrix. $blks$, block size that specifies when to switch to a standard algorithm for solving small-sized triangular matrix equations. $uplo$ indicates triangular form of A, B: 1–(upper, upper), 2–(upper, lower), 3–(lower, upper), 4–(lower, lower).

Output: X $(M \times N)$, the solution of $AXB^T - X = C$. X is allowed to overwrite C.

```
function [X] = rtrsydt(A, B, C, uplo, blks)
if 1 ≤ M, N ≤ blks then
    X = trsydt(A, B, C, uplo);
else switch uplo
case 1
    if 1 ≤ N ≤ M/2   % Case 1: Split A (by rows and columns), C (by rows only)
        X_2 = rtrsydt(A_22, B, C_2, 1, blks);
        C_1 = axb(−A_12, X_2, B^T, C_1);
        X_1 = rtrsydt(A_11, B, C_1, 1, blks);
        X = [X_1; X_2];
    elseif 1 ≤ M ≤ N/2   % Case 2: Split B (by rows and columns), C (by columns only)
        X_2 = rtrsydt(A, B_22, C_2, 1, blks);
        C_1 = axb(−A, X_2, B_12^T, C_1);
        X_1 = rtrsydt(A, B_11, C_1, 1, blks);
        X = [X_1, X_2];
    else   % M, N > blks   Case 3: Split A, B and C (all by rows and columns)
        X_22 = rtrsydt(A_22, B_22, C_22, 1, blks);
        C_12 = axb(−A_12, X_22, B_22^T, C_12); C_21 = axb(−A_22, X_22, B_12^T, C_21);
        X_12 = rtrsydt(A_11, B_22, C_12, 1, blks); X_21 = rtrsydt(A_22, B_11, C_21, 1, blks);
        if M < N then
            C_11 = axb(−A_12, [ X_21  X_22 ], [ B_11  B_12 ]^T, C_11);
            C_11 = axb(−A_11, X_12, B_12^T, C_11);
        else
            C_11 = axb(−[ A_11  A_12 ], [ X_12^T  X_22^T ]^T, B_12^T, C_11);
            C_11 = axb(−A_12, X_21, B_11^T, C_11);
        end
        X_11 = rtrsydt(A_11, B_11, C_11, 1, blks);
        X = [X_11, X_12; X_21, X_22];
    end
case 2   % Code for uplo = 2.
case 3   % Code for uplo = 3.
case 4   % Code for uplo = 4.
end
```

Algorithm 1: Recursive blocked algorithm for solving the triangular discrete-time Sylvester equation.

In the discussion above, we have assumed that both A and B are upper triangular (or quasi-triangular). However, it is straightforward to derive similar recursive splittings for the triangular SYDT, where A and B, each of them can be in either upper or lower Schur form.

We present a Matlab-style function $[X] = \mathbf{rtrsydt}(A, B, C, uplo, blks)$ for our recursive blocked solver in Algorithm 1. The input *uplo* signals the triangular structure of A and B. The function $[X] = \mathbf{trsydt}(A, B, C, uplo)$ implements an algorithm for solving triangular Sylvester block kernel problems (see Section 4.2).

3.2 Recursive triangular discrete-time Lyapunov solvers

Consider the real *discrete-time Lyapunov* (LYDT) or *Stein* matrix equation

$$AXA^T - X = C, \tag{2}$$

where A is upper triangular or quasi-upper triangular, i.e., in real Schur form. The right hand side C, the solution X and A are all of size $N \times N$. Typically, the solution overwrites the right hand side ($C \leftarrow X$). If C is symmetric, then X is symmetric as well, and the Stein matrix equation corresponds to the discrete-time standard algebraic Lyapunov equation. The LYDT equation (2) has a *unique solution* if and only if $\lambda_i \lambda_j \neq 1$ for all i and j. This follows from the definition of Sep[LYDT] in Section 2. If C is (semi)definite and $|\lambda_i| < 1$ for all i, then a unique (semi)definite solution exists [17].

Since A and C are of the same size, there is only one way of doing the *recursive splitting*:

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

If $C = C^T$, then $X_{21} = X_{12}^T$ and the recursive splitting leads to three triangular matrix equations:

$$\begin{aligned} A_{11}X_{11}A_{11}^T - X_{11} &= C_{11} - A_{12}X_{12}^T A_{11}^T - A_{11}X_{12}A_{12}^T - A_{12}X_{22}A_{12}^T, \\ A_{11}X_{12}A_{22}^T - X_{12} &= C_{12} - A_{12}X_{22}A_{22}^T, \\ A_{22}X_{22}A_{22}^T - X_{22} &= C_{22}. \end{aligned}$$

The first and the third are standard Stein-type equations, while the second is a triangular discrete-time Sylvester equation. We start by solving for X_{22} in the third equation. After updating C_{12} with respect to X_{22}, we can solve for X_{12}. Finally, we update C_{11} with respect to X_{12} and X_{22}, and solve for X_{11}.

The update of C_{11} is a symmetric rank-$2k$ (SYR2K) update

$$C_{11} = C_{11} - (A_{11}X_{12})A_{12}^T - A_{12}(A_{11}X_{12})^T,$$

where $A_{11}X_{12}$ is a triangular matrix multiply (TRMM) operation, followed by the GEMM-rich update

$$C_{11} = C_{11} - A_{12}X_{22}A_{12}^T,$$

which is performed by the SLICOT MB01RD subroutine, see Section 4.1.

In Algorithm 2, we present a Matlab-style function $[X] = \mathbf{rtrlydt}(A, C, blks)$ for our recursive blocked solver, which deals with the cases with a symmetric or nonsymmetric C. The functions $[C] = \mathbf{syr2k}(A, B, C)$ and $[X] = \mathbf{trmm}(A, C)$ implement the SYR2K-operation $C = C + AB^T + BA^T$, and a triangular matrix multiply $X = AC$, where one of the matrix arguments (A or C) is in triangular form, respectively. The function $[X] = \mathbf{trlydt}(A, C)$ implements an algorithm for solving Lyapunov kernel problems (see Section 4.2). For solving the triangular discrete-time Sylvester equations that appear, we make use of the recursive algorithm $[X] = \mathbf{rtrsydt}(A, B, C, uplo, blks)$, described in Section 3.1.

3.3 Recursive triangular generalized Sylvester solvers

Consider the real *generalized Sylvester* (GSYL) matrix equation

$$AXB^T - CXD^T = E, \tag{3}$$

Algorithm 2: rtrlydt

Input: A $(N \times N)$ in upper quasi-triangular (Schur) form. C $(N \times N)$ dense matrix. $blks$, block size that specifies when to switch to a standard algorithm for solving small-sized matrix equations.

Output: X $(N \times N)$, the solution of $AXA^T - X = C$. X is allowed to overwrite C, and is symmetric if $C = C^T$ on entry.

function $[X] =$ **rtrlydt**$(A, C, blks)$
if $1 \le N \le blks$ **then**
  $X =$ **trlydt**(A, C);
elseif C *is symmetric*
  % Split A and C (by rows and columns)
  $X_{22} =$ **rtrlydt**$(A_{22}, C_{22}, blks)$;
  $C_{12} =$ **axb**$(-A_{12}, X_{22}, A_{22}^T, C_{12})$;
  $X_{12} =$ **rtrsydt**$(A_{11}, A_{22}, C_{12}, 1, blks)$; $X_{21} = X_{12}^T$;
  $C_{11} =$ **syr2k**(**trmm**$(A_{11}, X_{12}), -A_{12}, C_{11})$;
  $C_{11} =$ **axb**$(-A_{12}, X_{22}, A_{12}^T, C_{11})$;
  $X_{11} =$ **rtrlydt**$(A_{11}, C_{11}, blks)$;
  $X = [X_{11}, X_{12};\ X_{21}, X_{22}]$;
else
  $X =$ **rtrsydt**$(A, A, C, 1, blks)$;
end

Algorithm 2: Recursive blocked algorithm for solving the triangular discrete-time Lyapunov or Stein equation.

where (A, C) of size $M \times M$ and (B, D) of size $N \times N$ are in generalized Schur form, i.e., A and B are upper quasi-triangular and C, B are upper triangular. The right hand side E and the solution X are of size $M \times N$, and typically, the solution overwrites the right hand side $(E \leftarrow X)$. The GSYL equation (3) has a *unique solution* if and only if $A - \lambda C$ and $D - \lambda B$ are regular and have disjoint spectra [4]. This follows also from the definition of Sep[GSYL] in Section 2.

We see that (3) is a generalization of the continuous-time Sylvester equation ($B = I_N$ and $C = I_M$), as well as, the discrete-time Sylvester equation ($C = I_M$ and $D = I_N$). As for these Sylvester solvers, we get three alternatives for doing a *recursive splitting*.

Case 1 $(1 \le N \le M/2)$. We split (A, C) by rows and columns, and E by rows only:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} B^T - \begin{bmatrix} C_{11} & C_{12} \\ & C_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} D^T = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix},$$

or equivalently

$$\begin{aligned} A_{11}X_1B^T - C_{11}X_1D^T &= E_1 - A_{12}X_2B^T + C_{12}X_2D^T, \\ A_{22}X_2B^T - C_{22}X_2D^T &= E_2. \end{aligned}$$

The original problem is split in two triangular generalized Sylvester equations. First, we solve for X_2 and after updating E_1 with respect to X_2, we can solve for X_1.

Case 2 $(1 \le M \le N/2)$. We split (B, D) by rows and columns, and E by columns only:

$$A \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} B_{11}^T & \\ B_{12}^T & B_{22}^T \end{bmatrix} - C \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} D_{11}^T & \\ D_{12}^T & D_{22}^T \end{bmatrix} = \begin{bmatrix} E_1 & E_2 \end{bmatrix},$$

or equivalently

$$AX_1B_{11}^T - CX_1D_{11}^T = E_1 - AX_2B_{12}^T + CX_2D_{12}^T,$$

$$AX_2B_{22}^T - CX_2D_{22}^T \quad = \quad E_2.$$

As in Case 1, we first solve for X_2 and then after updating the right hand side of the first equation, X_1 can be solved for.

Case 3 ($N/2 \leq M \leq 2N$). We split (A, C), (B, D) and E by rows and columns:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} B_{11}^T & \\ B_{12}^T & B_{22}^T \end{bmatrix} -$$

$$\begin{bmatrix} C_{11} & C_{12} \\ & C_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} D_{11}^T & \\ D_{12}^T & D_{22}^T \end{bmatrix} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}.$$

This recursive splitting results in the following four triangular generalized Sylvester equations:

$$\begin{aligned} A_{11}X_{11}B_{11}^T - C_{11}X_{11}D_{11}^T &= E_{11} - A_{12}X_{21}B_{11}^T - (A_{11}X_{12} + A_{12}X_{22})B_{12}^T + \\ & \quad C_{12}X_{21}D_{11}^T + (C_{11}X_{12} + C_{12}X_{22})D_{12}^T, \\ A_{11}X_{12}B_{22}^T - C_{11}X_{12}D_{22}^T &= E_{12} - A_{12}X_{22}B_{22}^T + C_{12}X_{22}D_{22}^T, \\ A_{22}X_{21}B_{11}^T - C_{22}X_{21}D_{11}^T &= E_{21} - A_{22}X_{22}B_{12}^T + C_{22}X_{22}D_{12}^T, \\ A_{22}X_{22}B_{22}^T - C_{22}X_{22}D_{22}^T &= E_{22}. \end{aligned}$$

We start by solving for X_{22} in the fourth equation. After updating E_{12} and E_{21} with respect to X_{22}, we can solve for X_{12} and X_{21}. Both updates and the triangular generalized Sylvester solves are independent operations and can be executed concurrently. Finally, after updating E_{11} with respect to X_{12}, X_{21} and X_{22}, we solve for X_{11}. Some of the updates of E_{11} can be combined in larger GEMM operations as we did for the SYDT equation. We obtain

$$\begin{aligned} E_{11} &= E_{11} - A_{12}X_{21}B_{11}^T + C_{12}X_{21}D_{11}^T - \\ & \quad (\begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \begin{bmatrix} X_{12} \\ X_{22} \end{bmatrix})B_{12}^T + (\begin{bmatrix} C_{11} & C_{12} \end{bmatrix} \begin{bmatrix} X_{12} \\ X_{22} \end{bmatrix})D_{12}^T. \end{aligned}$$

In Algorithm 3, a Matlab-style function $[X] =$ **rtrgsyl**$(A, B, C, D, E, blks)$ for our recursive blocked solver is presented. The algorithm also shows an alternative way to combine the matrix multiply operations into large GEMM operations, and the choice of which one to use is dependent on the sizes of M and N. The function $[X] =$ **trgsyl**(A, B, C, D, E) implements an algorithm for solving triangular generalized Sylvester kernel problems (see Section 4.2).

3.4 Recursive triangular generalized continuous-time Lyapunov solvers

Consider the real *generalized continuous-time Lyapunov* (GLYCT) matrix equation

$$AXE^T + EXA^T = C, \tag{4}$$

where A and E of size $N \times N$ are upper quasi-triangular and upper triangular, respectively. In other words, (A, E) is in generalized Schur form. The right hand side C and the solution X are of size $N \times N$, and typically, the solution overwrites the right hand side ($C \leftarrow X$). If C is symmetric, then X is symmetric as well. By choosing $E = I_N$ in (4), we get the standard continuous-time Lyapunov equation.

The GLYCT equation (4) has a *unique symmetric solution* if and only if $C = C^T$, and all eigenvalues λ_i of $A - \lambda E$ are finite with $\lambda_i + \lambda_j \neq 0$ for all i and j. This follows from the definition of Sep[GLYCT] in Section 2. If C is (semi)definite and $\mathrm{Re}\lambda_i < 0$ for all i, then a unique (semi)definite solution exists [36].

Since all matrices are square, we only have one way of doing a *recursive splitting*. We split A, E and C by rows and columns:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} E_{11}^T & \\ E_{12}^T & E_{22}^T \end{bmatrix} +$$

Algorithm 3: rtrgsyl

Input: (A, C) $(M \times M)$ and (B, D) $(N \times N)$ in upper generalized Schur form. E $(M \times N)$ dense matrix. $blks$, block size that specifies when to switch to a standard algorithm for solving small-sized triangular generalized Sylvester equations.

Output: X $(M \times N)$, the solution of $AXB^T - CXD^T = E$. X is allowed to overwrite E.

function $[X] = $ **rtrgsyl**$(A, B, C, D, E, blks)$
if $1 \le M, N \le blks$ **then**
  $X = $ **trgsyl**(A, B, C, D, E);
elseif $1 \le N \le M/2$ % Case 1: Split (A, C) (by rows and columns), E (by rows only)
  $X_2 = $ **rtrgsyl**$(A_{22}, B, C_{22}, D, E_2, blks)$;
  $E_1 = $ **axb**$(-A_{12}, X_2, B^T, E_1)$; $E_1 = $ **axb**(C_{12}, X_2, D^T, E_1);
  $X_1 = $ **rtrgsyl**$(A_{11}, B, C_{11}, D, E_1, blks)$;
  $X = [X_1; X_2]$;
elseif $1 \le M \le N/2$ % Case 2: Split (B, D) (by rows and columns), E (by columns only)
  $X_2 = $ **rtrgsyl**$(A, B_{22}, C, D_{22}, E_2, blks)$;
  $E_1 = $ **axb**$(-A, X_2, B_{12}^T, E_1)$; $E_1 = $ **axb**(C, X_2, D_{12}^T, E_1);
  $X_1 = $ **rtrgsyl**$(A, B_{11}, C, D_{11}, E_1, blks)$;
  $X = [X_1, X_2]$;
else % $M, N > blks$ Case 3: Split (A, C), (B, D) and E (all by rows and columns)
  $X_{22} = $ **rtrgsyl**$(A_{22}, B_{22}, C_{22}, D_{22}, E_{22}, blks)$;
  $E_{12} = $ **axb**$(-A_{12}, X_{22}, B_{22}^T, E_{12})$; $E_{12} = $ **axb**$(C_{12}, X_{22}, D_{22}^T, E_{12})$;
  $E_{21} = $ **axb**$(-A_{22}, X_{22}, B_{12}^T, E_{21})$; $E_{21} = $ **axb**$(C_{22}, X_{22}, D_{12}^T, E_{21})$;
  $X_{12} = $ **rtrgsyl**$(A_{11}, B_{22}, C_{11}, D_{22}, E_{12}, blks)$;
  $X_{21} = $ **rtrgsyl**$(A_{22}, B_{11}, C_{22}, D_{11}, E_{21}, blks)$;
  if $M < N$ **then**
    $E_{11} = $ **axb**$(-A_{11}, X_{12}, B_{12}^T, E_{11})$; $E_{11} = $ **axb**$(C_{11}, X_{12}, D_{12}^T, E_{11})$;
    $E_{11} = $ **axb**$(-A_{12}, \begin{bmatrix} X_{21} & X_{22} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} \end{bmatrix}^T, E_{11})$;
    $E_{11} = $ **axb**$(C_{12}, \begin{bmatrix} X_{21} & X_{22} \end{bmatrix}, \begin{bmatrix} D_{11} & D_{12} \end{bmatrix}^T, E_{11})$;
  else
    $E_{11} = $ **axb**$(-A_{12}, X_{21}, B_{11}^T, E_{11})$; $E_{11} = $ **axb**$(C_{12}, X_{21}, D_{11}^T, E_{11})$;
    $E_{11} = $ **axb**$(-\begin{bmatrix} A_{11} & A_{12} \end{bmatrix}, \begin{bmatrix} X_{12}^T & X_{22}^T \end{bmatrix}^T, B_{12}^T, E_{11})$;
    $E_{11} = $ **axb**$(\begin{bmatrix} C_{11} & C_{12} \end{bmatrix}, \begin{bmatrix} X_{12}^T & X_{22}^T \end{bmatrix}^T, D_{12}^T, E_{11})$;
  end
  $X_{11} = $ **rtrgsyl**$(A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, blks)$;
  $X = [X_{11}, X_{12}; X_{21}, X_{22}]$;
end

Algorithm 3: Recursive blocked algorithm for solving the triangular generalized Sylvester equation.

$$\begin{bmatrix} E_{11} & E_{12} \\ & E_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} A_{11}^T & \\ A_{12}^T & A_{22}^T \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

Since $X_{21} = X_{12}^T$, the recursive splitting results in three triangular generalized continuous-time Lyapunov equations:

$$\begin{aligned} A_{11}X_{11}E_{11}^T + E_{11}X_{11}A_{11}^T &= C_{11} - A_{12}X_{12}^T E_{11}^T - (A_{11}X_{12} + A_{12}X_{22})E_{12}^T - \\ &\quad E_{12}X_{12}^T A_{11}^T - (E_{11}X_{12} + E_{12}X_{22})A_{12}^T, \\ A_{11}X_{12}E_{22}^T + E_{11}X_{12}A_{22}^T &= C_{12} - A_{12}X_{22}E_{22}^T - E_{12}X_{22}A_{22}^T, \\ A_{22}X_{22}E_{22}^T + E_{22}X_{22}A_{22}^T &= C_{22}. \end{aligned}$$

We start by solving for X_{22} in the third equation. After updating C_{12} with respect to X_{22}, we can solve for X_{12}. Finally, after updating C_{11} with respect to X_{12} and X_{22}, we solve for X_{11}. We remark that the update of C_{11} includes two SYR2K operations, namely

$$\begin{aligned} C_{11} &= C_{11} - (A_{11}X_{12} + A_{12}X_{22})E_{12}^T - E_{12}(A_{11}X_{12} + A_{12}X_{22})^T, \\ C_{11} &= C_{11} - (E_{11}X_{12})A_{12}^T - A_{12}(E_{11}X_{12})^T, \end{aligned}$$

where $A_{11}X_{12}$ and $E_{11}X_{12}$ are TRMM operations and $A_{12}X_{22}$ is a GEMM operation.

Algorithm 4: rtrglyct

Input: (A, E) $(N \times N)$ in upper generalized Schur form. C $(N \times N)$ dense matrix. $blks$, block size that specifies when to switch to a standard algorithm for solving small-sized triangular generalized Lyapunov equations.

Output: X $(N \times N)$, the solution of $AXE^T + EXA^T = C$. X is allowed to overwrite C, and is symmetric if $C = C^T$ on entry.

function $[X] =$ **rtrglyct**$(A, E, C, blks)$
if $1 \leq N \leq blks$ **then**
    $X =$ **trglyct**(A, E, C);
elseif C *is symmetric*
    % Split (A, E), and C (all by rows and columns)
    $X_{22} =$ **rtrglyct**$(A_{22}, E_{22}, C_{22}, blks)$;
    $C_{12} =$ **axb**$(-A_{12}, X_{22}, E_{22}^T, C_{12})$; $C_{12} =$ **axb**$(-E_{12}, X_{22}, A_{22}^T, C_{12})$;
    $X_{12} =$ **rtrgsyl**$(A_{11}, E_{22}, E_{11}, A_{22}, C_{12}, blks)$; $X_{21} = X_{12}^T$;
    $C_{11} =$ **syr2k**(**trmm**(A_{11}, X_{12}) + **gemm**$(A_{12}, X_{22}, \emptyset)$, $-E_{12}, C_{11})$;
    $C_{11} =$ **syr2k**(**trmm**(E_{11}, X_{12}), $-A_{12}, C_{11})$;
    $X_{11} =$ **rtrglyct**$(A_{11}, E_{11}, C_{11}, blks)$;
    $X = [X_{11}, X_{12}; X_{21}, X_{22}]$;
else
    $X =$ **rtrgsyl**$(A, E, -E, A, C, blks)$;
end

Algorithm 4: Recursive blocked algorithm for solving the triangular generalized continuous-time Lyapunov equation.

In Algorithm 4, a Matlab-style function $[X] = \mathbf{rtrglyct}(A, E, C, blks)$ implementing our recursive blocked solver is presented. The function $[X] = \mathbf{trglyct}(A, E, C)$ implements an algorithm for solving triangular generalized Lyapunov kernel problems (see Section 4.2).

3.5 Recursive triangular generalized discrete-time Lyapunov solvers

Consider the real *generalized discrete-time Lyapunov* (GLYDT) matrix equation

$$AXA^T - EXE^T = C, \tag{5}$$

where A and E of size $N \times N$ are upper quasi-triangular and upper triangular, respectively. In other words, (A, E) is in generalized Schur form. The right hand side C and the solution X are of size $N \times N$, and typically, the solution overwrites the right hand side ($C \leftarrow X$). If C is symmetric, then X is symmetric as well. By choosing $E = I_N$ in (5), we get the standard discrete-time Lyapunov equation.

The GLYDT equation (5) has a *unique symmetric solution* if and only if $C = C^T$, and the eigenvalues λ_i of $A - \lambda E$ satisfy $\lambda_i \lambda_j \neq 1$ for all i and j (with the convention that $0 \cdot \infty = 1$). This follows from the definition of Sep[GLYDT] in Section 2. If C is (semi)definite and $|\lambda_i(A - \lambda E)| < 1$ for all i, then a unique (semi)definite solution exists [36].

Since all matrices are square, we only have one way of doing a *recursive splitting*. We split A, E and C by rows and columns:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} A_{11}^T & \\ A_{12}^T & A_{22}^T \end{bmatrix} -$$

$$\begin{bmatrix} E_{11} & E_{12} \\ & E_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} E_{11}^T & \\ E_{12}^T & E_{22}^T \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

Since $X_{21} = X_{12}^T$, the recursive splitting results in three triangular generalized discrete-time Lyapunov equations:

$$\begin{aligned} A_{11}X_{11}A_{11}^T - E_{11}X_{11}E_{11}^T &= C_{11} - A_{12}X_{12}^T A_{11}^T - (A_{11}X_{12} + A_{12}X_{22})A_{12}^T - \\ &\quad E_{12}X_{12}^T E_{11}^T - (E_{11}X_{12} + E_{12}X_{22})E_{12}^T, \\ A_{11}X_{12}A_{22}^T - E_{11}X_{12}E_{22}^T &= C_{12} - A_{12}X_{22}A_{22}^T - E_{12}X_{22}E_{22}^T, \\ A_{22}X_{22}A_{22}^T - E_{22}X_{22}E_{22}^T &= C_{22}. \end{aligned}$$

We start by solving for X_{22} in the third equation. After updating C_{12} with respect to X_{22}, we can solve for X_{12}. Finally, after updating C_{11} with respect to X_{12} and X_{22}, we solve for X_{11}.

As for the GLYCT equation (4), four of the two-sided matrix product updates of C_{11} can be expressed as two SYR2K operations:

$$C_{11} = C_{11} - (A_{11}X_{12})A_{12}^T - A_{12}(A_{11}X_{12})^T,$$

and

$$C_{11} = C_{11} - (E_{11}X_{12})E_{12}^T - E_{12}(E_{11}X_{12})^T,$$

where $A_{11}X_{12}$ and $E_{11}X_{12}$ are TRMM operations.

In Algorithm 5, a Matlab-style function $[X] = \mathbf{rtrglydt}(A, E, C, blks)$ implementing our recursive blocked solver is presented. The function $[X] = \mathbf{trglydt}(A, E, C)$ implements an algorithm for solving triangular generalized discrete-time Lyapunov kernel problems (see Section 4.2). For solving the triangular generalized Sylvester equations that appear, we make use of the recursive algorithm $[X] = \mathbf{rtrgsyl}(A, B, C, D, E, blks)$, described in Section 3.3.

3.6 Number of operations and execution order

In Table 1, we summarize the complexity measured in *floating point operations* (flops) of the standard elementwise explicit algorithms for solving two-sided triangular matrix equations (e.g., see [3, 4, 8, 36]). They are all based on backward or forward substitutions with one or several right hand sides. We remark that the difference in flops between the two extreme cases, i.e., when all diagonal blocks of the matrices in upper Schur form are of size 1×1 or 2×2, respectively, only show up in the lower order terms.

Assuming $2N > M \geq N$, the flopcounts of the recursive blocked algorithms can be expressed in terms of the following recurrence formulas:

$$\begin{aligned} F_{\mathrm{SYDT}}(M,N) &= 4F_{\mathrm{SYDT}}(M/2,N/2) + M^2N + \tfrac{3}{4}MN^2, & (6) \\ F_{\mathrm{LYDT}}(N) &= 2F_{\mathrm{LYDT}}(N/2) + F_{\mathrm{SYDT}}(N/2,N/2) + \tfrac{9}{8}N^3, & (7) \\ F_{\mathrm{GSYL}}(M,N) &= 4F_{\mathrm{GSYL}}(M/2,N/2) + 2M^2N + \tfrac{3}{2}MN^2, & (8) \\ F_{\mathrm{GLYCT}}(N) &= 2F_{\mathrm{GLYCT}}(N/2) + F_{\mathrm{GSYL}}(N/2,N/2) + \tfrac{7}{4}N^3, & (9) \\ F_{\mathrm{GLYDT}}(N) &= 2F_{\mathrm{GLYDT}}(N/2) + F_{\mathrm{GSYL}}(N/2,N/2) + \tfrac{9}{4}N^3. & (10) \end{aligned}$$

Algorithm 5: rtrglydt

Input: (A, E) $(N \times N)$ in upper generalized Schur form. C $(N \times N)$ dense matrix. $blks$, block size that specifies when to switch to a standard algorithm for solving small-sized triangular generalized Lyapunov equations.

Output: X $(N \times N)$, the solution of $AXA^T - EXE^T = C$. X is allowed to overwrite C, and is symmetric if $C = C^T$ on entry.

```
function [X] = rtrglydt(A, E, C, blks)
if 1 ≤ N ≤ blks then
    X = trglydt(A, E, C);
elseif C is symmetric
    % Split (A, E), and C (all by rows and columns)
    X22 = rtrglydt(A22, E22, C22, blks);
    C12 = axb(−A12, X22, A22^T, C12); C12 = axb(−E12, X22, E22^T, C12);
    X12 = rtrgsyl(A11, A22, E11, E22, C12, blks); X21 = X12^T;
    C11 = syr2k(trmm(A11, X12), −A12, C11);
    C11 = syr2k(trmm(E11, X12), −E12, C11);
    C11 = axb(−A12, X22, A12^T, C11); C11 = axb(−E12, X22, E12^T, C11);
    X11 = rtrglydt(A11, E11, C11, blks);
    X = [X11, X12; X21, X22];
else
    X = rtrgsyl(A, A, E, E, C, blks);
end
```

Algorithm 5: Recursive blocked algorithm for solving the triangular generalized discrete-time Lyapunov equation.

Matrix equation	Overall cost in flops
SYDT (1)	$2M^2N + MN^2$ $(M \leq N)$
	$M^2N + 2MN^2$ $(M > N)$
LYDT (2)	$\frac{4}{3}N^3$
GSYL (3)	$4M^2N + 2MN^2$ $(M \leq N)$
	$2M^2N + 4MN^2$ $(M > N)$
GLYCT (4)	$\frac{8}{3}N^3$
GLYDT (5)	$\frac{8}{3}N^3$

Table 1: Complexity of standard algorithms measured in flops.

Matrix equation	Overall cost in flops	Flop ratio $(M = N)$
SYDT (1)	$\frac{6}{4}M^2N + 2MN^2$ $(M \leq N)$	1.1667
	$2M^2N + \frac{6}{4}MN^2$ $(M > N)$	
LYDT (2)	$\frac{25}{12}N^3$	1.5625
GSYL (3)	$3M^2N + 4MN^2$ $(M \leq N)$	1.1667
	$4M^2N + 3MN^2$ $(M > N)$	
GLYCT (4)	$\frac{21}{6}N^3$	1.3125
GLYDT (5)	$\frac{25}{6}N^3$	1.5625

Table 2: Complexity of recursive blocked algorithms measured in flops.

These expressions correspond to the most general case when the recursive splitting is by rows and by columns for all input matrices. We have collected the cost for all two-sided matrix multiplications (including the SYR2K operations) of each recursive algorithm in the last term(s) of the recurrences. The flopcounts take any triangular structure of the matrices in each two-sided matrix product into account (see Section 4.1). Ignoring the lower order terms, it is straightforward to derive the expressions in Table 2 from the equations (6), (7), (8), (9), and (10), respectively.

Our recursive blocked algorithms require both extra workspace and more flops compared to the standard elementwise algorithms. The extra workspace is needed in the evaluation of the two-sided matrix multiplications (see Section 4.1). In the third column of Table 2, the ratios between the flopcounts for the recursive blocked algorithms and the standard algorithms are listed. Despite the quite large flops penalties of the recursive blocked algorithms, they outperform the standard algorithms for large enough problems (see measured performance results in Section 5). This fact is mainly due to the difference in their data reference patterns, i.e., the order in which they access data and how big chunks and how many times the data is moved in the memory hierarchy of the target computer system. The cost of redundant memory transfers can be devastating to the algorithm performance.

4 Design and Implementation Issues

In Part I [22], we describe the three levels of triangular matrix equation solvers used in our implementations. Here, we recapitulate the properties of these solvers. At the user-level there are the recursive blocked **rtr*** *solvers*. Each of them calls a **tr*** *block* or *sub-system solver* when the current problem sizes (M and/or N) from a recursive splitting are smaller than a certain block size, $blks$. Finally, each of the block solvers call a *superscalar kernel* for solving matrix equations with $M, N \leq 4$.

The reader might want to see Part I for discussions on the impact and design of the sub-system solvers and superscalar kernels, block sizes, BLAS implementations and parallelization issues. In this section, the impact of the two-sidedness of the matrix equations on the different routines is discussed, together with the design of the superscalar kernels and the parallelization of the algorithms.

4.1 Design of optimized two-sided matrix product kernels

The update of the right-hand side of the equations is done by the **axb** operation, i.e., an optimized kernel which performs the two-sided matrix product operation

$$C \leftarrow \beta C + \alpha \mathrm{op}(A) X \mathrm{op}(B)^T,$$

where α and β are real scalars, and $\mathrm{op}(Y)$ is Y or Y^T. A and/or B can be dense or triangular. Besides, one or several of the three matrices can be symmetric. Unfortunately, these operations need extra workspace, which can be substantial (typically the size of the current right hand side).

For a draft version that implements the generic operation $C = C + AXB$ see Algorithm 6. Depending on the sizes of A, X, and B, we should perform the matrix product as $(\mathrm{op}(A)X)\mathrm{op}(B)$ or as $\mathrm{op}(A)(X\mathrm{op}(B))$. Here, it is important that the BLAS implementation provides proper blocking for the cases which lead to bad memory access patterns. If matrices are stored in column-major order, the order $(A^T X)B$ might give a significant better access pattern than $A^T(XB)$. Also, A and/or B might be quasi-trapezoidal and/or quasi-triangular. Examples of both cases are found in **rtrsydt**, Algorithm 1. We also want to make use of symmetry properties, e.g., see **rtrlydt**, Algorithm 2, where $C \leftarrow C - AXA^T$, $X = X^T$, is computed.

In Algorithm 7, we show an implementation for one of the four cases of $\mathrm{op}(A)X\mathrm{op}(B)^T$, which makes use of high-performance level 3 BLAS routines. In the algorithm, the costs of computing $(\mathrm{op}(A)X)\mathrm{op}(B)$

Algorithm 6: axb

Input: A $(M \times P)$, X $(P \times Q)$, B $(Q \times N)$ and C $(M \times N)$.

Output: C $(M \times N)$, the result of the matrix product $C = C + AXB$.

```
function [C] = axb(A, X, B, C)
if A, B dense then
    C = gemm(A, gemm(X, B, ∅), C);
elseif A triangular, B dense
    C = gemm(trmm(A, X), B, C);
elseif A dense, B triangular
    C = gemm(A, trmm(X, B), C);
elseif A triangular, B triangular
    C = gemm(∅, ∅, trmm(A, trmm(X, B));
end
```

Algorithm 6: Algorithm for computing a GEMM-rich matrix product.

$(flops_{1A}+flops_{1B})$ and $\mathrm{op}(A)(X\mathrm{op}(B))$ $(flops_{2A}+flops_{2B})$ are compared. The operation which requires the least amount of flops is then chosen. Small trapezoidal matrices are treated as dense matrices, as the performance gain from handling the triangular and the rectangular parts separately is penalized by the extra function call and matrix setup. Subroutine setup costs are minimized by the use of our superscalar GEMM routine for problems which fit into level 1 cache, i.e., problems solved by the **tr*** sub-system solvers. In the algorithm, this is controlled by the parameter $blks$, i.e., the same parameter that is used in the recursive algorithms to switch from the recursive blocked algorithms to the sub-system solvers. In the implementation, fix-up code for handling any 2×2 diagonal blocks of the quasi-triangular matrices is implemented using level 1 BLAS routines.

For the symmetric case, $\tilde{C} \leftarrow C - AXA^T$, $X = X^T$ and $C = C^T$, the SLICOT MB01RD subroutine is used [39]. It reduces the complexity from $2M^3$ to $\frac{3}{2}M^3$, by calculating the upper triangular part of the symmetric matrix $\mathrm{triu}(\tilde{C}) = V + \mathrm{triu}(B) + \mathrm{stril}(B)^T + \mathrm{diag}(V) + \mathrm{diag}(\mathrm{triu}(B))$, where $B = ATA^T$, $T = \mathrm{triu}(X) - \frac{1}{2}\mathrm{diag}(X)$, and $V = \mathrm{triu}(C) - \frac{1}{2}\mathrm{diag}(C)$, and triu, stril, and diag denote the upper triangular, the strictly lower triangular, and the diagonal parts, respectively. The MB01RD routine is implemented using level 3 BLAS routines and requires no workspace.

However, for some problems, the complexity can be lowered even more, at the cost of extra workspace. In **rtrglydt**, Algorithm 5, the temporary result $W = A_{12}X_{22}$ in the operation $C_{12} = \mathbf{axb}(-A_{12}X_{22}A_{22}^T)$ can be reused in the operation $C_{11} = \mathbf{axb}(-A_{12}X_{22}A_{22}^T)$. Analogously for the E matrix. While this triples the workspace needed, it lowers the overall complexity of the algorithm from $\frac{25}{6}N^3$ to $\frac{23}{6}N^3$.

4.2 Impact and design of superscalar kernel solvers

In Part I, we describe the strategy behind the kernel solvers in detail. While they perform only a small fraction of the total amount of floating point operations, they can have a huge impact on the overall performance (see Section 5.1 in [22]).

The objective for the superscalar kernel solvers is performance as well as accuracy. In order to achieve high performance, various optimization techniques are used. To solve a small sub-system we use LU factorization with partial pivoting on the Kronecker-product Z-matrix representation and overflow guarding. By using partial pivoting, the entire solver can be completely unrolled without the resulting code becoming too large. Overflow guarding is used to signal ill-conditioning. If a good pivot element cannot

Algorithm 7: axb

Input: A $(M \times P)$, X $(P \times Q)$, B $(Q \times N)$ and C $(M \times N)$. A is dense or upper-trapezoidal, B is dense or lower-trapezoidal. W ($\max(MQ, PN)$ elements), temporary work buffer. $blks$, block size that specifies when to switch to superscalar GEMM with low call overhead instead of level 3 BLAS operations.

Output: C $(M \times N)$, the result of the matrix product $C = C + AXB$.

function $[C] = \mathbf{axb}(A, X, B, C, W, blks)$
if *A dense* **then**
  $flops_{1A} = 2MPQ$; $flops_{2B} = 2MPN$;
elseif *A triangular or trapezoidal*
  $flops_{1A} = M^2Q + 2M(P-M)Q$; $flops_{2B} = M^2N + 2M(P-M)N$;
end
if *B dense* **then**
  $flops_{1B} = 2MQN$; $flops_{2A} = 2PQN$;
elseif *B triangular or trapezoidal*
  $flops_{1B} = MN^2 + 2MN(Q-N)$; $flops_{2A} = PN^2 + 2PN(Q-N)$;
end

if $flops_{1A} + flops_{1B} < flops_{2A} + flops_{2B}$ **then**
  if *A dense* **or** $\max(M, P, Q) < blks$ **then**
    $W = \mathbf{gemm}(A, X, \emptyset)$;
  else
    $W = X(1:M, 1:Q)$;
    $W = \mathbf{trmm}(A(1:M, 1:M), W)$;
    $W = \mathbf{gemm}(A(1:M, M+1:P), X(M+1:P, 1:Q), W)$;
  end
  if *B dense* **or** $\max(M, Q, N) < blks$ **then**
    $C = \mathbf{gemm}(W, B, C)$;
  else
    $W(1:M, 1:N) = \mathbf{trmm}(W(1:M, 1:N), B(1:N, 1:N))$;
    $C = C + W(1:M, 1:N)$;
    $C = \mathbf{gemm}(W(1:M, N+1:Q), B(N+1:Q, 1:N), C)$;
  end
else
  if *B dense* **or** $\max(P, Q, N) < blks$ **then**
    $W = \mathbf{gemm}(X, B, \emptyset)$;
  else
    $W = X(1:P, 1:N)$;
    $W = \mathbf{trmm}(W, B(1:N, 1:N))$;
    $W = \mathbf{gemm}(X(1:N, N+1:Q), B(N+1:Q, 1:N), W)$;
  end
  if *A dense* **or** $\max(M, P, N) < blks$ **then**
    $C = \mathbf{gemm}(A, W, C)$;
  else
    $W(1:M, 1:N) = \mathbf{trmm}(A(1:M, 1:M), W(1:M, 1:N))$;
    $C = C + W(1:M, 1:N)$;
    $C = \mathbf{gemm}(A(1:M, M+1:P), W(M+1:P, 1:N), C)$;
  end
end

Algorithm 7: Algorithm for computing an **axb** operation.

be found, the partial pivoting solver is aborted, and the system is solved using complete pivoting and thereby trading performance for the stability and accuracy of the algorithm. The single most important effect comes from the complete inlining. Instead of calling LAPACK and BLAS routines, these operations are performed by the subroutine itself. While this procedure does not lead to any good code reuse, it would be devastating for performance to call LAPACK and BLAS for doing operations for the very short vectors involved in these operations. Instead, our approach is to use one single routine which constructs the Kronecker product Z-matrix, factorizes it, and does the backward and forward substitutions. This eliminates the calling overhead and leads to a much higher register reuse.

In a recursive splitting, $\pm$-sign variants of a matrix equation may appear. For instance, $AXB^T - CXD^T = E$ and $AXB^T + CXD^T = E$ must be solved in the recursive blocked generalized Lyapunov algorithms. To achieve high performance, we have developed $\pm$-sign variants of the **trgsyl** kernel and others.

4.3 Parallelization issues

In Part I [22], two different ways of parallelization of the recursive algorithms for SMP machines are discussed. The first way is by making use of the GEMM-richness of the recursive algorithms and simply use an SMP-aware implementation. While this is by far the easiest way, it will only parallelize the updates, not the solves. The second way is to use the task parallelism in the recursion tree. For example, in Case 3 of the **rtrsydt** algorithm, where both M and N are split, the block matrices X_{12} and X_{21} can be solved for concurrently. Also, some of the updates can be done in parallel. In the **rtrsydt** algorithm, this applies to the updates to C_{12} and C_{21}. However, the performance does not always improve with this parallelization, sometimes it even deteriorates. This is due to the fact that the updates themselves call SMP BLAS subroutines. Hence, the parallelization of the updates put extra pressure on the thread scheduler and memory buses, which may outweigh the extra amount of parallelism.

5 Performance Results of Recursive Blocked Algorithms

The recursive blocked algorithms for solving two-sided triangular matrix equations have been implemented in Fortran 90, using the facilities for recursive calls of subprograms, dynamic memory allocation and threads. In this section, sample performance results of these implementations executing on IBM Power, MIPS and Intel Pentium processor-based systems are presented. We have selected a few processors representing different vendors and different architecture characteristics and memory hierarchies. The details of the processors, memory subsystems, and operating systems are all described in Part I [22]. The only difference from Part I is the Power3 machine, which is running at 375 MHz, giving each processor a theoretical peak rate of 1500 Mflops/s. It has a level 1 data cache of 64 kB and a level 2 cache of 4 MB. The performance results (measured in Mflops/s) are computed using the flopcounts presented in Table 1 of Section 3.6. Most of the results are displayed in graphs, where the performance of different variants of the recursive blocked implementations are visualized together with existing routines in the state-of-the-art SLICOT library [39]. Moreover, the relative performance (measured as speedup) between different algorithms including sequential as well as parallel implementations are presented in tables. Most of the results presented should be quite self-explanatory. Therefore, our discussion is restricted to significant or unexpected differences between the implementations executing on different computing platforms. The accuracy of the results computed by our recursive blocked algorithms are overall very good and similar to the accuracy obtained by the corresponding SLICOT routines. For benchmark problems see [30, 31, 39].

5.1 Triangular discrete-time Sylvester equation

In Figure 1, we show performance graphs for different algorithms and implementations, executing on IBM Power3 and Intel Pentium III processor-based systems, for solving triangular discrete-time Sylvester equations. In Table 3, measured timings and associated speedup results are presented.

The SLICOT SB04PY implements an explicit Bartels-Stewart solver and is mainly a level 2 routine, which explains its poor performance behavior. Our recursive blocked **rtrsydt** shows between a two-fold and a 118-fold speedup with respect to SLICOT SB04PY and an additional speedup up to 2.14 on a four processor Power3 node for large enough problems.

	IBM Power3					Intel Pentium III				
	Time (sec)		Speedup			Time (sec)		Speedup		
$M = N$	A	B	B/A	C/B	D/B	A	B	B/A	C/B	D/B
100	1.73e-2	7.41e-3	2.33	0.96	1.16	2.20e-2	2.20e-2	1.00	0.96	1.22
250	6.20e-1	6.93e-2	8.95	1.02	0.98	6.83e-1	2.31e-1	2.96	0.96	1.33
500	2.32e+1	4.60e-1	50.50	1.03	1.48	7.82e+0	1.56e+0	5.00	0.96	1.35
1000	2.44e+2	3.26e+0	74.65	1.45	1.94	7.04e+1	1.10e+1	6.40	1.04	1.48
1500	9.37e+2	1.08e+1	86.66	1.68	2.04	2.78e+2	3.58e+1	7.76	1.09	1.53
2000	2.84e+3	2.41e+1	117.71	1.98	2.14	9.07e+2	8.03e+1	11.29	1.13	1.57

A – SLICOT SB04PY
B – **rtrsydt**
C – B + linking with SMP BLAS
D – C + utilizing explicit // in the recursion tree

Table 3: Performance results for the triangular discrete-time Sylvester equation—IBM Power3 (left) and Intel Pentium III (right). Labels A–D represent different algorithms and implementations.

5.2 Triangular discrete-time Lyapunov equation

In Figure 2 and Table 4, performance and timing results for the triangular discrete-time Lyapunov equation are presented, now using IBM PowerPC 604e and SGI MIPS R10000 processor-based systems. We compare our recursive blocked **rtrlydt** algorithm with the SLICOT SB03MX routine, which mainly is a level 2 implementation as well. As expected, the relative behavior between the different algorithms and implementations follows qualitatively that of the discrete-time Sylvester equation. We remark that the speedup of **rtrlydt** with respect to the SLICOT SB03MX is between 1.9 and 76 and an additional speedup of 2.6 on a four-processor IBM PowerPC 604e node for large enough problems. The results for the MIPS R10000 show between an 1.5-fold to over 28-fold speedup. Due to lack of support for nested multi-threading with OpenMP and incompabilities between pthreads and efficient SMP BLAS, we do not show any parallel performance results for the MIPS processor (see discussion in Section 5.5 in Part I [22]).

5.3 Triangular generalized Sylvester equation

In Figure 3, performance graphs for different algorithms and implementations, executing on IBM Power3 and Intel Pentium III processor-based systems, for solving the triangular generalized Sylvester equation are displayed. In Table 5, we present the measured timings and associated speedup results. We have not found any library or public software for solving triangular generalized Sylvester equations, so the results presented here are for our recursive blocked algorithms only. With one exception (variant B on IBM

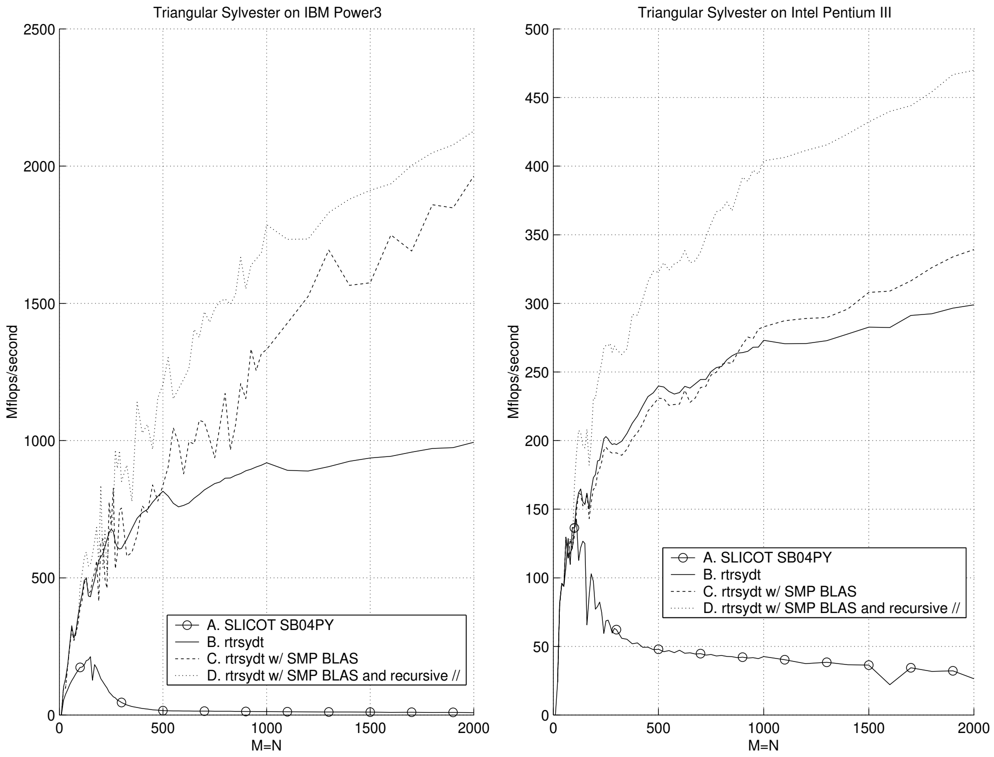

Figure 1: Performance results for the triangular discrete-time Sylvester equation ($M = N$)—IBM Power3 (left) and Intel Pentium III (right).

	IBM PowerPC 604e				MIPS R10000		
	Time (sec)		Speedup		Time (sec)		Speedup
N	A	B	B/A	C/B	A	B	B/A
100	2.43e-2	1.28e-2	1.90	0.90	2.35e-2	1.56e-2	1.51
250	7.58e-1	1.34e-1	5.64	1.37	3.32e-1	1.38e-1	2.40
500	1.59e+1	9.48e-1	16.79	1.82	5.07e+0	9.46e-1	5.36
1000	2.17e+2	6.80e+0	31.88	2.19	1.24e+2	7.45e+0	16.66
1500	1.12e+3	2.19e+1	51.05	2.44	6.21e+2	2.53e+1	24.55
2000	3.74e+3	4.95e+1	75.56	2.62	1.76e+3	6.28e+1	27.95

A – SLICOT SB03MX
B – **rtrlydt**
C – B + linking with SMP BLAS

Table 4: Performance results for the triangular Lyapunov equation—IBM PowerPC 604e (left) and SGI Onyx2 MIPS R10000 (right). Labels A–C represent different algorithms and implementations.

Power3), the results are very similar to the corresponding results of the discrete-time Sylvester equation, which is a special case of the generalized Sylvester equation by choosing $C = D = I_n$.

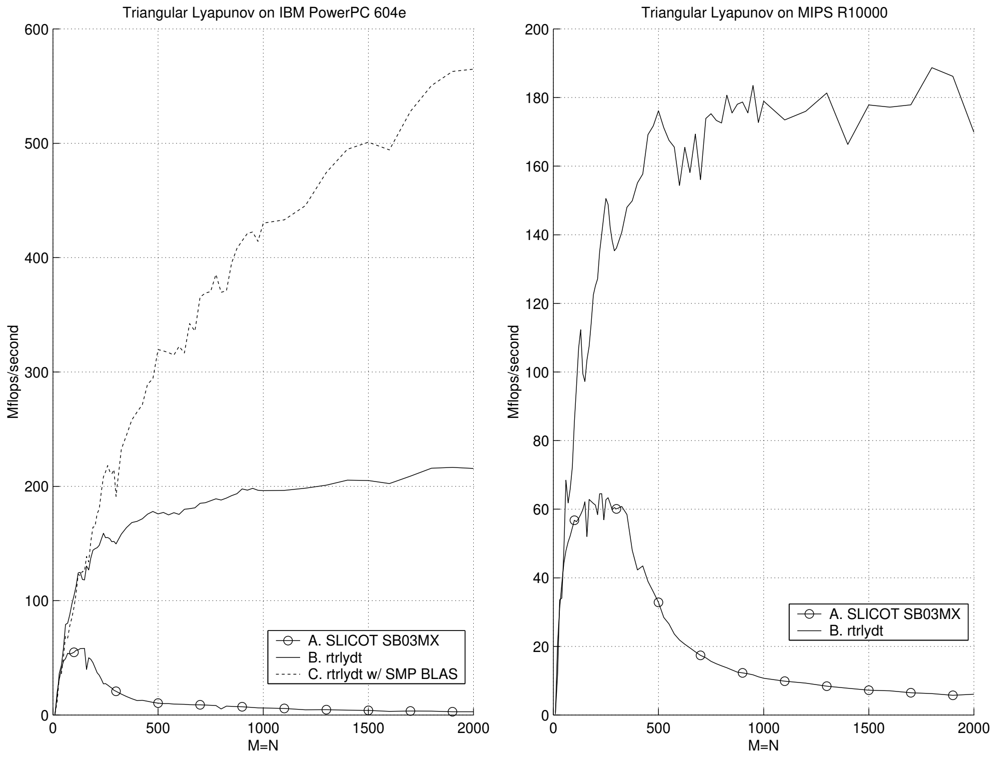

Figure 2: Performance results for the triangular discrete-time Lyapunov equation—IBM PowerPC 604e (left) and SGI Onyx2 MIPS R10000 (right).

	IBM Power3			Intel Pentium III		
	Time (sec)	Speedup		Time (sec)	Speedup	
$M = N$	A	B/A	C/A	A	B/A	C/A
100	1.30e-02	0.96	1.16	4.30e-02	0.98	1.26
250	1.31e-01	0.48	1.26	4.49e-01	0.96	1.34
500	8.91e-01	0.67	1.50	3.07e+00	0.96	1.38
1000	6.42e+00	1.05	1.91	2.18e+01	1.05	1.51
1500	2.13e+01	1.34	2.09	7.10e+01	1.09	1.54
2000	4.77e+01	1.48	2.16	1.60e+02	1.14	1.59

A – **rtrgsyl**
B – A + linking with SMP BLAS
C – B + utilizing explicit // in the recursion tree

Table 5: Performance results for the triangular generalized Sylvester equation—IBM Power3 (left) and Intel Pentium III (right). Labels A–C represent different algorithms and implementations.

5.4 Triangular generalized Lyapunov equations

In Figure 4 and Tables 6 and 7, performance and timing results for the generalized continuous-time and discrete-time Lyapunov equations are presented. We compare the SLICOT routines SG03MX and SG03MY with our recursive blocked solvers **rtrglydt** and **rtrglyct**, respectively. The speedup of **rtrglydt** with respect to the SLICOT SG03MX is between 7 and 43 with a further speedup of 2.6 on a

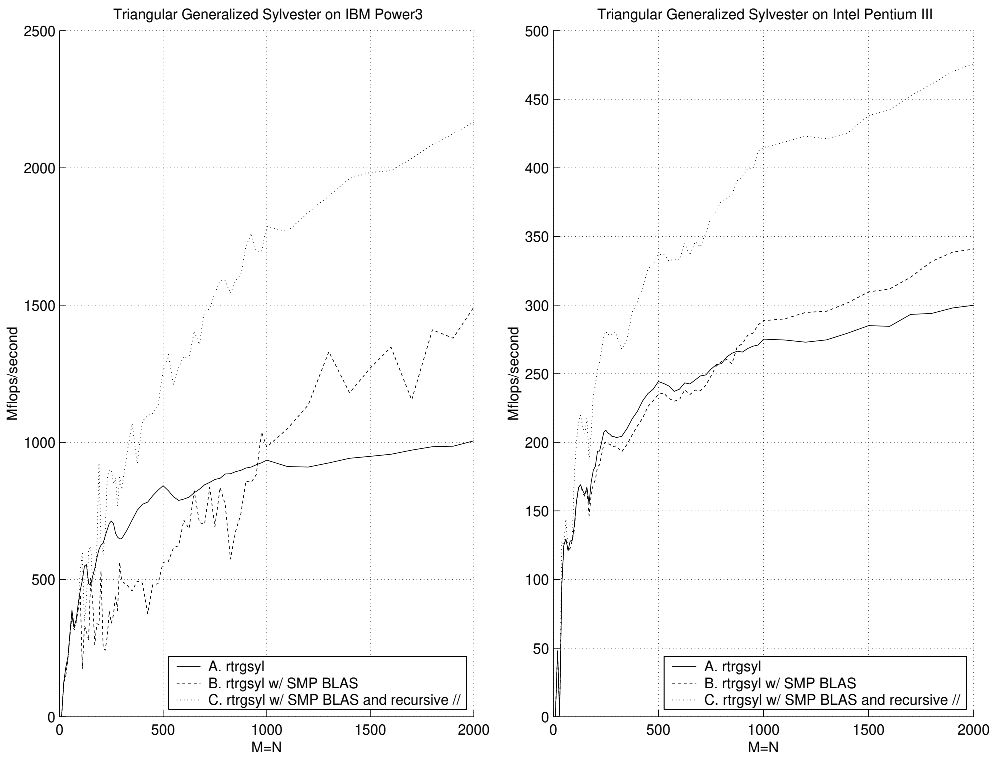

Figure 3: Performance results for the triangular generalized Sylvester equation ($M = N$)—IBM Power3 (left) and Intel Pentium III (right).

four-processor IBM PowerPC 604e node. Similar results are obtained when comparing the continuous-time generalized solvers **rtrglyct** and SLICOT SG03MY. The SLICOT SG03MX and SG03MY mainly perform level 1 and 2 operations, which explain their weak performance.

a)	Discrete time				Continuous time			
	Time (sec)		Speedup		Time (sec)		Speedup	
N	A	B	B/A	C/B	D	E	E/D	F/D
100	1.54e-01	2.19e-02	7.04	0.91	1.53e-01	1.90e-02	8.06	0.96
250	2.86e+00	2.42e-01	11.85	1.45	2.82e+00	2.13e-01	13.26	1.42
500	3.97e+01	1.72e+00	23.11	1.68	3.78e+01	1.56e+00	24.18	1.72
1000	5.24e+02	1.23e+01	42.67	2.21	5.03e+02	1.14e+01	44.23	2.15
1500	-	4.00e+01	-	2.44	-	3.70e+01	-	2.40
2000	-	9.05e+01	-	2.62	-	8.35e+01	-	2.60

A – SLICOT SG03MX
B – **rtrglydt**
C – B + linking with SMP BLAS
D – SLICOT SG03MY
E – **rtrglyct**
F – E + linking with SMP BLAS

Table 6: Performance results for triangular generalized Lyapunov equations—IBM PowerPC 604e. Labels A–F represent different algorithms and implementations.

	Discrete time			Continuous time		
	Time (sec)		Speedup	Time (sec)		Speedup
N	A	B	B/A	D	E	E/D
100	2.52e-01	2.82e-02	8.94	2.49e-01	2.40e-02	10.38
250	2.46e+00	2.53e-01	9.72	2.44e+00	2.29e-01	10.67
500	2.31e+01	1.73e+00	13.30	2.29e+01	1.59e+00	14.45
1000	2.66e+02	1.39e+01	19.20	2.71e+02	1.24e+01	21.81
1500	1.11e+03	4.65e+01	23.88	1.15e+03	4.24e+01	27.20
2000	3.12e+03	1.16e+02	26.86	3.25e+03	1.05e+02	30.95
	A –	SLICOT SG03MX				
	B –	**rtrglydt**				
	D –	SLICOT SG03MY				
	E –	**rtrglyct**				

Table 7: Performance results for triangular generalized Lyapunov equations—SGI Onyx2 MIPS R10000. Labels A–E represent different algorithms and implementations.

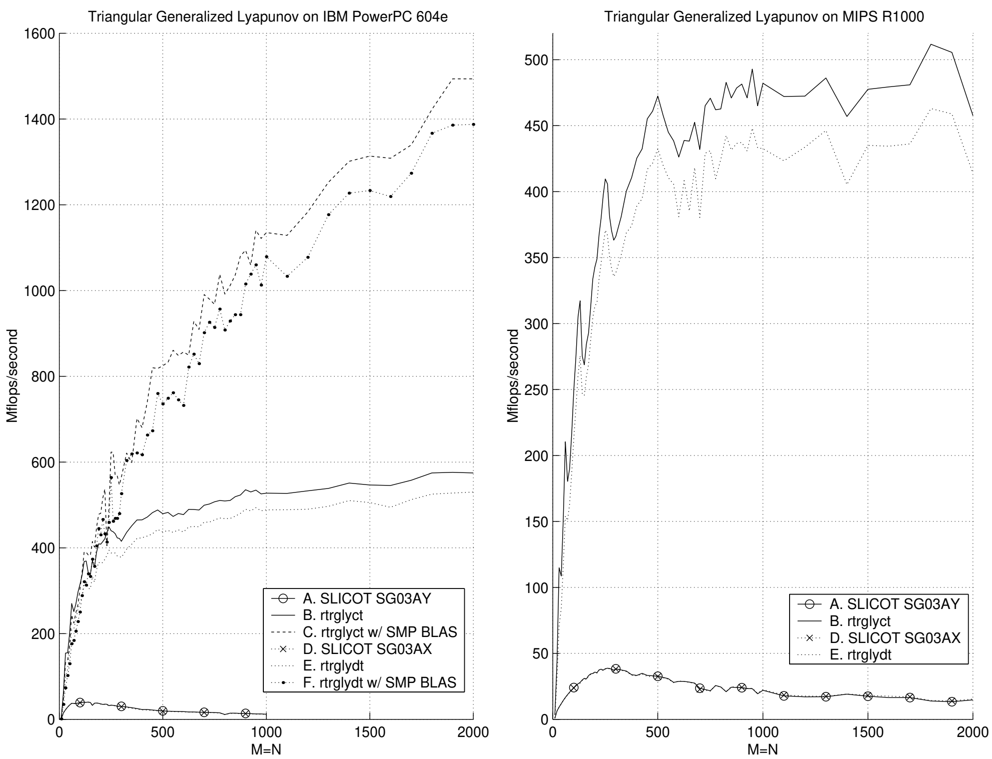

Figure 4: Performance results for the triangular generalized discrete-time and continuous-time Lyapunov equations—IBM PowerPC 604e (left) and SGI Onyx2 MIPS R10000 (right).

5.5 Relative performance of the recursive blocked solvers

We have also investigated the relative performance between the recursive blocked algorithms of the two-sided triangular matrix equations. In Table 8 (upper part), the relative performance of the solvers with respect to **rtrsydt** is displayed. All timings are from using one IBM Power3 processor. In the lower part, the ratios of the dominating terms of the flops counts in Table 2 are shown, and we see that the time and flops ratios agree nicely. We remark that the flops ratio of **rtrglydt** corresponds to the algorithm implemented, which has a lower dominating factor than recorded in Table 2 (see Section 4.1).

	Time	Time relative to **rtrsydt**			
N	**rtrsydt**	**rtrlydt**	**rtrgsyl**	**rtrglydt**	**rtrglyct**
100	0.0075	0.68	1.77	1.22	0.99
250	0.0700	0.60	1.87	1.11	0.97
500	0.4607	0.60	1.93	1.09	0.99
1000	3.2667	0.60	1.96	1.10	1.00
1500	10.8120	0.60	1.97	1.09	1.00
2000	24.1414	0.61	1.98	1.10	1.01

	Flops	Flops relative to **rtrsydt**			
N	**rtrsydt**	**rtrlydt**	**rtrgsyl**	**rtrglydt**	**rtrglyct**
	$3.5N^3$	0.60	2.0	1.10 (1.19)	1.0

Table 8: Relative performance results for recursive blocked solvers.

5.6 Impact on solving unreduced two-sided generalized matrix equations

As for the one-sided matrix equations [22], we have investigated the impact of the choice of triangular matrix equation solver on the time to solve unreduced two-sided matrix equations. Although the transformation of an unreduced matrix equation to a triangular counterpart and the backtransformation of the solution are normally at least as costly operations (measured in flops) as the triangular solve, the impact of using our recursive triangular solvers can be substantial.

For illustration, we first use the SLICOT routine SG03AD which solves unreduced generalized discrete-time Lyapunov equations. SG03AD also has an option to compute an estimate of the separation Sep[GLYDT] (see Section 2).

In Table 9a, timings for the SG03AD routine are displayed for problem sizes ranging from 50 to 1000 using two different triangular matrix equation solvers. These solvers are SG03AX provided in SLICOT, which implements a variant of the Bartels-Stewart method calling BLAS [36], and our recursive blocked **rtrglydt** algorithm. In the second column, the total times for solving an unreduced system with SG03AX as the triangular solver are displayed. This includes the time for the generalized Schur factorization and backtransformation of the solution. In the fourth and fifth columns, similar results are displayed when SG03AX is replaced by the **rtrglydt** routine. We see up to 90 % speedup for the problem sizes considered. The numbers in the lower part of Table 9a also include the time for computing a 1-norm-based-estimate for Sep[GLYDT]. The condition estimation process includes repeated calls (typically five) to the generalized triangular solver, and as expected we see a four to five-fold speedup when using our recursive solver.

In Table 9b, the corresponding results are displayed when we have replaced the LAPACK routines DGGHRD and DHGEQZ by Dackland-Kågström's blocked Hessenberg-triangular reduction and QZ algorithms [5] for transforming the regular pair (A, E) to generalized Schur form. For large enough problems, this gives another factor of two speedup.

We have also compared the routine SYLG [8, 9] which implements a variant of the Hessenberg-Schur method [10] for solving unreduced generalized Sylvester equations (3). In Table 10a, timings for SYLG and a generalized Sylvester solver calling our recursive blocked solver are displayed. The upper part shows results when the size of (A, C) and (B, D) are the same ($M = N$) and the lower part shows results when $M = 10N$. SYLG calls the original LINPACK QZ routines (QZHESQ, QZITG, and QZVALG) for the initial reductions, while **rtrgsyl** is preceded by calls to the LAPACK routines DGGHRD and DHGEQZ

a) N	SG03AD using SG03AX Total time	Solver part	SG03AD using **rtrglydt** Total time	Solver part	Speedup	Job
50	0.0277	49.9 %	0.0185	20.1 %	1.50	X
100	0.180	51.2 %	0.0967	9.0 %	1.86	X
250	2.89	46.8 %	1.62	4.7 %	1.79	X
500	59.0	42.3 %	34.5	1.5 %	1.71	X
750	303.4	42.0 %	177.5	0.9 %	1.71	X
1000	646.6	44.6 %	361.8	1.0 %	1.79	X
50	0.117	87.6 %	0.0263	45.6 %	4.44	X+Sep
100	0.709	87.3 %	0.152	40.6 %	4.68	X+Sep
250	9.98	84.5 %	2.08	25.4 %	4.81	X+Sep
500	178.6	80.9 %	37.8	9.4 %	4.73	X+Sep
750	924.1	80.9 %	184.4	4.5 %	5.01	X+Sep
1000	2076.6	82.7 %	391.8	8.4 %	5.30	X+Sep

b) N	SG03AD using SG03AX Total time	Solver part	SG03AD using **rtrglydt** Total time	Solver part	Speedup	Job
50	0.0306	44.7 %	0.019	11.2 %	1.61	X
100	0.213	43.2 %	0.129	6.5 %	1.65	X
250	3.18	42.5 %	1.90	3.9 %	1.67	X
500	41.8	59.7 %	17.3	2.9 %	2.41	X
750	187.1	68.2 %	61.1	2.7 %	3.06	X
1000	428.9	67.2 %	144.1	2.5 %	2.98	X
50	0.120	85.4 %	0.0285	39.1 %	4.19	X+Sep
100	0.741	83.5 %	0.166	26.2 %	4.47	X+Sep
250	10.3	82.1 %	2.69	31.0 %	3.83	X+Sep
500	161.3	89.5 %	20.0	15.3 %	8.06	X+Sep
750	807.7	92.6 %	67.9	12.1 %	11.89	X+Sep
1000	1857.4	92.4 %	174.0	18.9 %	10.68	X+Sep

Table 9: In a), the total execution times for solving a unreduced generalized discrete-time Lyapunov equation (upper part) and computing an estimate of Sep[GLYDT] (lower part) are displayed for two different triangular generalized Lyapunov solvers. Here, the reduction to generalized Schur form of (A, E) is performed using the LAPACK routines DGGHRD and DHGEQZ. In b), the corresponding results are displayed when using the blocked algorithms described in [5] for the initial condensing operation on (A, E).

for the condensing operations of (A, C) and (B, D). In Table 10b, the corresponding results are displayed when using the blocked algorithms described in [5] for the initial reductions in SYLG and in combination with **rtrgsyl**. Also for this matrix equation, we see a substantial impact in using our recursive blocked algorithm (almost a factor 3 speedup for $N = 1000$). We remark that we only see small benefits (or no benefits at all) in using our recursive algorithm for small problem sizes or when $M = 10N$. This result is not due to the properties of the recursive solvers, but rather to the properties of the Hessenberg-Schur method, where only one of the matrices of A and D is reduced to Schur form, and the other is reduced to Hessenberg form. The Hessenberg-Schur algorithm is best suited for cases when $M \gg N$ or $M \ll N$, when only the smaller of the matrix pairs is reduced to quasi-upper-triangular Schur form.

a)		SYLG		Reduction+**rtrgsyl**		
M	N	Total time	Solver part	Total time	Solver part	Speedup
50	50	0.0448	38.5 %	0.0386	30.4 %	1.16
100	100	0.315	39.1 %	0.191	6.8 %	1.65
250	250	5.43	33.7 %	3.34	3.9 %	1.63
500	500	129.0	20.5 %	63.3	1.4 %	2.04
750	750	556.4	24.4 %	251.1	1.2 %	2.22
1000	1000	1386.3	25.5 %	613.9	1.0 %	2.26
50	5	0.0088	19.2 %	0.0133	3.4 %	0.66
100	10	0.0638	18.8 %	0.0871	1.2 %	0.73
250	25	1.02	17.5 %	1.59	0.5 %	0.64
500	50	24.5	10.4 %	30.9	0.2 %	0.79
750	75	114.1	11.7 %	126.4	0.1 %	0.90
1000	100	286.1	12.1 %	302.4	0.1 %	0.95

b)		SYLG		Reduction+**rtrgsyl**		
M	N	Total time	Solver part	Total time	Solver part	Speedup
50	50	0.0443	38.3 %	0.0363	7.0 %	1.22
100	100	0.311	39.5 %	0.250	5.2 %	1.25
250	250	4.74	38.7 %	3.82	3.4 %	1.24
500	500	85.5	31.0 %	34.1	2.6 %	2.51
750	750	352.0	38.6 %	121.7	2.4 %	2.89
1000	1000	870.7	40.7 %	290.9	2.2 %	2.99
50	5	0.0094	17.7 %	0.0162	1.7 %	0.58
100	10	0.0500	24.0 %	0.117	0.9 %	0.43
250	25	0.779	22.9 %	1.81	0.4 %	0.43
500	50	15.4	16.5 %	16.3	0.3 %	0.94
750	75	67.4	19.7 %	57.2	0.3 %	1.18
1000	100	163.2	21.3 %	139.9	0.2 %	1.17

Table 10: In a), the total execution times for solving a unreduced generalized Sylvester equation are displayed for two different generalized Sylvester solvers ($M = N$ upper part, $M = 10N$ lower part). Here, the original reduction/QZ package is used (QZHESQ, QZITG, and QZVALG for SYLG [8, 9] and DGGHRD and DHGEQZ for **rtrgsyl**, respectively). In b), the corresponding results are displayed when using the blocked algorithms described in [5] for the initial reduction of matrix pairs to generalized Schur form.

6 Summary and Some Conclusions

The performance results verify that our recursive approach is also very efficient for solving two-sided triangular matrix equations on today's hierarchical memory computer systems. Despite the quite large flops penalties of the recursive blocked algorithms they outperform the standard algorithms for large enough problems. For example, solving discrete-time Sylvester and Lyapunov equations with coefficient matrices of size 2000×2000 take around one hour using current routines in the SLICOT library [39], while the solution times of our recursive blocked algorithms are less than one minute for the same problems. This is partly due to the difference in the data reference patterns of the algorithms. Our recursive blocked algorithms match automatically the memory hierarchy of a target machine and provide good data locality. The recursion is ended when the remaining subproblems to be solved are smaller than a given block size, which is the only architecture-dependent parameter in our algorithms. As described in the Part I paper on one-sided matrix equations [22], we develop new high-performance superscalar kernels for solving the remaining small-sized triangular matrix equations and light-weight GEMM operations, which implies that a larger part of the total execution time is spent in high-performance GEMM operations. Moreover, we have developed optimized two-sided matrix product kernels that take any symmetry properties as well as any triangular or trapezoidal structure of the matrices into account. In order to maximize the performance of these two-sided matrix product operations, e.g., AXB^T, we make use of optimized level 3 BLAS, which by default require some temporary storage. Altogether, this leads to much faster algorithms for solving reduced as well as unreduced two-sided and generalized Sylvester and Lyapunov matrix equations and different associated condition estimation problems.

Acknowledgements

This research was conducted using the resources of the High Performance Computing Center North (HPC2N) and PDC–Parallelldatorcentrum at KTH, Stockholm. We also thank Fred Gustavson and the colleagues in the Umeå HPC and Parallel Computing Research Group for stimulating and fruitful discussions.

Financial support has been provided by the Swedish Research Council for Engineering Sciences under grant TFR 98-604.

References

[1] E. Anderson, Z. Bai, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen. *LAPACK Users' Guide*, Third Edition. SIAM Publications, Philadelphia, 1999.

[2] Z. Bai, J. Demmel and A. McKenney. On Computing Condition Numbers for the Nonsymmetric Eigenproblem, *ACM Trans. Math. Software*, 19:202–223, 1993.

[3] R.H. Bartels and G.W. Stewart. Algorithm 432: Solution of the Equation $AX + XB = C$, *Comm. ACM*, 15(9):820–826, 1972.

[4] K.-W.E. Chu, The solution of the matrix equation $AXB - CXD = Y$ and $(YA - DZ, YC - BZ) = (E, F)$. *Linear Algebra Appl.*, 93:93–105, 1987.

[5] K. Dackland and B. Kågström. Blocked Algorithms and Software for Reduction of a Regular Matrix Pair to Generalized Schur Form. *ACM Trans. Math. Software*, Vol. 25, No. 4, 425–454, 1999.

[6] 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.

[7] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Soft.*, 16(1):18–28, 1990.

[8] J.D. Gardiner, A.J. Laub, J.J. Amato, and C.B. Moler. Solution of the Sylvester Matrix Equation $AXB^T + CXD^T = E$, *ACM Trans. Math. Software*, 18:223–231, 1992.

[9] J.D. Gardiner, M.R. Wette, A.J. Laub, J.J. Amato, and C.B. Moler. A Fortran 77 Software Package for Solving the Sylvester Matrix Equation $AXB^T + CXD^T = E$, *ACM Trans. Math. Software*, 18:232–238, 1992.

[10] G. Golub, S. Nash, and C. Van Loan. A Hessenberg-Schur Method for the Matrix Problem $AX + XB = C$. *IEEE Trans. Autom. Contr.*, AC-24(6):909–913, 1979.

[11] G. Golub, and C. Van Loan. *Matrix Computations*, Third Ed., Johns Hopkins University Press, Baltimore, 1996.

[12] F. Gustavson. Recursion leads to automatic variable blocking for dense linear algebra. *IBM J. Res. Develop*, 41(6):737–755, November 1997.

[13] F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling. Recursive Blocked Data Formats and BLAS's for Dense Linear Algebra Algorithms. In Kågström et al. (eds), *Applied Parallel Computing, PARA'98*, Lecture Notes in Computer Science, Vol. 1541, pp 195–206, Springer-Verlag, 1998.

[14] F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling. Superscalar GEMM-based Level 3 BLAS – The On-going Evolution of a Portable and High-Performance Library. In Kågström et al. (eds), *Applied Parallel Computing, PARA'98*, Lecture Notes in Computer Science, Vol. 1541, pp 207-215, Springer-Verlag, 1998.

[15] F. Gustavson and I. Jonsson. Minimal-storage high-performance Cholesky factorization via blocking and recursion, *IBM J. Res. Develop*, 44(6):823–849, November 2000.

[16] W.W. Hager. Condition Estimates, *SIAM J. Sci. Stat. Comp.*, 5:311–316, 1984.

[17] S.J. Hammarling. Numerical Solution of the Stable, Non-negative Definite Lyapunov Equation, *IMA J. Num. Anal.*, 2:303–323, 1982.

[18] N.J. Higham. Fortran Codes for Estimating the One-Norm of a Real or Complex Matrix with Applications to Condition Estimation, *ACM Trans. Math. Software*, 14:381–396, 1988.

[19] N.J. Higham. Perturbation Theory and Backward Error for $AX - XB = C$, *BIT*, 33:124–136, 1993.

[20] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, 1996.

[21] I. Jonsson and B. Kågström. Parallel Triangular Sylvester-type Matrix Equation Solvers for SMP Systems using Recursive Blocking, in *Applied Parallel Computing: New Paradigms for HPC Industry and Academia*, Lecture Notes in Computer Science, vol. 1947, pp 64–74, Springer Verlag, 2001.

[22] I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part I: One-Sided and Coupled Sylvester-Type Equations, Report UMINF 01.05, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden, 2001 (submitted to *ACM TOMS*).

[23] B. Kågström. A Perturbation Analysis of the Generalized Sylvester Equation $(AR-LB, DR-LE) = (C, F)$, *SIAM J. Matrix Anal. Appl.*, 15(4):1045–1060, 1994.

[24] B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski, *Applied Parallel Computing. Large Scale Scientific and Industrial Problems.*, Lecture Notes in Computer Science, No. 1541, Springer-Verlag, Berlin, 1998.

[25] 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.

[26] 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.

[27] B. Kågström and P. Poromaa. Distributed and Shared Memory Block Algorithms for the Triangular Sylvester Equation with sep^{-1} Estimator, *SIAM J. Matrix Anal. and Appl.*, 13(1):90–101, 1992.

[28] B. Kågström and P. Poromaa. LAPACK–Style Algorithms and Software for Solving the Generalized Sylvester Equation and Estimating the Separation between Regular Matrix Pairs. *ACM Trans. Math. Software*, 22(1):78–103, 1996.

[29] B. Kågström and L. Westin. Generalized Schur methods with condition estimators for solving the generalized Sylvester equation. *IEEE Trans. Autom. Contr.*, 34(4):745–751, 1989.

[30] D. Kressner, V. Mehrmann and T. Penzl. CTLEX–a Collection of Benchmark Examples for Continuous-Time Lyapunov Equations, *SLICOT Working Note* 1999-6, 1999.

[31] D. Kressner, V. Mehrmann and T. Penzl. DTLEX–a Collection of Benchmark Examples for Discrete-Time Lyapunov Equations, *SLICOT Working Note* 1999-7, 1999.

[32] A. Lindkvist. High-performance recursive BLAS kernels using new data formats for the QR factorization. *Master Thesis*, Report UMNAD-325.00, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden, 2000.

[33] C. B. Moler and G. W. Stewart. An Algorithm for Generalized Matrix Eigenvalue Problems, *SIAM J. Num. Anal.*, 10:241–256, 1973.

[34] B. Nichols, D. Buttlar, J. Proulx Farrell, and J. Farrell. *Pthreads Programming: A POSIX Standard for Better Multiprocessing*, O'Reilly & Associates, ISBN:1565921151, 1996.

[35] OpenMP Fortran Application Program Interface, Version 2.0, November 2000, www.openmp.org/specs/

[36] T. Penzl. Numerical Solution of Generalized Lyapunov Equations, *Advances in Comp. Math.*, 8:33–48, 1998.

[37] P. Poromaa. *High Performance Computing: Algorithms and Library Software for Sylvester Equations and Certain Eigenvalue Problems with Applications in Condition Estimation*, PhD Thesis UMINF-97.16, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden, June, 1997.

[38] P. Poromaa. Parallel Algorithms for Triangular Sylvester Equations: Design, Scheduling and Scalability Issues. In Kågström et al. (eds), *Applied Parallel Computing, PARA'98*, Lecture Notes in Computer Science, Vol. 1541, pp 438–446, Springer-Verlag, 1998.

[39] SLICOT library and the Numerics in Control Network (NICONET) website: www.win.tue.nl/niconet/index.html

[40] G.W. Stewart and J-G. Sun. *Matrix Perturbation Theory*, Academic Press, 1990.